REFERENCE MANUAL

Editors: Bonnie Gacnik
Bob Lackman

GENeral Scientific Data PROcessor

SCIENTIFIC COMPUTING DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO
GENPRO DRIVER Operation
REFERENCE MANUAL

Editors: Bonnie Gacnik
Bob Lackman

GENeral Scientific Data PROcessor

SCIENTIFIC COMPUTING DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO
## DRIVER CONTROLS

### CONTENTS

1. **INTRODUCTION** ........................................... 3-DRIVER.1  
2. **GENERAL CONTROLS** ........................................ 3-DRIVER.2  
   2.1 **BEGIV** ............................................... 3-DRIVER.2  
   2.2 **BEGSNP** ............................................... 3-DRIVER.4  
   2.3 **BITCHR** ............................................... 3-DRIVER.8  
   2.4 **BANTSU** ............................................... 3-DRIVER.9  
   2.5 **DATLOG** ............................................... 3-DRIVER.10  
   2.6 **DEBUG** ............................................... 3-DRIVER.11  
   2.7 **DELIV** ............................................... 3-DRIVER.11  
   2.8 **DUMP** ................................................ 3-DRIVER.12  
   2.9 **ENDIV** ............................................... 3-DRIVER.13  
   2.10 **ENDSNP** ............................................. 3-DRIVER.16  
   2.11 **FLUSHP** ............................................. 3-DRIVER.20  
   2.12 **GAPVAL** ............................................. 3-DRIVER.22  
   2.13 **IABORT** ............................................. 3-DRIVER.22  
   2.14 **ITYPOP** (Derived) .................................... 3-DRIVER.24  
   2.15 **IVDIM** .............................................. 3-DRIVER.26  
   2.16 **KCHECK** ............................................. 3-DRIVER.27  
   2.17 **KDUMP** .............................................. 3-DRIVER.28  
   2.18 **KERR** ................................................ 3-DRIVER.29  
   2.19 **KICHE** ............................................... 3-DRIVER.30  
   2.20 **KFILE** ............................................... 3-DRIVER.31  
   2.21 **KPRINT** ............................................. 3-DRIVER.32  
   2.22 **KPROC** ............................................... 3-DRIVER.33  
   2.23 **KUNIT** ............................................... 3-DRIVER.34  
   2.24 **LABIV** ............................................... 3-DRIVER.36  
   2.25 **LIMREC** ............................................. 3-DRIVER.37  
   2.26 **MAXCHR** ............................................. 3-DRIVER.38  
   2.27 **NAMEOP** (Derived) ................................... 3-DRIVER.38  
   2.28 **NLMLIB** ............................................. 3-DRIVER.38  
   2.29 **NAMPD** ............................................... 3-DRIVER.39  
   2.30 **NUMBIT** ............................................. 3-DRIVER.39  
   2.31 **NUMCYC** ............................................. 3-DRIVER.41  
   2.32 **NUMOPT** ............................................. 3-DRIVER.43  
   2.33 **OVRPLO** ............................................. 3-DRIVER.45  
   2.34 **OVRLLP** ............................................. 3-DRIVER.46  
   2.35 **PDAT** ................................................ 3-DRIVER.47  
   2.36 **PRINT** .............................................. 3-DRIVER.48  
   2.37 **PROJECT** ............................................ 3-DRIVER.49  
   2.38 **PRTIME** ............................................. 3-DRIVER.49  
   2.39 **RECSIZ** ............................................. 3-DRIVER.50  
   2.40 **SCALIV** ............................................. 3-DRIVER.51  
3. **VARIABLE CONTROLS** ....................................... 3-DRIVER.53  
   3.1 **DGET** (Derived) ...................................... 3-DRIVER.53  
   3.2 **DINPUT** (Derived) ..................................... 3-DRIVER.54
3.3 IN RATE (Derived) ........................................... 3-DRIVER 55
3.4 NAMVAR (Derived) ........................................ 3-DRIVER 57
3.5 RATE ......................................................... 3-DRIVER 57
1. **INTRODUCTION**

The DRIVER Operation is the data manager for the entire GENPRO processor. The DRIVER handles all communication with the other Operations, performs their space array allocations, loads global common blocks, provides data access and store indices to the Operations, and governs general data flow.

When necessary, the DRIVER performs a Case as a series of two or more Job Steps. When this occurs an INDR(ive) Operation is loaded as the first Operation in the Operation Sequence and an OUTD(rive) Operation is loaded as the last Operation in the Sequence. The output of one Job Step becomes the input of the next Job Step, etc.

There are three basic types of Controls associated with the DRIVER:

- Those which relate to data management for this Case.
  
  These include such items as the run interval, DRIVER informational and debug displays, and machine dependent constants.

- Those which relate to potential Job Steps.
  
  Job Steps automatically occur only when needed. The need arises when the amount of data to be processed in a Case is large relative to the data array allocations made in the version of the GENPRO code being executed.

  The DRIVER can Job Step from the initial values of the related Controls. The only potential problem is that FORTRAN units 9 and 10 are chosen as the initial values for disk storage of Job Step outputs. If this represents a conflict refer to Control KUNIT below.

- Those which relate to Value Sharing convenience.
  
  Controls such as BEGSNP, ENDSNP, FLUSHP, and PROJECT are not used by the DRIVER directly, but are included as DRIVER Controls as a matter of GENPRO setup convenience.
2. GENERAL CONTROLS

2.1 BEGIV

Purpose: Specifies the value of the Independent Variable at which to begin processing.

Form:

\[
\text{BEGIV} = (\text{real1}, \ldots, \text{realN})
\]

where the number of Elements in each Group, N, is defined by Control IVDIM, and where each Element represents one component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then,

\[
\text{BEGIV} = (3., 51., 0.)
\]

would result in hour 3, minute 51, and second 0 being the first value of the Independent Variable to be processed.

Initialized Value: \( \text{BEGIV} = (0., 0., 0.) \)

Diagnostics:

***** OPERATION NO. 1 *****

INDR

***** OPERATION NO. 1 *****

MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2

CONTROL 'BEGIV' MISSING FROM CONTROL ARRAY

FATAL******** FATAL******** FATAL******** FATAL******** FATAL********

Cause: Control missing from Control Array.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control BEGIV is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

May 1, 1983

Release 1.0
**** OPERATION NO. 1 *** N D R **** OPERATION NO. 1 ***

MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'BEGIV' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******

Cause: The Group length of Control BEGIV is not equal to the number of
dimensioned components of the Independent Variable, $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number.
PD release numbers must match the code release numbers. The number of
Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM
of the associated code release.

**** OPERATION NO. 1 *** N D R **** OPERATION NO. 1 ***

MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'BEGIV' VALUE '$' INCONSISTENT WITH CONTROL 'ENDIV' VALUE '$'
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******

Cause: The value of Control BEGIV is greater than or equal to ENDIV,
the end time of this Case.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for BEGIV and ENDIV.

If the value of Control BEGIV is less than the first value of the
Independent Variable obtained from the data, the value of Control BEGIV
is changed to the first input value and the following message to this
effect is generated.

***** (DRIVER) CASE INFORMATION ***** (DRIVER) CASE INFORMATION *****
*
* THE STARTING VALUE OF THE I.V., BEGIV, HAS BEEN RESET FROM
* THE INPUT VALUE '$' TO THE 1ST OBSERVED VALUE '$'
* ***** DRIVER ***** INFORMATION ***** DRIVER ***** INFORMATION *****

Dependencies: IVDIM, ENDIV

Controls BEGIV and ENDIV work as ordered pairs describing the beginning
and the ending of the processing period of a Case.

For both Controls the number of Elements in a Group is defined by the
DRIVER Control IVDIM.
Comments: The number of Elements in a Group for Control BEGIV is initially defined in the Programmer Directive file for this Operation to be three.

2.2 BEGSNP

Purpose: Marks the beginning of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: BEGSNP = (real1,...,realN),...,real1,...,realN)

where the number of Elements in each Group, N, is defined by Control IVDIM,

and where each Element represents a component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then,

BEGSNP = (3., 51., 0.)

would cause a Snapshot Period to begin at hour 3, minute 51, and second 0.

Initialized Value: BEGSNP = (0.0, 0.0, 0.0)

This value represents the beginning of one Snapshot of an Independent Variable with three components. For a positive valued Independent Variable it also represents a Snapshot which will begin at the first Cycle of the data for any Case.

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'BEGSNP' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
**FATAL** **FATAL** **FATAL** **FATAL** **FATAL** **FATAL**

*Cause: The Group length of Control BEGSNP is not equal to the number of dimensioned components of the Independent Variable, $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number.

PD release numbers must match the code release numbers. The number of
Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM of the associated code release.

***** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL

**Cause:** The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP.

***** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 10 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH 'ENDSNP' VALUE '$'; ASSUMING '$'
****WARNING********WARNING********WARNING********WARNING********WARNING********WARNING

**Cause:** The value of Control BEGSNP is less than the ENDSNP of the previous Snapshot Period.

**Program action:** A warning message is printed.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
INCONSISTENT SNAPSHOT INTERVALS '$' THROUGH '$' HAVE BEEN TURNED OFF.
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL

**Cause:** A previous fatal error was detected in either Control BEGSNP or ENDSNP.

**Program action:** All Snapshot Periods starting with the first one to produce a fatal error condition are deactivated.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAP INTERVAL 'I' 'BEGSNP' = '$' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE.
DID YOU WANT TO SKIP A CYCLE?
****WARNING********WARNING********WARNING********WARNING********WARNING********WARNING

**Cause:** The value of Control BEGSNP is one Cycle larger than the ENDSNP

Release 1.0 May 1, 1983
2.5 DATLOG

Purpose: Gives the number of logical records (Cycles) in one physical record involved in the output of data by the OUTD(rive) Operation when Job Stepping is being done.

Form: DATLOG = integer

where 'integer' is non-negative.

Initialized Value: DATLOG = 0

This initial value assumes that no Job Step restart is to occur. As a Job Step occurs, DATLOG will contain the actual number of logical records in a physical record which is being written (Subroutine OUTD(ive)), or read (Subroutine INDR(ive)).

Diagnostics:

***** OPERATION NO. 1 *****

MESSAGE No. 3 FROM SUBPROGRAM INITID AT SECTION 1.2 EX STAGE IS 2
CONTROL 'DATLOG' MISSING FROM CONTROL ARRAY; ASSUMING '$'

WARNING**********WARNING**********WARNING**********WARNING**********WARNING**********WARNING

Cause: The Control DATLOG was not found in the DRIVER INDR(ive) PD File.

Program action: (1) A warning message is issued, and (2) an assumed value is used.

Fix: Check the release number of the PD File against the release number of the code files for consistency. Also check all Directives for proper ordering and completeness.

***** OPERATION NO. 1 *****

MESSAGE No. 8 FROM SUBPROGRAM INITID AT SECTION 1.2 EX STAGE IS 2
CONTROL 'DATLOG' VALUE '$' UNREASONABLE; ASSUMING '$'

WARNING**********WARNING**********WARNING**********WARNING**********WARNING**********WARNING

Cause: The value is less than or equal to zero (0).

Program action: (1) A warning message is issued, and (2) an assumed value is used.

Fix: If the assumed value is incorrect respecify the correct value.

Dependencies: LIMREC, NUMBIT, KUNIT, RECSIZ, NUMCYC

For a more detailed explanation of Job Stepping refer to the comment section of Control KUNIT.
2.6 DEBUG

Purpose: Specifies verbose trace prints to be activated to help find execution problems. These printouts are especially useful to the GENPRO consultants.

Form: \( \text{DEBUG} = (\text{name1}), (\text{name2}), \ldots, (\text{STOP}) \)

where 'name1', 'name2', \ldots, specify informational displays which are one of:

- CONTROLS The unedited Control Array of each Operation in the Case
- FLOW Data Window flow tracing
- SNAP Snapshot Interval flow tracing
- STOP Indicates the end of list processing

Up to four Groups separated by commas may be assigned, where each Group consists of one name Element from the list above. The last Group must always have the value STOP to indicate the end of the list.

The order of the Groups is irrelevant: all information requested will always be generated in the same sequence. If the value of a Group is misspelled, it is deleted from the list and thereby ignored.

Initialized Value: \( \text{DEBUG} = (\text{NOCONTROLS}), (\text{NOSNAP}), (\text{NOFLOW}), (\text{STOP}) \)

Dependencies: KCHECK

Comments: These printouts for debug assistance are helpful in tracing setup errors and errors introduced by the attempt to develop and interface new codes.

All DEBUG output is written to the FORTRAN unit defined by DRIVER Control KCHECK.

If a requested display does not occur check the assignment to unit KCHECK and the spelling of the option name in DEBUG.

2.7 DELIV

Purpose: Defines the interval of the Independent Variable spanning one Cycle in units of the composite value of the Independent Variable.

Form: \( \text{DELIV} = \text{real} \)

where 'real' is the interval of the Independent Variable spanning one Cycle, in units of the composite value of the Independent Variable.

Release 1.0

May 1, 1983
Initialized Value: DELIV = 1.

Diagnostics:

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'DELIV' MISSING FROM CONTROL ARRAY
*****FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****

Cause: Control missing from Control Array.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control DELIV is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 5 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'DELIV' VALUE '$' INVALID; ASSUMING '$'
*****FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****

Cause: The value of Control DELIV is less than or equal to zero (0). The Independent Variable must increase (positive DELIV) during a Case.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Choose a positive increment of the Independent Variable Cycle. Typically this is the increment of a logical record of the input data.

Dependencies: SCALIV

Comments: The composite and the component forms of the Independent Variable are related by the computational factors given in Control SCALIV.

2.8 DUMP

Purpose: Specifies sections of the SPACE array to be printed. The SPACE array addresses of Control Arrays, Operation storage of local information (SAV), Operation buffers, and arrays where the processed data are
stored are listed as LOCCA, LOCSAV, LOCBUF, and LOCDAT in the output generated by the SIZE option of the PRINT Directive. The lengths of these arrays are also printed as LENCA, LENSAV, LENBUF, and LENDAT under the SIZE option. These SPACE array pointers and lengths are listed for all Operations in the Operation Sequence. Selected portions of this storage can be dumped in integer, real, or character form using the DUMP Control.

Form: \( \text{DUMP} = (\text{integer}_1, \text{integer}_2, \text{integer}_3, \text{integer}_4) \)

where \( \text{integer}_1 \) is the number of an Operation. After each call to this Operation the dump specified by the remaining Elements will occur.

\( \text{integer}_2 \) is the beginning address for the dump,
\( \text{integer}_3 \) is the ending address for the dump, and
\( \text{integer}_4 \) is the format for the dump. The available formats are:

1. 12I10
2. 12F10.2
3. 12A10

Initialized Value: \( \text{DUMP} = (0,1,1,1) \)

Dependencies: KDUMP

Comments: These printouts are useful to trace wild stores being generated while attempting to develop and interface new codes.

A value of zero (0) for \( \text{integer}_1 \) is a no op condition. As many of these DUMP requests can be stacked as wanted. The form would be:

\( \text{DUMP} = (\text{numop},\text{index}_1,\text{index}_2,\text{iformat}), \)
\( (\text{numop},\text{index}_1,\text{index}_2,\text{iformat}), \)
\( (\text{numop},\text{index}_1,\text{index}_2,\text{iformat}), \) etc.

All DUMP output is written to the FORTRAN unit defined by DRIVER Control KDUMP.

If a requested display does not occur check the assignment to unit KDUMP.

2.9 ENDIV

Purpose: Specifies the value of the Independent Variable at which to end processing.

Release 1.0

May 13, 1983
Form: \( \text{ENDIV} = (\text{real}_1, \ldots, \text{real}_N) \)

where the number of Elements in each Group, \( N \), is defined by Control IVDIM,

and where each Element represents one component of the Independent Variable from the most significant component, 'real_1', to the least significant, 'real_N'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then,
\[ \text{BEGIV} = (17., 06., 30.) \]

would result in hour 17, minute 06, and second 29 being the last value of the Independent Variable to be processed.

NOTE! Processing will end one Cycle BEFORE the value given in Control ENDIV.

Initialized Value: \( \text{ENDIV} = (0., 1., 0.) \)

Diagnostics:

****** OPERATION NO. 1 ****** I N D R ****** OPERATION NO. 1 ******
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'ENDIV' MISSING FROM CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: Control ENDIV is missing from the DRIVER PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control ENDIV is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

****** OPERATION NO. 1 ****** I N D R ****** OPERATION NO. 1 ******
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'ENDIV' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***
Cause: The Group length of Control ENDIV is not equal to the number of dimensioned components of the Independent Variable, $IVDIM$.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM$ of the associated code release.

***** OPERATION NO. 1 *** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'BEGIV' VALUE '$' INCONSISTENT WITH CONTROL 'ENDIV' VALUE '$'
**FATAL** **FATAL** **FATAL** **FATAL** **FATAL** **FATAL**

Cause: The value of Control ENDIV is less than or equal to BEGIV, the start time of this Case.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for BEGIV and ENDIV.

***** ERROR IN DRIVER (SUB G2INIT) ***** AT 3.1 ***********************
THE FIRST VALUE OF THE INDEPENDENT VARIABLE ON ANY INPUT UNIT,
* VMIN = '$' EXCEEDS THE REQUESTED STOPPING POINT ENDIV = '$'
* NOTE: IF VMIN IS ONLY 1 CYCLE LARGER THAN ENDIV THIS USUALLY MEANS
* AN INPUT OPERATION WAS EITHER NOT CALLED OR NOT LOADED.
* --- EXECUTION HAS BEEN TERMINATED
***CATASTROPHIC*** ***CATASTROPHIC*** ***CATASTROPHIC*** ***CATASTROPHIC***

Cause: The value of Control ENDIV is less than the first indicated time of the INPUT data.

Program action: The Case is terminated.

Fix: The possibilities include:

- Missing or mismatched Directive Files
- The wrong INPUT dataset is being read
- An inappropriate ENDIV was selected
- The INPUT code has not been loaded
- The INPUT Operation is not being called by the DRIVER routine OPCALL

Release 1.0 May 1, 1983
The first three possibilities involve an error in setting up the Directives. The last two possibilities imply a code or loader problem which might occur in an attempt to create and interface a new INPUT Operation.

If the requested end time for a Case exceeds the last time of the dataset being processed, then the end time of the Case, ENDIV, is truncated to the last observed time, and the following message is printed.

**** (DRIVER) CASE INFORMATION **** (DRIVER) CASE INFORMATION ****
*
* THE ENDING VALUE OF THE I.V., ENDIV, HAS BEEN RESET FROM
* THE INPUT VALUE '$1 TO THE LAST OBSERVED VALUE '$'
*
***** DRIVER ***** INFORMATION ***** DRIVER ***** INFORMATION *****

Dependencies: IVDIM, BEGIV

Controls BEGIV and ENDIV work as ordered pairs describing the beginning and the ending of the processing period of a Case.

For both Controls the number of Elements in a Group is defined by the DRIVER Control IVDIM.

Comments: The number of Elements in a Group for Control ENDIV is initially defined in the Programmer Directive file for this Operation to be three (3).

NOTE: The last Cycle to be processed is one Cycle BEFORE the value given in Control ENDIV.

2.10 ENDSNP

Purpose: Marks the ending of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: ENDSNP = (real1,...,realN),...,(real1,...,realN)

where the number of Elements in each Group, N, is defined by Control IVDIM,

and where each Element represents a component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

May 1, 1983 Release 1.0
EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then,
ENDSNP = (17., 6., 30.) would cause a Snapshot Period to end at hour 17, minute 6, and second 30.

Initialized Value: ENDSNP = (99., 99., 99.)

This value represents the ending of one Snapshot of an Independent Variable with three components.

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'ENDSNP' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The Group length of Control ENDSNP is not equal to the number of dimensioned components of the Independent Variable, $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM of the associated code release.

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for BEGSNP and ENDSNP.

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 70 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAPSHOT INTERVAL '$' ENDSNP = '$' EXCEEDS 'ENDSNP' = '$' OF PRIOR INTERVAL
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value of Control ENDSNP is less than the ENDSNP of the previous Snapshot Period.
Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for the Snapshot Periods.

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
INCONSISTENT SNAPSHOT INTERVALS 'M' THROUGH 'M' HAVE BEEN TURNED OFF.
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: A previous fatal error was detected in either Control BEGSNP or ENDSNP.

Program action: All Snapshot Periods starting with the first one to produce a fatal error condition are deactivated.

Fix: Respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAP INTERVAL 'I' 'BEGSNP' = 'M' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE.
DID YOU WANT TO SKIP A CYCLE?
***WARNING********WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The value of Control BEGSNP is one Cycle larger than the ENDSNP of the previous Snapshot Period.

Program action: A warning message is printed.

Fix: If the intent is to display all Cycles Respecify the BEGSNP of interval I equal to the ENDSNP of interval I-1.

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
CONTROL 'BEGSNP' VALUE 'M' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE 'M'
MESSAGE NO. 61 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
OPERATION 'M' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, (2) the Operation in question is removed from processing.

Fix: Since these intervals were previously examined in DRIVER routine LODCOM, the code of the Operation in question might have modified them, or random data stores might have overwritten them. See DRIVER Controls DEBUG and DUMP for debug aid.
Multiple Snapshot Periods (Groups) can be defined on the total run interval from BEGIV to ENDIV; however, these intervals cannot overlap, but the start of the current interval should repeat the end of the last interval for continuous processing. For example,

\[
\text{BEGIV} = (12.,20.,00.) \quad \$ \quad \text{ENDIV} = (14.,24.,00.)
\]

Over this run interval a legal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>Interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(12.,00.,00.), (12.,30.,00.), (14.,00.,00.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.), (13.,30.,00.), (99.,99.,99.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the intervals from \(13.,30.,00.\) to \(14.,00.,00.\) and \(14.,20.,00.\) to \(14.,24.,00.\) will not generate a display.

Also note that the last Snapshot Period exceeds ENDIV for this Case. The DRIVER will reset it to ENDIV. A similar truncation of a BEGSNP that starts before BEGIV would occur.

Finally note that the start of interval two equals the end of interval one. This insures continuous output. A warning is issued if the start of a subsequent interval is exactly one time step (see DRIVER Control DELIV) larger than the ending of a previous interval.

Over this run interval an illegal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>Interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.), (12.,20.,00.), (14.,00.,00.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.), (13.,30.,00.), (13.,20.,00.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The second interval above is illegal because the BEGSNP of \(12.,20.,00.\) is less than the ENDSNP of interval one.

Also, the third interval above is illegal because the BEGSNP of \(14.,00.,00.\) is larger than the ENDSNP of \(13.,20.,00.\).

**NOTE:** These are fatal conditions and no processing of a display period with a fatal error nor any subsequent intervals is done even if the processor has been instructed to continue in spite of fatal errors (see DRIVER Control IABORT, value 5).

Dependencies: BEGSNP, IVDIM

For Control ENDSNP, the number of Elements in a Group is defined by DRIVER Control IVDIM.

Controls BEGSNP and ENDSNP work as ordered pairs describing the beginning and the ending of a Snapshot Period, that is, the Mth Group of
Control BEGSNP marks the beginning of the Mth Snapshot Period, and the Mth Group of Control ENDSNP marks the ending of that same period. Therefore, these two Controls must have the same number of Groups.

In addition, Snapshot Periods may not overlap. This implies that the next Group of Control BEGSNP must specify an Independent Variable value greater than that of the previous Group of Control ENDSNP.

Comments: The number of Elements in a Group for Control ENDSNP is initially defined in the Programmer Directive file for this Operation to be three (3).

The Snapshot Periods must be an increasing function of the Independent Variable. This implies that the Groups of the Control ENDSNP must be arranged in increasing order.

All Snapshot Periods must lie within the interval of the Independent Variable delimited by DRIVER Controls BEGIV and ENDIV.

The DRIVER Operation actually does not use the value of this Control. It is included in this Operation only to provide convenient access to this value by other Operations. NOTE: The last Cycle to be displayed is one Cycle BEFORE the value given in Control ENDSNP.

2.11 FLUSHP

Purpose: Sets the length of a Flush Period.

A Flush Period is a sub-interval of the Snapshot Period marking the frequency of a specific action. At the end of the specific action, all data within the Flush Period is discarded, or "flushed", and execution on subsequent data within the Snapshot Period is continued.

Form: FLUSHP = integer

where 'integer' is the number of Cycles in the Flush Period.

Initialized Value: FLUSHP = 900

For data with an Independent Variable of time with each Cycle representing one second in time, this number represents fifteen minutes of data.

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'FLUSHP' MISSING FROM CONTROL ARRAY
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***
Cause: Control FLUSHP is missing from the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control FLUSHP is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ **** * $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 6 FROM SUBPROGRAM LODCOM AT SECTION 3.5 EX STAGE IS 2
CONTROL 'FLUSHP' VALUE '$' INVALID; VALID RANGE '$' TO '$'; ASSUMING '$'
*****WARNING********WARNING********WARNING********WARNING********WARNING********WARNING*****

Cause: The value of Control FLUSHP is less than or equal to zero (0).

Program action: (1) A warning message is printed, and (2) the interval for displays, FLUSHP, is reset to the full run interval of the Case.

Fix: If specific intervals for display are wanted, respecify and correlate Controls, BEGSNP, ENDSNP, and FLUSHP.

Dependencies: BEGSNP, ENDSNP, DELIV

Controls BEGSNP and ENDSNP define the Snapshot Periods of which the Flush Period is a sub-interval.

Comments: The Flush Period begins with the first Cycle of data in the Snapshot Period. A counter is employed to count the number of Cycles from the beginning of the Flush Period to the length of the Flush Period defined in Control FLUSHP or until the end of the Snapshot Period. If the number of Cycles remaining in a Snapshot Period is greater than the Flush Period, then a "flush" is performed, the count of Cycles in a Flush Period is reset to zero (0), and a new Flush Period is begun. If the number of Cycles remaining in the Snapshot Period is less than the Flush Period, then a "flush" is performed on an abbreviated Flush Period, marked by the end of the Snapshot Period.

The DRIVER Operation actually does not use the value of this Control. It is included in this Operation only to provide convenient access to this value by other Operations.
2.12 GAPVAL

Purpose: Declares a value which is inserted into the data in place of missing or bad data. All Operations are designed to recognize this special value, and treat it as a gap in the data.

Form: \( \text{GAPVAL} = \text{real} \)

where 'real' is selected by the user to be a value which would not otherwise be present in the real data.

Initialized Value: \( \text{GAPVAL} = 99999 \).

Diagnostics:

****** OPERATION NO. 1 ***** I N D R ****** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM LODCOM AT SECTION 2.3 EX STAGE IS 2
CONTROL 'GAPVAL' MISSING FROM CONTROL ARRAY; ASSUMING '$'

WARNING********WARNING********WARNING********WARNING********WARNING********

Cause: The Control GAPVAL was not found in the DRIVER INDR(ive) PD File.

Program action: (1) A warning message is issued, and (2) a value of 99999. is used.

Fix: Check the release number of the PD File against the release number of the code files for consistency. Also check all Directives for proper ordering and completeness.

Dependencies: GAPFLG

Comments: All Operations are coded to look for this value in the data stream and take appropriate actions when a flag is activated to indicate their presence in the data stream. The flag is GAPFLG. When GAPFLG is a zero (0.), no GAPVAL values are present, when GAPFLG is a one (1.), some of the GAPVAL values occur in the data Cycles currently being processed. The GAPFLG search is activated by the SETRNG Operation.

2.13 IABORT

Purpose: Defines the point at which processing is abandoned due to the detection of fatal errors.

Form: \( \text{IABORT} = \text{integer} \)

where 'integer' is one of the following:
OPERATION CONTROLS

1 If any fatal errors are detected, stop execution anytime after the Programmer Directive file and User Directive files have been input for every Operation.

2 If any fatal errors are detected, stop execution anytime after each Operation has been called to check the consistency of its Controls.

3 If any fatal errors are detected, stop execution anytime after memory space allocation has been completed.

4 If any fatal errors are detected, stop execution anytime after the call to every Operation involving the first Cycles of data.

5 Do not stop the execution due to any detected errors.

Any other value of 'integer' will result in a diagnostic and cause 'integer' to be set to a value of five (5) if IABORT is greater than five (5), or to one (1) if IABORT is less than one (1).

Initialized Value: IABORT = 4

This initial value will try to force execution through the stage where each Operation has been called once with real data.

Diagnostics:

***** OPERATION NO. 1 *****
MESSAGE NO. 6 FROM SUBPROGRAM LODCOM AT SECTION 2.3 EX STAGE IS 2
CONTROL 'IABORT' VALUE '$' INVALID; VALID RANGE '$' TO '$'; ASSUMING '$'
*****WARNING*****WARNING*****WARNING*****WARNING*****WARNING*****

Cause: The value of Control IABORT is less than one (1) or greater than five (5).

Program action: (1) A warning message is printed, and (2) a value less than one (1) is reset to one (1), or a value greater than five (5) is reset to 5.

Fix: Reset IABORT in the legal range of one (1) to five (5).

Comments: Philosophically, it was decided that GENPRO would not cease execution at the occurrence of the first detected fatal error, but would forge ahead as long as possible. This has the positive advantage that the user does not need to run the Case once for each error detected; a single execution can detect and print out numerous fatal errors.

This philosophy has a negative side as well. Since the processor will attempt to forge ahead, it will be wasting computer resources executing incorrectly on the data beyond the point of the last fatal error. Some errors might even be due to previous errors.
Thus the values assignable to IABORT have been chosen to match the stages of initialization that occur within an execution. First stage, all Directives are input and translated; second stage, each Operation is called and allowed to check the correctness of its Controls; third stage, space is allocated for the execution of each Operation; fourth stage, each Operation is called with the first set of Cycles of real data, and finally; fifth stage, each Operation is then called repeatedly with sets of Cycles of real data. A value from one (1) to five (5) selected for Control IABORT allows the user to force execution to continue through the matching stage of initialization listed above.

For example, the initialized value of Control IABORT (4), will print out all Directive errors, all errors for values of Controls, all memory allocation errors, and all errors in data flow for the first Cycle, without trying to process all remaining Cycles.

A table of error summaries is printed at the end of each stage of initialization along with the individual error messages from the Operations which detected them.

2.14 ITYPOP (Derived)

Purpose: Specifies the type of Operation. This Control applies to the INDR Operation, which is an input Operation of the DRIVER used in Job Stepping, if necessary.

Form: ITYPOP = integer

where 'integer' is one of:

-1 In-Place Transformation Operation
0 Snapshot Operation
1 Input Operation
2 Transformation Operation

Initialized Value: ITYPOP = 1

Diagnostics:

***** OPERATION NO.  $ *****   $ $ $ $ $ ***** OPERATION NO.  $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'ITYPOP' MISSING FROM CONTROL ARRAY
***FATAL******FATAL********FATAL********FATAL********FATAL******FATAL***

Cause: The Control ITYPOP was not found in the PD File.

May 1, 1983
Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control ITYPOP is not in the PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

****** OPERATION NO. $ ***** $ $ $ $ $ $****** OPERATION NO. $ *****
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The value of Control ITYPOP is less than a minus one (-1) or greater than two (2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

****** OPERATION NO. $ ***** $ $ $ $ $ $****** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match
the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies: NUMCYC

Comments: The value of this Control is derived from the Programmer Directive file.

2.15 IVDIM

Purpose: Designates the number of components of the Independent Variable.

Form: IVDIM = integer

where 'integer' is the number of components of the Independent Variable, and must be less than, or equal to, the global IFTRAN variable $IVDIM.

Initialized Value: IVDIM = 3

Diagnostics:

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL "IVDIM" MISSING FROM CONTROL ARRAY
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The Control IVDIM is missing from the Control Array.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 6 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL "IVDIM" VALUE '$' INVALID; VALID RANGE '$' TO '$'; ASSUMING '$'
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The value of Control IVDIM is less than zero (0) or greater than the global IFTRAN variable $IVDIM.

Program action: (1) A fatal error flag is set, (2) IVDIM is given the value 1, and (3) execution continues subject to the value assigned to
DRIVER Control IABORT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies:

BEGIV, ENDIV, SCALIV, LABIV, BEGSNP, ENDSNP

Comments: The Independent Variable can be expressed in two forms: a single value called the composite, and a set of values or components. Algorithms to change a composite into its component parts and back again are written into the code, and allow flexibility in defining Controls related to the Independent Variable. The number of components for the Independent Variable is defined by DRIVER Control IVDIM.

**EXAMPLE:** Assume the Independent Variable is time and its components are hour, minute, and second. Further assume that the Cycle interval given in Control DELIV is one second and the data to be processed begins at noon. Then the components of the Independent Variable are simply expressed as hour 12., minute 0., and second 0. The composite would be more complex and have less physical meaning: second 4320., that is 12 hours times 60 minutes per hour, times 60 seconds per minute.

2.16 **KCHECK**

**Purpose:** Declares a logical unit number on which audits, if any, are written. Audits monitor the flow of data by printing critical values and tables that are used to check out the correct running of the processor.

**Form:**

KCHECK = integer

where 'integer' is greater than zero (0) and less than the maximum unit number ascribable (a system dependent number).

**Initialized Value:** KCHECK = 6

**Diagnostics:**

***** OPERATION NO. 1 *****
I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM LODCOM AT SECTION 2.1 EX STAGE IS 2
"CONTROL 'KCHECK' MISSING FROM CONTROL ARRAY; ASSUMING '6'
**WARNING********WARNING********WARNING********WARNING********WARNING********WARNING********WARNING********WARNING********WARNING

Release 1.0
May 1, 1983
cause: The Control KCHECK is missing from the Control Array.

Program action: (1) A warning message is issued, and (2) the standard print unit, 6, is assigned to KCHECK.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KCHECK is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

The same logical unit number may be used for other types of output information from GENPRO, that is, Controls KDUMP, KERR, KFICHE, KFILM, KPRINT, and KPROC may have the same value as Control KCHECK.

2.17 KDUMP

Purpose: Declares a logical unit number on which dumps of large arrays and other debug information, if any, are written.

Form: KDUMP = integer

where 'integer' is greater than zero (0) and less than the maximum unit number ascribable (a system dependent number).

Initialized Value: KDUMP = 4

Diagnostics:

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM LODCOM AT SECTION 2.1 EX STAGE IS 2
CONTROL 'KDUMP' MISSING FROM CONTROL ARRAY; ASSUMING '6'
****WARNING*********WARNING*********WARNING*********WARNING********

Cause: The Control KDUMP is missing from the Control Array.

Program action: (1) A warning message is issued, and (2) the standard print unit, 6, is assigned to KDUMP.

Fix: Check the Programmer Directive File for a valid release number.
PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KDUMP is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

The same logical unit number may be used for other types of output information from GENPRO, that is, Controls KCHECK, KERR, KFICHE, KFILM, KPRINT, and KPROC may have the same value as Control KDUMP.

2.18 KERR

Purpose: Declares a logical unit number on which diagnostics, if any, are written.

Form: KERR = integer

where 'integer' is greater than zero (0) and less than the maximum unit number ascribable (a system dependent number).

Initialized Value: KERR = 6

Diagnostics:

***** OPERATION NO. 1 *****  I N D R  ***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM LODCOM AT SECTION 2.1 EX STAGE IS 2
CONTROL 'KERR' MISSING FROM CONTROL ARRAY; ASSUMING '6'
*****WARNING*****WARNING*****WARNING*****WARNING*****WARNING*****WARNING*****

Cause: The Control KERR is missing from the Control Array.

Program action: (1) A warning message is issued, and (2) the standard print unit, 6, is assigned to KERR.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KERR is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG,
option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

The same logical unit number may be used for other types of output material from GENPRO, that is, Controls KDUMP, KCHECK, KFICHE, KFILM, KPRINT, and KPROC may have the same value as Control KERR.

There are four different types of diagnostics:

Information:
A message is printed notifying the user of some normal action that has been taken.

Warning:
A message is printed notifying the user that an error has been detected, but some action, which corrected the error, has been taken.

Fatal:
A message is printed notifying the user that an error has been detected in an Operation for which no action can be taken to correct the error. The value of Control IABORT will determine whether further execution will be attempted.

Catastrophic:
A message is printed notifying the user that an error has been detected which will adversely affect the entire processor. Regardless of the value of Control IABORT, the entire processor will be terminated immediately following such a diagnostic.

2.19 KFICHE

Purpose: Declares a logical unit number on which microfiche output, if any, is written.

Form: \[ \text{KFICHE} = \text{integer} \]

where 'integer' is greater than zero (0) and less than the maximum unit number ascribable (a system dependent number).

Initialized Value: KFICHE = 4

Diagnostics:
 Cause: The Control KFICHE is missing from the Control Array.

Program action: (1) A warning message is issued, and (2) the FORTRAN unit, 4, is assigned to KFICHE.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KFICHE is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

The same logical unit number may be used for other types of output information from GENPRO, that is, Controls KDUMP, KERR, KCHECK, KFILM, KPRINT, and KPROC may have the same value as Control KFICHE.

Microfiche output consists of plot instructions in metacode. For a complete description of NCAR metacode, see the NCAR Graphics manual.

This unit naturally assumes the existence of fiche capability on the host system.

2.20 KFILM

Purpose: Declares a logical unit on which microfilm output, if any, is written.

Form: KFILM = integer

where 'integer' is greater than zero (0) and less than the maximum unit number ascribable (a system dependent number).

Initialized Value: KFILM = 4

Diagnostics:
***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 ***#
MESSAGE NO. 3 FROM SUBPROGRAM LODCOM AT SECTION 2.1 EX STAGE IS 2
CONTROL 'KFILM' MISSING FROM CONTROL ARRAY; ASSUMING '4'

*****WARNING*******WARNING*******WARNING*******WARNING*******WARNING*******WARNING*****

Cause: The Control KFILM is missing from the Control Array.

Program action: (1) A warning message is issued, and (2) the FORTRAN unit, 4, is assigned to KFILM.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KFILM is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

The same logical unit number may be used for other types of output information from GENPRO, that is, Controls KDUMP, KERR, KFICHE, KCHECK, KPRINT, and KPROC may have the same value as Control KFILM.

Microfilm output consists of plot instructions in metacode. For a complete description of NCAR metacode, see the NCAR Graphics manual.

This unit naturally assumes the existence of microfilm capability on the host system.

2.21 KPRINT

Purpose: Declares a logical unit number on which tabular information from the PRINT Operation, if any, are written.

Form: \[ KPRINT = \text{integer} \]

where 'integer' is greater than zero (0) and less than the maximum unit number ascribable (a system dependent number).

Initialized Value: KPRINT = 4

Diagnostics:
**Cause:** The Control KPRINT is missing from the Control Array.

**Program action:** (1) A warning message is issued, and (2) the standard print unit, 6, is assigned to KPRINT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KPRINT is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

**Comments:** A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

The same logical unit number may be used for other types of output information from GENPRO, that is, Controls KDUMP, KERR, KFICHE, KFILM, KCHECK, and KPROC may have the same value as Control KPRINT.

### 2.22 KPROC

**Purpose:** Declares a logical unit number on which processor messages are written. The processor messages consist of information which is always printed to inform the user of the normal flow of the processor.

**Form:** 

\[
\text{KPROC} = \text{integer}
\]

where 'integer' is greater than zero (0) and less than the maximum unit number ascribable (a system dependent number).

**Initialized Value:** KPROC = 6

**Diagnostics:**

**** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM LODCOM AT SECTION 2.1 EX STAGE IS 2
CONTROL 'KPRINT' MISSING FROM CONTROL ARRAY; ASSUMING '6'
WARNING********WARNING********WARNING********WARNING********WARNING****
OPERATION CONTROLS

Cause: The Control KPROC is missing from the Control Array.

Program action: (1) A warning message is issued, and (2) the standard print unit, 6, is assigned to KPROC.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KPROC is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

The same logical unit number may be used for other types of output information from GENPRO, that is, Controls KDUMP, KERR, KFICHE, KFILM, KPRINT, and KCHECK may have the same value as Control KPROC.

2.23 KUNIT

Purpose: Declares two logical unit numbers required for the execution of the DRIVER Operation.

Form:  

\[
\text{KUNIT} = (\text{integer}_1, \text{integer}_2)
\]

where 'integer1' and 'integer2' are greater than zero (0), less than the maximum unit number ascribable (a system dependent number), and not equal to each other.

Initialized Value: KUNIT = (9,10)

Diagnostics:

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM INITID AT SECTION 1.2 EX STAGE IS 2
CONTROL 'KUNIT' MISSING FROM CONTROL ARRAY; ASSUMING '$'
*****WARNING*****WARNING*****WARNING*****WARNING*****WARNING*****

***** OPERATION NO. $ ***** O U T D ***** OPERATION NO. $ *****
MESSAGE NO. 3 FROM SUBPROGRAM INITOD AT SECTION 1.2 EX STAGE IS 2
CONTROL 'KUNIT' MISSING FROM CONTROL ARRAY; ASSUMING '$'
*****WARNING*****WARNING*****WARNING*****WARNING*****WARNING*****

May 1, 1983

Release 1.0
Cause: The Control KUNIT is missing from the Control Array.

Program action: (1) A warning message is issued, and (2) values of 9 and 10 are assigned to the two FORTRAN units.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control KUNIT is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: A physical device must be attached to the logical unit number given here, either explicitly through system dependent job control language or implicitly (such as some systems assuming logical unit 6 to be the printer device unless otherwise explicitly stated).

If the size of a Case is large in terms of the number of Operations and Dependent Variables to be processed, it might have to be executed as a series of Job Steps. This process is managed by the driver with one required Control, KUNIT, and two optional Controls, LIMREC and NUMBIT, to be set by the user. KUNIT connects two media units for intermediate file storage, generally disk or tape media. NUMBIT is the number of high order bits of the floating point word to be retained during the intermediate Job Stepping I/O. LIMREC is a maximum record size for this intermediate I/O. DATLOG is the largest number of data logical records (Cycles) which would fit within LIMREC, or one (1) if LIMREC is smaller than a data logical record. RECSIZ is the actual data physical record size written, namely DATLOG data logical records of NUMBIT bits per datum.

A Job Step restart using the in-line DRIVER input Operation, INDR(ive), requires the four Controls KUNIT, RECSIZ, DATLOG and NUMBIT. The restart will read data physical records of RECSIZ numeric storage units from the first unit in KUNIT. The number of data logical records in the data physical record is DATLOG. NUMBIT is the number of bits in a packed datum. RECSIZ, DATLOG, and NUMBIT must be set to the same values they had when the output was written.

Since the data is written in floating point form, restarts can only occur on machines of the same floating point type.

In order to restart a Case at a Job Step break the data written to the KUNIT units must be saved. Either the DRIVER input INDR or the general INPUT Operation can then be used to read this data.

The same logical unit number may NOT be used for other types of output information from GENPRO, that is, Controls KCHECK, KDUMP, KERR, KFICHE, KFILM, KPRINT, and KPROC may NOT have the same value as Control KUNIT.
2.24 LABIV

Purpose: Provides names for each of the components of the Independent Variable.

Form: LABIV = (name1,...,nameN)

where the number of Elements in each Group, N, is defined by Control IVDIM,

and where each Element represents the name of one component of the Independent Variable from the most significant component, 'name1', to the least significant, 'nameN'.

Initialized Value: LABIV = ( HRS, MNS, SEC )

Diagnostics:

***** OPERATION NO.  1 *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'LABIV' MISSING FROM CONTROL ARRAY
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The Control LABIV was not found in the DRIVER PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control LABIV is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO.  1 *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'LABIV' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The number of Elements in the Control LABIV is not equal to the IFTRAN variable $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of
Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM of the associated code release.

Dependencies: IVDIM

Comments: The number of Elements in a Group for Control LABIV is initially defined in the Programmer Directive file for this Operation to be three (3).

2.25 LIMREC

Purpose: Determines the maximum number of numeric storage units (often called "computer word" or just "word") in one data physical record involved in the output of data by the OUTD(rive) Operation when Job Stepping is being done.

Form: LIMREC = integer

where 'integer' is non-negative.

Initialized Value: LIMREC = 0

This initial value requests the DRIVER to allocate the buffer size needed for I/O data records if intermediate Job Stepping is required. The derived value is approximately the total available data space divided by twice the total number of Operations in the Operation Sequence.

Diagnostics:

***** OPERATION NO. 1 *****  O U T D  ***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM INITOD AT SECTION 1.2 EX STAGE IS 2
CONTROL 'LIMREC' MISSING FROM CONTROL ARRAY; ASSUMING '$'
*****WARNING*****WARNING*****WARNING*****WARNING*****WARNING*****

Cause: The Control LIMREC was not found in the DRIVER PD File.

Program action: (1) A warning message is issued, and (2) the default derivation is used.

Fix: Check the release number of the PD File against the release number of the code files for consistency. Also check all Directives for proper ordering and completeness.

Dependencies: LIMREC, RECSIZ, KUNIT, DATLOG, RATE

Comments: Only integral numbers of data logical records (Cycles) are written as a data physical record in the Job Step I/O. Thus, the actual data physical record size will be the maximum number of data logical records.
that would pack into a size less than or equal to LIMREC numeric storage units.

If LIMREC is incorrectly set to a size less than one (1) packed data logical record the data physical record size defaults to one (1) packed data logical record.

For a more detailed explanation of Job Stepping refer to the comment section of DRIVER Control KUNIT.

2.26 MAXCHR

Purpose: Specifies the maximum number of characters which can be printed on one line of the output device to which tabular output is to be written.

Form: MAXCHR = integer

where 'integer' is positive and is dependent on the physical limitation of the output device.

Initialized Value: MAXCHR = 128

Comments: This Control is useful for statistical and tabular displays. Standard output to a paper printer might use 128 characters per line while terminal output might be restricted to 78 characters per line.

The DRIVER uses MAXCHR to regulate the columnar width of its RATE table display. (See DRIVER Control PRINT.)

2.27 NAMEOP (Derived)

Purpose: Assigns a unique name to this Operation, which is used in informational and diagnostic prints.

Form: NAMEOP = name

Initialized Value: NAMEOP = INDR

Comments: The value of this Control is derived from the Programmer Directive file.

2.28 NAMLIB

Purpose: Provides file name(s) which comprise all required code for this Operation if stored on some permanent storage medium (such as PLIB on the CDC 7600).
Form: NAMLIB = string1, ..., stringN

where each 'string' is a file name.

Initialized Value: NAMLIB = 'NONE'

Comments: This Control is currently nonfunctional.

2.29 NAMPD

Purpose: Provides the name of the Programmer Directive file for this Operation.

Form: NAMPD = string

Initialized Value: NAMPD = 'NONE'

Comments: This Control is currently nonfunctional.

2.30 NUMBIT

Purpose: The number of binary bits of one data word, or datum, in the data physical record output from a Job Step. It is some number of the high order bits of a full floating point numeric storage unit (computer word).

Form: NUMBIT = integer

where 'integer' will either be:

0 requests a full numeric storage unit, BITNSU, or

N a positive value typically 30 bits or more, but not exceeding BITNSU.

Initialized Value: NUMBIT = 0

This initial value presumes a full numeric storage unit of the host machine on which GENPRO is to be executed, BITNSU.

Diagnostics:
***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM INITID AT SECTION 1.2 EX STAGE IS 2
CONTROL 'NUMBIT' MISSING FROM CONTROL ARRAY; ASSUMING '¥'
WARNING********WARNING********WARNING********WARNING********WARNING****

***** OPERATION NO. $ *****
MESSAGE NO. 3 FROM SUBPROGRAM INITOD AT SECTION 1.2 EX STAGE IS 2
CONTROL 'NUMBIT' MISSING FROM CONTROL ARRAY; ASSUMING '¥'
WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The Control NUMBIT was not found in the DRIVER PD File.

Program action: A Warning message is issued.

Fix: Check the release number of the PD File against the release number of the code files for consistency. Also check all Directives for proper ordering and completeness.

***** OPERATION NO. 1 *****
MESSAGE NO. 8 FROM SUBPROGRAM INITID AT SECTION 1.2 EX STAGE IS 2
CONTROL 'NUMBIT' VALUE '¥' UNREASONABLE; ASSUMING '¥'
WARNING********WARNING********WARNING********WARNING********WARNING****

***** OPERATION NO. $ *****
MESSAGE NO. 8 FROM SUBPROGRAM INITOD AT SECTION 1.2 EX STAGE IS 2
CONTROL 'NUMBIT' VALUE '¥' UNREASONABLE; ASSUMING '¥'
WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The value is less than zero (0) or greater than BITNSU.

Program action: (1) A warning message is issued, and (2) an assumed value of BITNSU is used.

Fix: If the assumed value is incorrect respecify the correct value.

Dependencies: LIMREC, RECSIZ, KUNIT, DATLOG, RATE

Comments: If DRIVER Job Stepping is needed, data values are reduced to NUMBIT high order floating point bits. This is done to reduce the amounts of intermediate I/O when the machine word size exceeds the required accuracy of the data.

Since the data is written in floating point form, restarts can only occur on machines of the same floating point type.

For a more detailed explanation of Job Stepping refer to the comment section of Control KUNIT.
NUMCYC

Purpose: Defines the amount of space, in units of the number of data Cycles, which is to be allocated for this Operation's use. This Control applies to the INDR(ive) Operation, which is an Input Operation controlled by the DRIVER. It is used in Job Stepping, if necessary.

Form: NUMCYC = integer

where 'integer' is one of:

>0 That number of Cycles are allocated, suspending the automatic Cycle allocation algorithm of the DRIVER for this Operation.

0 Execution of this Operation is suspended for this run.

-1 The DRIVER optimizes the number of Cycles to be allocated based on the best flow for all Operations involved in this run.

-2 Available only for select Snapshot Operations, this value instructs the DRIVER that no space is to be allocated for data Cycles for this Operation.

Initialized Value: NUMCYC = 0

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'NUMCYC' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control NUMCYC is less than a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Release 1.0 May 1, 1983
Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** ERROR IN DRIVER (SUB G2INIT) ***** AT 5.3 ***********************
* INSUFFICIENT CYCLES SPECIFIED FOR OP $ NAME = $
* OVERLAP NEEDED = ' $' BUT NUMCYC IS ONLY ' $'
* THE OVERLAP REQUESTED BY THE ACCESSING OPERATIONS IS
  * NO. NAME OVRLAP
  * OP ' $' ' $' ' $'
  * OP ' $' ' $' ' $'
**FATAL********FATAL********FATAL********FATAL********FATAL******** FATAL***

Cause: The number of Cycles of data storage requested through Directive NUMCYC is less than is needed to provide the requested Cycles of overlap.

Program action: (1) A fatal error flag is set, (2) the positive value requested in the NUMCYC Directive is reset to -1, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the validity of the requested overlap in the cited Operations. If the problem does not surface take your printouts to a GENPRO consultant.

***** ERROR IN DRIVER (SUB G2INIT) ***** AT 6.0 ***********************
* OPERATION NUMBER ' $' NAME = ' $' TYPE = ' $' NUMCYC = ' $' LENCYC = ' $'
* LENCYC MUST BE POSITIVE IF NUMCYC .NE. -2 OR 0
* THE VARIABLE CONTROL RATE IS EITHER MISSING OR EMPTY
* CHECK THE OPERATION CONTROLS - NUMCYC HAS BEEN RESET TO -2
**FATAL********FATAL********FATAL********FATAL********FATAL******** FATAL***

Cause: None of the Dependent Variables has a requested output rate (value of Control RATE) greater than zero (0), but the value of Control NUMCYC (-1 or +) requests a data buffer.

Program action: (1) A fatal error flag is set, (2) the value of Control NUMCYC is reset to -2 (no buffer), and (3) execution continues subject to the value assigned to DRIVER Control IABORT.
Fix: Check the validity of the RATE and NUMCYC Directives. Also verify that the Variable Area Vector (VECVAR) has been defined. If the problem is still unresolved check the release numbers and ordering of the Programmer Directive files.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: ITYPOP, RATE

Comments: The value of this Control is used to define the number of Cycles needed for the DRIVER's Input Operation, INDR(ive). The INDR(ive) Operation is a much simplified version of the general INPUT Operation, and is used solely to input data which has been output due to a Job Stepping procedure initiated by the DRIVER Operation.

The value for Control NUMCYC is initially zero (0), meaning that the INDR(ive) Operation is turned off, since in the normal case, Job Stepping is not done. If it becomes necessary to Job Step, then the value of Control NUMCYC is changed internally to a positive value, reflecting the number of Cycles that are output in each physical record by the OUTD(ive) Operation.

There is only one situation in which the user would specify a value for Control NUMCYC other than the initialized value of zero (0). If Job Stepping is performed, the OUTD(ive) Operation has written out data, and the Case terminates before the next Job Step, then another Case may be run which picks up where the last Job Step left off. In this case, the value of Control NUMCYC must be set to a positive value which is the same as the number of Cycles in one physical record as output by the OUTD(ive) Operation.

2.32 NUMOPT

Purpose: Specifies the minimum number of Cycles which are to be processed in each call to every Operation.

Form: NUMOPT = integer

where 'integer' is positive.

Initialized Value: NUMOPT = 1

Diagnostics:

** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING** WARNING**

Release 1.0

May 1, 1983
**OPERATION CONTROLS**

**Cause:** The Control NUMOPT is missing from the DRIVER PD File.

**Program action:** (1) A warning message is issued, (2) the value one (1) is assigned to Control NUMOPT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** OPERATION NO. 1 *****

IN D R

MESSAGE NO. 5 FROM SUBPROGRAM LODCOM AT SECTION 2.3 EX STAGE IS 2
CONTROL 'NUMOPT' VALUE '$' INVALID; ASSUMING '1'

***** WARNING********WARNING********WARNING********WARNING********WARNING*******

**Cause:** The value of Control NUMOPT is less than one (1).

**Program action:** (1) A warning message is issued, (2) the value one (1) is assigned to Control NUMOPT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** ERROR IN DRIVER (SUB INCORE) ***** AT 2.1

* JOB STEP '$' NEEDS '$' CELLS, AVAILABLE = '$' SHORTAGE = '$'
* THE OP NUMBERS ARE '$' '$' '$' ... 
* OP NAME TYPE LENCA LENSAV LENBUF NUMCYC NCYEND MINCY LENCYC LENDAT

* $ $ $ $ $ $ $ $ $ $ $ $ $ 
* $ $ $ $ $ $ $ $ $ $ $ $ $ 
* $ $ $ $ $ $ $ $ $ $ $ $ $ 

***** ERROR IN DRIVER (SUB INCORE) ***** AT 2.1

CHECK THE NUMBER(NUMCYC) AND LENGTH(LENCYC) OF CYCLES
CHECK LENCA LENSAV LENBUF AND LENDAT
IF ALL ARE OKAY, POSSIBLE AVENUES TO PURSUE INCLUDE:
1 - REDUCE NUMOPT, THE NUMBER OF CYCLES FOR FLOW EFFICIENCY - MINIMUM = 1
2 - REDUCE THE AMOUNT OF DATA - FEWER VARIABLES, REDUCED RATES, ETC.
3 - REDUCE THE CYCLE INTERVAL, DELIV - REMEMBER THAT EACH VARIABLE MUST OCCUR AT LEAST ONCE PER CYCLE, OR
4 - REDIMENSION SPACE TO BE AT LEAST '$' CELLS LARGER

***** CATASTROPHIC********CATASTROPHIC********CATASTROPHIC********CATASTROPHIC*****

**Cause:** The value of Control NUMOPT is too large for the allocated data space.

**Program action:** The Case terminates with a message which suggests a corrective line of action.

May 1, 1983

Release 1.0
Fix: Reset NUMOPT to a size which is consistent with the SPACE array dimension LENSP.

Dependencies: NUMCYC

Comments: All available space is allocated among the Operations involved in a given Case. The allocation process begins by attempting to allocate NUMOPT Cycles for the first Operation, then determines how much, if any memory space remains. If the first Operation cannot be allocated NUMOPT Cycles from the available memory space, then a catastrophic error is printed and the Case is aborted. The next Operation is then allocated NUMOPT Cycles if there is enough memory space, and the process is repeated for every Operation in the Case.

If the value of Control NUMCYC for an Operation is a positive value, yet is less than the value of Control NUMOPT, then the DRIVER Operation will attempt to allocate NUMOPT Cycles. If, on the other hand, the value of Control NUMCYC is greater than the value of Control NUMOPT, then the DRIVER Operation will attempt to allocate NUMCYC Cycles.

2.33 OVRFLO

Purpose: Identifies a special value inserted between logical divisions of memory reserved for data, used to mark the boundaries of these divisions.

Form: OVRFLO = real

where 'real' may be any number that is not within the range of values for any data which is recorded by the processor.

Initialized Value: OVRFLO = -99999.

Diagnostics:

***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM LODCOM AT SECTION 2.3 EX STAGE IS 2
CONTROL 'OVRFLO' MISSING FROM CONTROL ARRAY; ASSUMING '-99999.'
WARNING

Cause: The Control OVRFLO is missing from the DRIVER PD File.

Program action: (1) A warning message is issued, (2) the value -99999 is assigned to Control OVRFLO.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.
Comments: At the completion of each call of an Operation, the boundaries of all of its logical divisions are compared to the value of Control OVRFLO. Since the boundary was originally set to the value of Control OVRFLO, any discrepancy would indicate that some data was written incorrectly past its defined boundary. A diagnostic is generated by DRIVER routine MANAGE (section 9) notifying the user of a GENPRO memory bound error.

2.34 OVRLAP

Purpose: Defines the number of Cycles on each side of the current Cycle(s) required to carry out the requested computations.

Form: OVRLAP = integer

where 'integer' is a small, non-negative number representing the number of Cycles of data on each side of the current computational Cycles.

Initialized Value: OVRLAP = 0

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 5 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'OVRLAP' VALUE '$' INVALID; ASSUMING '0'
*****WARNING*******WARNING*******WARNING*******WARNING*******WARNING*******

Cause: The value of Control OVRLAP is less than zero (0).

Program action: (1) A warning diagnostic is generated, and (2) a value of 0 is assigned to Control OVRLAP.

Fix: Assign a non-negative value to OVRLAP or allow the Operation to select its own value.

***** OPERATION NO. $ ***** $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 11 FROM SUBPROGRAM LODCOM AT 3.4 EX STAGE IS 2
CONTROL 'OVRLAP' VALUE '$' IS LARGE; IS THIS WHAT YOU INTENDED?
*****WARNING*******WARNING*******WARNING*******WARNING*******WARNING*******

Cause: The value of Control OVRLAP is greater than three (3).

Program action: A warning diagnostic is generated.

Fix: The action requested for this Operation would involve more than 3 Cycles of past and future data. If this is a reasonable request proceed and ignore the above warning.

May 1, 1983
***** ERROR IN DRIVER (SUB G2INIT) ***** AT 5.3 *******************
* INSUFFICIENT CYCLES SPECIFIED FOR OP $ NAME = $
* OVERLAP NEEDED = '$' BUT NUMCYC IS ONLY '$'
* THE OVERLAP REQUESTED BY THE ACCESSING OPERATIONS IS
  * NO. NAME OVRLAP
  * OP '$' '$' '$'
  * OP '$' '$' '$'
**FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The number of Cycles of data storage requested through Directive NUMCYC is less than is needed to provide the requested Cycles of overlap.

Program action: (1) A fatal error flag is set, (2) the positive value requested in the NUMCYC Directive is reset to -1, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the validity of the requested overlap in the cited Operations. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: Normally this Control is internally computed in the Operations which have a need for past or future data in order to accomplish the requested actions. A user would override that computation (setting Control OVRLAP to a positive value) only if there is a specific reason to do so.

2.35 PRDATE

Purpose: Associates a project date with the data being processed, most typically the date that this data was collected.

Form: PRDATE = ( string1, string2, string3 ) where each string may be up to four (4) characters in length,
where 'string1' is the day of the month,
where 'string2' is the month of the year,
where 'string3' is the year.

Initialized Value: PRDATE = ( ' ', ' ', ' ', ' ' )

Comments: The value of Control PRDATE is used only for informational purposes, usually it precedes various display outputs.
2.36 PRINT

Purpose: Specifies informational output options to be generated by the DRIVER Operation.

Form: PRINT = (name1), (name2), ..., (STOP)

where 'name1', 'name2', ..., specify informational displays which are one of:

CASE  Project or Case related information, including the summary of Job Steps in each Case, a summary of Cases in a Job with the respective run intervals, and late start and early end values of the Independent Variable of the Input data files.

RATE  A table of the rates of all Dependent Variables, which lists the rate of each Dependent Variable for each Operation in the Operation Sequence. Much valuable information can be garnered from this table including:

- Misspelled names of Dependent Variables
- where a Dependent Variable is input or derived (its rate would change from zero to a positive integer)
- Where a Dependent Variable rate is modified (its rate would change from one positive integer to another)
- Where a Dependent Variable is dropped (its rate would change from a positive integer back to zero (0))

SNAP  A table of the Snapshot Intervals specified for all Operations, used primarily to show that the Snapshot Intervals are complete and consistent.

TIME  Processor timing of translation, data management, and Operations, printed at the completion of each Job Step of each Case.

SIZE  A table of locations and lengths of all data partitions for all Operations of the Operation Sequence. The partitions shown are for Controls, save space, buffers, data, and scratch (work) space. Additionally, the length of one Cycle, the number of Cycles of overlap, and the number of Cycles in the Data Window are listed for each Operation.

Up to six Groups separated by commas may be assigned, where each Group consists of one name Element from the list above. The last Group must always have the value STOP to indicate the end of the list.

The order of the Groups is irrelevant: all information requested will always be generated in the same sequence. If the value of a Group is misspelled, it is deleted from the list and thereby ignored.
Initialized Value: PRINT = (CASE), (RATE), (SIZE), (SNAP), (TIME), (STOP)

Dependencies: KPROC, MAXCHR

All PRINT displays are written to the FORTRAN unit defined by DRIVER Control KPROC.

The number of characters to be displayed on one line of the RATE table are set by DRIVER Control MAXCHR.

Comments: These informational displays are helpful in bringing GENPRO up on a new application, especially for the GENPRO consultants. Once a run has been verified as acceptable for production processing these displays can be selectively turned off by moving the STOP option to follow the wanted list of options.

If a requested display does not occur check the assignment to unit KPROC and the spelling of the option name in PRINT.

2.37 PROJECT

Purpose: Provides a textual description or title of the project, which is used to label the various output generated by this Operation.

Form: PROJECT = string

where 'string' may be up to forty (40) characters in length.

Initialized Value: PROJECT = '

that is, forty (40) blank characters.

Comments: The DRIVER Operation actually does not use the value of this Control. It is included in this Operation only to provide convenient access to this value by other Operations.

2.38 PRTIME

Purpose: Associates a project time with the data being processed, most typically the time that this data was collected.

Form: PRTIME = ( string1, string2, string3 ) where each string may be up to four (4) characters in length,

where 'string1' is the hour of the day,

where 'string2' is the minute of the hour,
where 'string3' is second of the minute.

Initialized Value: PRTIME = ( ' ', ' , ' )

Comments: The value of Control PRTIME is used for informational purposes, usually it precedes various display outputs.

2.39 RECSIZ

Purpose: Determines the number of numeric storage units (often called "computer word" or just "word") in one data physical record involved in the output of data by the OUTD(rive) Operation when Job Stepping is being done.

In the special situation in which a Job Step restart is being performed, the INDR(ive) Operation of the DRIVER is turned on by setting NUMCYC = DATLOG of the corresponding OUTD(rive) Operation which wrote the data file. Then RECSIZ, NUMBIT, and DATLOG must be set to the values used by OUTD(rive), and KUNIT must connect the file unit.

Form: RECSIZ = integer

where 'integer' is non-negative.

Initialized Value: RECSIZ = 0

This initial value assumes that no Job Step restart is to occur. As a Job Step occurs, RECSIZ will contain the actual data physical record size which is being written (Subroutine OUTD(rive)), or read (Subroutine INDR(ive)).

A positive value, coupled with a positive NUMCYC, would imply a Job Step restart.

Diagnostics:

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 3 FROM SUBPROGRAM INITID AT SECTION 1.2 EX STAGE IS 2
CONTROL 'RECSIZ' MISSING FROM CONTROL ARRAY; ASSUMING '$'
****WARNING********WARNING********WARNING********WARNING********WARNING****

***** OPERATION NO. $ ***** O U T D ***** OPERATION NO. $ *****
MESSAGE NO. 3 FROM SUBPROGRAM INITOD AT SECTION 1.2 EX STAGE IS 2
CONTROL 'RECSIZ' MISSING FROM CONTROL ARRAY; ASSUMING '$'
****WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The Control RECSIZ was not found in the DRIVER PD File.

Program action: A warning message is issued.
Fix: Check the release number of the PD File against the release number of the code files for consistency. Also check all Directives for proper ordering and completeness.

**** OPERATION NO. 1 ****

MESSAGE NO. 8 FROM SUBPROGRAM INITID AT SECTION 1.2 EX STAGE IS 2
CONTROL 'RECSIZ' VALUE '$' UNREASONABLE; ASSUMING '$'

****WARNING******WARNING******WARNING******WARNING******WARNING****

**** OPERATION NO. $ ****

MESSAGE NO. 8 FROM SUBPROGRAM INITOD AT SECTION 1.2 EX STAGE IS 2
CONTROL 'RECSIZ' VALUE '$' UNREASONABLE; ASSUMING '$'

****WARNING******WARNING******WARNING******WARNING******WARNING****

Cause: The value is less than [(the logical record length)+3] or greater than LENSPE.

Program action: (1) A warning message is issued and (2) Control RECSIZ is assigned the value [DATLOG *(the logical record length) + 3].

Fix: If the assumed value is incorrect respecify the correct value.

Dependencies: LIMREC, NUMBIT, KUNIT, DATLOG, NUMCYC

For a more detailed explanation of Job Stepping refer to the comment section of Control KUNIT.

2.40 SCALIV

Purpose: Provides factors for each of the components of the Independent Variable used to scale the components into one composite value of the Independent Variable.

Form: SCALIV = (real1, ..., realN)

where the number of Elements in each Group, N, is defined by Control IVDIM, and where each Element represents the scale factor of one component of the Independent Variable from the most significant scale factor, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is Time recorded in components of hour, minute, and second, then,

SCALIV = (24., 1., .01667)

would result in a composite Independent Variable in units of minutes.

Initialized Value: SCALIV = (3600., 60., 1.)

This initial set of values defines an Independent Variable composite of
seconds when the units of the three components are hours, minutes, and
seconds.

Diagnostics:

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'SCALIV' MISSING FROM CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Control SCALIV was not found in the DRIVER PD File.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number.
PD release numbers must match the code release numbers. Then verify
that the ordering of the PDs and UDs is correct in the set of Directive
files. If Control SCALIV is not in the DRIVER PD, that PD is a wrong
release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG,
option CONTROLS. Also consider the use of DRIVER Control DUMP. If the
problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. 1 ***** I N D R ***** OPERATION NO. 1 *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'SCALIV' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The number of Elements in the Control SCALIV is not equal to the
IFTRAN variable $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number.
PD release numbers must match the code release numbers. The number of
Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM
of the associated code release.

Dependencies: IVDIM, BEGIV, ENDIV, BEGSNP, ENDSNP

Comments: The number of Elements in a Group for Control SCALIV is initially
defined in the Programmer Directive file for this Operation to be three
(3).

Each of the following Controls is defined in component form and is
scaled into composite form using the values given in Control SCALIV:
BEGIV, ENDIV, BEGSNP, ENDSNP.
3. VARIABLE CONTROLS

3.1 INDGET (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the input to this Operation for this call, is obtained.

Form: INDGET = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INDGET = 0

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDGET' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION ' $ ' TERMINATED; NUMYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Control INDGET was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDGET is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ ***** $ $ $ $ $ $ $ $ $ $ OPERATION NO. $ *****
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The data indices associated with Control INDGET are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues
subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDGET is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indices throughout the entire program flow. If an INDGET index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: INRATE

Comments: The value of Control INDGET is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control INRATE, and by the number of Cycles passed to this Operation in this call, are stored contiguously.

3.2 INDPUT (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the result from this Operation for this call, is stored.

Form: INDPUT = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INDPUT = 0

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDPUT' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The Control INDPUT was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDPUT is not in the Operation PD, that PD is a
wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

```
***** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*****
```

Cause: The data indices associated with Control INDPUT are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDPUT is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indices throughout the entire program flow. If an INDPUT index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: RATE

Comments: The value of Control INDPUT is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control RATE, and by the number of Cycles available to this Operation in this call, are stored contiguously.

3.3 INRATE (Derived)

Purpose: Specifies the number of data values per Cycle to be input to this Operation for this Dependent Variable.

Form: \[ \text{INRATE} = \text{integer} \]

where 'integer' is greater than or equal to zero (0).

Initialized Value: INRATE = 0
**Diagnostics:**

**** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INRATE' MISSING FROM CONTROL ARRAY

MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

**Cause:** The Control INRATE was not found in the Operation PD File.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INRATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

**** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

**** OPERATION NO. $ ***** $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOBMV AT SECTION 1.2 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

**Cause:** The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

**Program action:** (1) A fatal flag is set, (2) the data move is not performed.

**Fix:** Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.
Dependencies: INDGET, RATE

Comments: The value assigned to Control INRATE is derived in the DRIVER from the value of Control RATE of the last Transformation Operation.

When the value of Control INRATE is zero (0), no data is to be passed to this Operation, and thus no function is performed for this Dependent Variable.

The number of data values per Cycle given in Control INRATE are stored contiguously, with the address of the first data value defined by Control INDGET.

3.4 NAMVAR (Derived)

Purpose: Assigns a name to be associated with a Dependent Variable.

Form: \( NAMVAR = \text{name} \)

Initialized Value: \( NAMVAR = NAMVAR \)

Comments: The value of Control NAMVAR is indirectly derived in the Translator by the Area Vector Directives which manipulate items of the Variable Area Vector.

3.5 RATE

Purpose: Specifies the number of data values per Cycle to be stored by this Operation for this Dependent Variable.

Form: \( RATE = \text{integer} \)

where 'integer' is greater than or equal to zero (0).

Initialized Value: \( RATE = 1 \)

Diagnostics:

***** OPERATION NO. $ ***** $ $ $ $ $ $ ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'RATE' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
FATALFATALFATALFATALFATALFATALFATALFATALFATALFATAL

If NUMCYC = -1, or positive, the RATE Control must appear in the Programmer Directive file.

Release 1.0 May 1, 1983
**OPERATION CONTROLS**

**Cause:** The Control RATE was not found in the Operation PD File.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control RATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

**Causes**

The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

**Program action:** (1) A fatal flag is set, (2) the data move is not performed.

**Fix:** Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.

**ERROR IN DRIVER (SUB G2INIT) AT 6.0 **

* OPERATION NUMBER '$' NAME = '$' TYPE = '$' NUMCYC = '$' LENCYC = '$'
* LENCYC MUST BE POSITIVE IF NUMCYC .NE. -2 OR 0
* THE VARIABLE CONTROL RATE IS EITHER MISSING OR EMPTY
* CHECK THE OPERATION CONTROLS - NUMCYC HAS BEEN RESET TO -2
**FATAL**

**Cause:** None of the Dependent Variables has a requested output rate.
(value of Control RATE) greater than zero (0), but the value of Control NUMCYC (-1 or +) requests a data buffer.

Program action: (1) A fatal error flag is set, (2) the value of Control NUMCYC is reset to -2 (no buffer), and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the validity of the RATE and NUMCYC Directives. Also verify that the Variable Area Vector (VECVAR) has been defined. If the problem is still unresolved check the release numbers and ordering of the Programmer Directive files.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: INRATE

Comments: The number of data values per Cycle given in Control RATE are stored contiguously, with the address of the first data value defined by Control INDPUT.
# DESP CONTROLS

## CONTENTS

1. INTRODUCTION ........................................................................ 3-DESP.1

2. GENERAL CONTROLS .................................................................. 3-DESP.2
   2.1 BEGSNP ........................................................................... 3-DESP.2
   2.2 ENDSNP ........................................................................... 3-DESP.6
   2.3 FLUSHP ............................................................................ 3-DESP.10
   2.4 ITYPOP (Derived) ................................................................. 3-DESP.12
   2.5 KOUT ............................................................................ 3-DESP.13
   2.6 LOUT ............................................................................ 3-DESP.14
   2.7 MAXCHR ........................................................................ 3-DESP.16
   2.8 NAMEOP (Derived) ................................................................. 3-DESP.17
   2.9 NAMLIB .......................................................................... 3-DESP.17
   2.10 NAMPD .......................................................................... 3-DESP.17
   2.11 NUMCYC ......................................................................... 3-DESP.17
   2.12 OVRLAP ....................................................................... 3-DESP.19
   2.13 PROJECT ....................................................................... 3-DESP.20

3. VARIABLE CONTROLS .................................................................. 3-DESP.21
   3.1 DELTOL .......................................................................... 3-DESP.21
   3.2 INDGET (Derived) ................................................................. 3-DESP.22
   3.3 INDPUT (Derived) ................................................................. 3-DESP.23
   3.4 INRATE (Derived) ................................................................. 3-DESP.24
   3.5 LSHFTS (Derived) ................................................................. 3-DESP.26
   3.6 NAMVAR (Derived) ................................................................. 3-DESP.26
   3.7 NUMGAP (Derived) ................................................................. 3-DESP.27
   3.8 NUMPTS (Derived) ................................................................. 3-DESP.27
   3.9 NUMSPK (Derived) ................................................................. 3-DESP.27
   3.10 NUMUNC (Derived) ............................................................... 3-DESP.28
   3.11 NWIDTH ........................................................................ 3-DESP.28
   3.12 RATE (Derived) ................................................................. 3-DESP.29
   3.13 SOURCE (Linked) ................................................................. 3-DESP.31
   3.14 TOTPTS (Derived) ................................................................. 3-DESP.32
   3.15 TRPTYP ........................................................................ 3-DESP.33
   3.16 UNITS ........................................................................ 3-DESP.34
1. INTRODUCTION

The purpose of the DESP Operation is to remove Spikes from the data values of a Dependent Variable. Spikes are defined as large perturbations of a relatively small number of data values. In despiking a Dependent Variable two basic functions are performed: detection and correction.

Detection is done by differencing adjacent data values and comparing this delta to a maximum 'reasonable' range for the Dependent Variable. If this 'reasonable' range is exceeded, then either a Spike or a Level Shift exists at this point. Level Shift is defined to be a shifting of all succeeding data values by some large, 'unreasonable' amount. Thus a second step in detection is performed in which the next several data values are examined to distinguish between a Spike, involving only a few 'unreasonable' data values, and a Level Shift, involving many such consecutive values.

The user has complete control in both aspects of the detection phase, being able to set the maximum 'reasonable' range for every Dependent Variable, and the number of data values to be compared to distinguish between Spikes and Level Shifts.

Once a Spike has been detected and isolated from Level Shifts, correction of the unreasonable data values is performed. This involves interpolation between the last good data value before the Spike and the first good data value after the Spike. The user has control over the type of interpolation performed: available types are linear, quadratic, and cubic interpolation.

The DESP Operation generates various statistics while it is operating on the data values of the Dependent Variables. These statistics count the number of Spikes, Level Shifts, etc. However, these statistics are generated only during user-specified intervals of the Independent Variable. Although the DESP Operation is a Transformation Operation, it employs Snapshot Periods, which are normally employed in Snapshot Operations, as the user-specified intervals over which despike statistics are generated. In other words, the DESP Operation works full-time (as all Transformation Operations do) on detecting and correcting Spikes in the data values of the requested Dependent Variables, but the generation of the despike statistics works only over the user-requested Snapshot periods (as all Snapshot Operations do).
2. GENERAL CONTROLS

2.1 BEGSNP

Purpose: Marks the beginning of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: \( \text{BEGSNP} = (\text{real}_1, \ldots, \text{real}_N), \ldots, (\text{real}_1, \ldots, \text{real}_N) \)

where the number of Elements in each Group, N, is defined by Control \text{IVDIM},

and where each Element represents a component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then,

\( \text{BEGSNP} = (3., 51., 0.) \)

would cause a Snapshot Period to begin at hour 3, minute 51, and second 0.

Initialized Value: \( \text{BEGSNP} = (0.0, 0.0, 0.0) \)

This value represents the beginning of one Snapshot of an Independent Variable with three components. For a positive valued Independent Variable it also represents a Snapshot which will begin at the first Cycle of the data for any Case.

Diagnostics:

**** OPERATION NO. $ **** D E S P ***** OPERATION NO. $ ****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'BEGSNP' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The Group length of Control BEGSNP is not equal to the number of dimensioned components of the Independent Variable, $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM.

May 1, 1983

Release 1.0
**OPERATION CONTROLS**

of the associated code release.

*-*-*-* OPERA-TION NO. $ **** D E S P  *-*-*-* OPERA-TION NO. $ ****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL*****

**Cause:** The value of Control BEGSNP is greater than or equal to ENDSNP,
the end time of this Snapshot Period.

**Program action:** (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP.

*-*-*-* OPERA-TION NO. $ **** D E S P  *-*-*-* OPERA-TION NO. $ ****
MESSAGE NO. 10 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH 'ENDSNP' VALUE '$'; ASSUMING '$'
******WARNING********WARNING********WARNING********WARNING********WARNING*****

**Cause:** The value of Control BEGSNP is less than the ENDSNP of the pre-
vious Snapshot Period.

**Program action:** A warning message is printed.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP intervals.

*-*-*-* OPERA-TION NO. $ **** D E S P  *-*-*-* OPERA-TION NO. $ ****
MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
INCONSISTENT SNAPSHOT INTERVALS '$' THROUGH '$' HAVE BEEN TURNED OFF.
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL*****

**Cause:** A previous fatal error was detected in either Control BEGSNP or
ENDSNP.

**Program action:** All Snapshot Periods starting with the first one to
produce a fatal error condition are deactivated.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP intervals.

*-*-*-* OPERA-TION NO. $ **** D E S P  *-*-*-* OPERA-TION NO. $ ****
MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAP INTERVAL 'I' 'BEGSNP' = '$' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE.
**DO YOU WANT TO SKIP A CYCLE?**
******WARNING********WARNING********WARNING********WARNING********WARNING*****

**Cause:** The value of Control BEGSNP is one Cycle larger than the ENDSNP
of the previous Snapshot Period.

Release 1.0

May 1, 1983
Program action: A warning message is printed.

Fix: If the intent is to display all Cycles respecify the BEGSNP of interval I equal to the ENDSNP of interval I-1.

***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'

***** OPERATION NO. $ *****
MESSAGE NO. 61 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, (2) the Operation in question is removed from processing.

Fix: Since these intervals were previously examined in DRIVER routine LODCOM, the code of the Operation in question might have modified them, or random data stores might have overwritten them. See DRIVER Controls DEBUG and DUMP for debug aid.

Multiple Snapshot Periods (Groups) can be defined on the total run interval from BEGIV to ENDIV; however, these intervals cannot overlap although the start of the current interval should repeat the end of the last interval for continuous processing. For example,
BEGIV = (12.,20.,00.) $ ENDIV = (14.,24.,00.)

Over this run interval a legal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,30.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(14.,20.,00.)</td>
</tr>
</tbody>
</table>

Note that the intervals from 13.,30.,00. to 14.,00.,00. and 14.,20.,00. to 14.,24.,00. will not generate a display.

Also note that the first Snapshot Period is less than BEGIV for this Case. The DRIVER will reset it to BEGIV. If an ENDSNP value extends beyond ENDIV, a similar truncation would occur.

Finally note that the start of interval two equals the end of interval one. This insures continuous output. A warning is issued if the start of a subsequent interval is exactly one time step (see DRIVER Control DELIV) larger than the ending of a previous interval.

Over this run interval an illegal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,20.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(14.,20.,00.)</td>
</tr>
</tbody>
</table>

The second interval above is illegal because the BEGSNP of (12.,20.,00.) is less than the ENDSNP of interval one.

Also, the third interval above is illegal because the BEGSNP of (14.,00.,00.) is larger than the ENDSNP of (13.,20.,00.).

NOTE: These are fatal conditions and no processing of a display period with a fatal error nor any subsequent intervals is done even if the processor has been instructed to continue in spite of fatal errors (see DRIVER Control IABORT, value 5).

Dependencies: ENDSNP, IVDIM (see DRIVER Controls Document)

For Control BEGSNP, the number of Elements in a Group is defined by the number of components of the Independent Variable, given by DRIVER Control IVDIM.

Controls BEGSNP and ENDSNP work as ordered pairs describing the beginning and the ending of a Snapshot Period, that is, the Mth Group of Control BEGSNP marks the beginning of the Mth Snapshot Period, and the Mth Group of Control ENDSNP marks the ending of that same period. Therefore, these two Controls must have the same number of Groups.
In addition, Snapshot Periods may not overlap. This implies that the next Group of Control BEGSNP must specify an Independent value greater than that of the Previous Group of Control ENDSNP.

Comments: The number of Elements in a Group for Control BEGSNP is initially defined in the Programmer Directive file for this Operation to be three (3).

The Snapshot Periods must be an increasing function of the Independent Variable. Therefore, the Groups of Control BEGSNP must be arranged in increasing order.

All Snapshot Periods must lie within the interval of the Independent Variable delimited by Controls BEGIV and ENDIV.

Although the DESP Operation is a Transformation Operation, it employs the Snapshot periods as intervals over which despike statistics will be gathered and output to the logical device defined by the value of Control KOUT.

2.2 ENDSNP

Purpose: Marks the ending of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: \(\text{ENDSNP} = (\text{real1},...,\text{realN}),...,\text{realN})\)

where the number of Elements in each Group, \(N\), is defined by Control IVDIM,

and where each Element represents a component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then,

\(\text{ENDSNP} = (17., 6., 30.)\)

would cause a Snapshot Period to end at hour 17, minute 6, and second 30.

Initialized Value: \(\text{ENDSNP} = (99., 99., 99.)\)

This value represents the ending of one Snapshot of an Independent Variable with three components.

Diagnostics:
***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'ENDSNP' LENGTH 'G' IS INVALID. CHANGE LENGTH TO 'G' IN PD FILE
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL****

Cause: The Group length of Control ENDSNP is not equal to the number of
dimensioned components of the Independent Variable, $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number.
PD release numbers must match the code release numbers. The number of
Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM
of the associated code release.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE 'G' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE 'G'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL****

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP,
the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Resspecify consistent values for BEGSNP and ENDSNP.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 70 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAPSHOT INTERVAL 'G' 'ENDSNP' = 'G' EXCEEDS 'ENDSNP' ='G' OF PRIOR INTERVAL
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL****

Cause: The value of Control ENDSNP is less than the ENDSNP of the pre-
vious Snapshot Period.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Resspecify consistent values for the Snapshot Periods.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
INCONSISTENT SNAPSHOT INTERVALS 'G' THROUGH 'G' Have been TURNED OFF.
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL****

Cause: A previous fatal error was detected in either Control BEGSNP or
ENDSNP.

Program action: all Snapshot Periods starting with the first one to produce a fatal error condition are deactivated.

Fix: respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAP INTERVAL 'I' 'BEGSNP' = '$' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE.
DID YOU WANT TO SKIP A CYCLE?

***** WARNING********WARNING********WARNING********WARNING********WARNING*****

Cause: The value of Control BEGSNP is one Cycle larger than the ENDSNP of the previous Snapshot Period.

Program action: A warning message is printed.

Fix: if the intent is to display all Cycles Respecify the BEGSNP of interval I equal to the ENDSNP of interval I-1.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
"CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
MESSAGE NO. 61 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
OPERATION '$' TERMINATED; NUMCYC SET TO 0

****FATAL********FATAL********FATAL********FATAL********FATAL********FATAL****

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, (2) the Operation in question is removed from processing.

Fix: Since these intervals were previously examined in DRIVER routine LODCOM, the code of the Operation in question might have modified them, or random data stores might have overwritten them. See DRIVER Controls DEBUG and DUMP for debug aid.

Multiple Snapshot Periods (Groups) can be defined on the total run interval from BEGV to ENDIV; however, these intervals cannot overlap, but the start of the current interval should repeat the end of the last interval for continuous processing. For example,
BEGIV = (12.,20.,00.) $ \quad$ ENDIV = (14.,24.,00.)

Over this run interval a legal set of Snapshot Periods would be:

interval 1 2 3

BEGSNP = (12.,00.,00.) , (12.,30.,00.) , (14.,00.,00.)
ENDSNP = (12.,30.,00.) , (13.,30.,00.) , (99.,99.,99.)

Note that the intervals from 13.,30.,00. to 14.,00.,00. and 14.,20.,00. to 14.,24.,00. will not generate a display.

Also note that the last Snapshot Period exceeds ENDIV for this Case. The DRIVER will reset it to ENDIV. A similar truncation of a BEGSNP that starts before BEGIV would occur.

Finally note that the start of interval two equals the end of interval one. This insures continuous output. A warning is issued if the start of a subsequent interval is exactly one time step (see DRIVER Control DELIV) larger than the ending of a previous interval.

Over this run interval an illegal set of Snapshot Periods would be:

interval 1 2 3

BEGSNP = (00.,00.,00.) , (12.,20.,00.) , (14.,00.,00.)
ENDSNP = (12.,30.,00.) , (13.,30.,00.) , (13.,20.,00.)

The second interval above is illegal because the BEGSNP of (12.,20.,00.) is less than the ENDSNP of interval one.

Also, the third interval above is illegal because the BEGSNP of (14.,00.,00.) is larger than the ENDSNP of (13.,20.,00.).

NOTE: These are fatal conditions and no processing of a display period with a fatal error nor any subsequent intervals is done even if the processor has been instructed to continue in spite of fatal errors (see DRIVER Control IABORT, value 5).

Dependencies: BEGSNP, IVDIM (see DRIVER Controls Document)

For Control ENDSNP, the number of Elements in a Group is defined by DRIVER Control IVDIM.

Controls BEGSNP and ENDSNP work as ordered pairs describing the beginning and the ending of a Snapshot Period, that is, the Mth Group of Control BEGSNP marks the beginning of the Mth Snapshot Period, and the Mth Group of Control ENDSNP marks the ending of that same period. Therefore, these two Controls must have the same number of Groups.

In addition, Snapshot Periods may not overlap. This implies that the
next Group of Control BEGSNP must specify an Independent Variable value
greater than that of the previous Group of Control ENDSNP.

Comments: The number of Elements in a Group for Control ENDSNP is initially
declared in the Programmer Directive file for this Operation to be three
(3).

The Snapshot Periods must be an increasing function of the Independent
Variable. This implies that the Groups of the Control ENDSNP must be
arranged in increasing order.

All Snapshot Periods must lie within the interval of the Independent
Variable delimited by DRIVER Controls BEGIV and ENDIV.

Although the DESP Operation is a Transformation Operation, it employs
the Snapshot periods as intervals over which despiking statistics will be
gathered and output to the logical device defined by the value of Control
KOUT. **NOTE:** The last Cycle to be displayed is one Cycle **BEFORE**
the value given in Control ENDSNP.

2.3 **FLUSHP**

**Purpose:** Sets the length of a Flush Period.

A Flush Period is a sub-interval of the Snapshot Period marking the
frequency of a specific action. At the end of the specific action, all
data within the Flush Period is discarded, or "flushed", and execution
on subsequent data within the Snapshot Period is continued.

**Form:** FLUSHP = integer

where 'integer' is the number of Cycles in the Flush Period.

**Initialized Value:** FLUSHP = 900

For data with an Independent Variable of time with each Cycle
representing one second in time, this number represents fifteen minutes
of data.

**Diagnostics:**

```
***** OPERATION NO. $ ***** DESP ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'FLUSHP' MISSING FROM CONTROL ARRAY
****FATAL****FATAL****FATAL****FATAL****FATAL****FATAL****

Cause: Control FLUSHP is missing from the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution

May 1, 1983
```

Release 1.0
continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control FLUSHP is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 6 FROM SUBPROGRAM LODCOM AT SECTION 3.5 EX STAGE IS 2
CONTROL 'FLUSHP' VALUE '$' INVALID; VALID RANGE '$' TO '$'; ASSUMING '$'
****WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The value of Control FLUSHP is less than or equal to zero (0.).

Program action: (1) A warning message is printed, and (2) the interval for displays, FLUSHP, is reset to the full run interval of the Case.

Fix: If specific intervals for display are wanted, respecify and correlate Controls, BEGSNP, ENDSNP, and FLUSHP.

Dependencies: BEGSNP, ENDSNP, DELIV (see DRIVER Controls Document)

Controls BEGSNP and ENDSNP define the Snapshot Periods of which the Flush Period is a sub-interval.

Comments: The Flush Period begins with the first Cycle of data in the Snapshot Period. A counter is employed to count the number of Cycles from the beginning of the Flush Period to the length of the Flush Period defined in Control FLUSHP or until the end of the Snapshot Period. If the number of Cycles remaining in a Snapshot Period is greater than the Flush Period, then a "flush" is performed, the count of Cycles in a Flush Period is reset to zero (0), and a new Flush Period is begun. If the number of Cycles remaining in the Snapshot Period is less than the Flush Period, then a "flush" is performed on an abbreviated Flush Period, marked by the end of the Snapshot Period.

Although the DESP Operation is a Transformation Operation, it employs the Flush Periods as intervals over which despike statistics will be re-initialized for the gathering and output to the logical device defined by the value of Control KOUT.
2.4 ITYPOP (Derived)

Purpose: Specifies the type of Operation.

Form: ITYPOP = integer

where 'integer' is one of:

-1 In-Place Transformation Operation
0 Snapshot Operation
1 Input Operation
2 Transformation Operation

Initialized Value: ITYPOP = 2

Diagnostics:

----- OPERATION NO. $ ----- DESP ----- OPERATION NO. $ ----- MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2 CONTROL 'ITYPOP' MISSING FROM CONTROL ARRAY
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL******

Cause: The Control ITYPOP was not found in the PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control ITYPOP is not in the PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

----- OPERATION NO. $ ----- DESP ----- OPERATION NO. $ ----- MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2 CONTROL 'ITYPOP' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2 OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******

Cause: The value of Control ITYPOP is less than a minus one (-1) or greater than two (2).
Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****

MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4  EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4  EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies: NUMCYC

Comments: The value of this Control is derived from the Programmer Directive file.

The DESP Operation is a Transformation Operation.

2.5 KOUT

Purpose: Designates the logical names of the devices on which the despike statistics generated by this Operation are output.

Form: KOUT = name1, ..., nameN

where N is the number of devices on which the statistics are to be output, and is greater than, or equal to one (1),

where 'name1' through 'nameN' must be one of:

Release 1.0 May 1, 1983
KPROC  The generated despike statistics will be written to the unit specified by the DRIVER Control KPROC
KPRINT The generated despike statistics will be written to the unit specified by the DRIVER Control KPRINT
KFILM  The generated despike statistics will be written to the unit specified by the DRIVER Control KFILM
KFICHE The generated despike statistics will be written to the unit specified by the DRIVER Control KFICHE
NONE  This value is used as an optional terminator to the list of logical names of devices which, when present, will cause all values given after it to be ignored

Initialized Value: KOUT = KPROC

Diagnostics:

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM DESP & INDESP EX STAGE IS 5
CONTROL 'KOUT' MISSING FROM CONTROL ARRAY
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: Control KOUT missing from the Programmer Directive file.
Program action: 1) skip output, 2) fatal error flag set.
Fix: Control KOUT must be in Programmer Directive file.

Dependencies: KPROC, KPRINT, KFILM, KFICHE (see DRIVER Controls document)

Comments: Control KOUT allows the despike statistics generated in the DESP Operation to be written out to any of a number of logical units. Even multiple copies may be sent to the same device by specifying that logical unit name more than once.

The assignments of the logical unit names to a logical unit number is made in the DRIVER Controls listed above. The meanings ascribed to each name is also given in the documentation for those Controls.

2.6 LOUT

Purpose: Enumerates a set of Controls for which values of every Dependent Variable will be printed in columnar form in the output generated by the DESP Operation.
Form: \( \text{LOUT} = \text{name}_1, \ldots, \text{name}_N \)

where \( N \) is the number of Controls which are to head columns in the printed output,

and where each 'name' is one of:

- **STOP**: This value is used as an optional terminator to the list of names of Controls which, when present, will cause all values given after it to be ignored.

- **DELTOL**: The value delta tolerance between two adjacent values for each Dependent Variable will be printed.

- **LSHFTS**: The total number of Level Shifts detected for each Dependent Variable will be printed.

- **NUMGAP**: The number of gaps in the data (marked by the value given by DRIVER Control GAPVAL) for each Dependent Variable will be printed.

- **NUMPTS**: The number of data values removed by interpolation for each Dependent Variable will be printed.

- **NUMSPK**: The number of Spikes detected for each Dependent Variable will be printed.

- **NUMUNC**: The number of Spikes detected, but unable to be corrected through interpolation, for each Dependent Variable will be printed.

- **NWIDTH**: The maximum number of data values that a correctable Spike may contain for each Dependent Variable will be printed.

- **RATE**: The number of data values per Cycle to be used in computing statistics for each Dependent Variable will be printed.

- **TOTOPS**: The total number of data values involved in the search for data values for each Dependent Variable will be printed for each Dependent Variable will be printed.

- **UNITS**: The units of measurement for each Dependent Variable will be printed.

**Initialized Value:** \( \text{LOUT} = \text{UNITS}, \text{RATE}, \text{TOTPTS}, \text{NUMSPK}, \text{NUMPTS}, \text{LSHFTS}, \text{NUMUNC}, \text{NUMGAP}, \text{DELTOL}, \text{NWIDTH}, \text{STOP} \)

**Diagnostics:**

**Release 1.0**

**May 1, 1983**
MESSAGE NO. 4 FROM SUBPROGRAM DSPOUT EX STAGE IS 5

CONTROL 'LOUT' VALUE ' $ ' NOT IMPLEMENTED

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The list of Controls specified in LOUT contains a Control name which is not an existing Dependent Variable Control. This usually means a Control name has been misspelled.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the LOUT list and verify that all Control names found in this list are valid for this Operation.

The initialized value causes the most useful statistics to be printed.

Comments: The name of each Dependent Variable, given by Control NAMVAR, will automatically appear as the first column in all printed output, and thus, does not need to be specified as a value of the LOUT Control.

The order of values for Control LOUT is not significant. The order that the columns of values for Controls will take is dependent on the order that Controls appear in the Control Array.

The values given to Control LOUT must match exactly the desired Control given in the Variable Controls section of the Control Array for the DESP Operation.

2.7 MAXCHR

Purpose: Specifies the maximum number of characters which can be printed on one line of the output device to which tabular output is to be written.

Form: MAXCHR = integer

where 'integer' is positive and is dependent on the physical limitation of the output device.

Initialized Value: MAXCHR = 128

Comments: This Control is useful for statistical and tabular displays. Standard output to a paper printer might use 128 characters per line while terminal output might be restricted to 78 characters per line.

DESP uses MAXCHR to regulate the columnar width of its despike statistics tabular display.
2.8 **NAMEOP (Derived)**

*Purpose:* Assigns a unique name to this Operation, which is used in normal and diagnostic prints.

*Form:* \( \text{NAMEOP} = \text{name} \)

*Initialized Value:* \( \text{NAMEOP} = \text{DESP} \)

*Comments:* The value of this Control is derived from the Programmer Directive file.

2.9 **NAMLIB**

*Purpose:* Provides file name(s) which comprise all required code for this Operation if stored on some permanent storage medium (such as PLIB on the CDC 7600).

*Form:* \( \text{NAMLIB} = \text{string}_1, \ldots, \text{string}_N \)

where each 'string' is a file name.

*Initialized Value:* \( \text{NAMLIB} = \text{'}NONE\text{'} \)

*Comments:* This Control is currently nonfunctional.

2.10 **NAMPD**

*Purpose:* Provides the name of the Programmer Directive file for this Operation.

*Form:* \( \text{NAMPD} = \text{string} \)

*Initialized Value:* \( \text{NAMPD} = \text{'}NONE\text{'} \)

*Comments:* This Control is currently nonfunctional.

2.11 **NUMCYC**

*Purpose:* Defines the amount of space, in units of the number of data Cycles, which is to be allocated for this Operation's use.

*Form:* \( \text{NUMCYC} = \text{integer} \)

where 'integer' is one of:
OPERATION CONTROLS

>0 That number of Cycles are allocated, suspending the automatic
Cycle allocation algorithm of the DRIVER for this Operation.

0 Execution of this Operation is suspended for this run.

-1 The DRIVER optimizes the number of Cycles to be allocated based
on the best flow for all Operations involved in this run.

-2 Available only for select Snapshot Operations, this value
instructs the DRIVER that no space is to be allocated for data
Cycles for this Operation.

Initialized Value: NUMCYC = -1

Diagnostics:

***** OPERATION NO. $ **** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'NUMCYC' VALUE 'x' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value of Control NUMCYC is less than -2.

Program action: (1) A fatal error flag is set, (2) the involved Opera-
tion is turned off by setting NUMCYC = 0, and (3) execution continues
subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number.
PD release numbers must match the code release numbers. Also check the
ordering and completeness of the PD and UD Directive sets. If the
problem does not surface, contact a GENPRO consultant.

***** OPERATION NO. $ **** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC
equals a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Opera-
tion is turned off by setting NUMCYC = 0, and (3) execution continues
subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero
defines a Transformation Operation, whereas, NUMCYC = -2 is only valid
for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies: ITYPOP, RATE

Comments: A positive value for Control NUMCYC results in the automatic optimal allocation of space to be turned off and that number of Cycles to be allocated instead. This option should be used with care, since it suspends the optimal flow of data in the processor and may thus cause bottlenecks in the flow and increase the execution time of the entire processor. by employing a positive value, there should be good reason for its use.

A zero (0) value for Control NUMCYC causes this Operation to be turned off. This allows an Operation to be turned off without removing the Programmer Directive and the User Directive files.

2.12 OVRLAP

Purpose: Defines the number of Cycles on each side of the current Cycle(s) required to carry out the requested computations.

Form: \( OVRLAP = \text{integer} \)

where 'integer' is a small, non-negative number representing the number of Cycles on each side of the current computational Cycles.

Initialized Value: \( OVRLAP = 0 \)

Diagnostics:

\[ \text{**** OPERATION NO. } $ \text{ ***** DE S P ***** OPERATION NO. } $ \text{ *****} \]
\[ \text{MESSAGE NO. 11 FROM SUBPROGRAM INDESP EX STAGE IS 2} \]
\[ \text{CONTROL 'OVRLAP' VALUE '$' IS LARGE; IS THIS WHAT YOU INTENDED?} \]
\[ \text{****WARNING****WARNING****WARNING****WARNING****WARNING****WARNING****} \]

Cause: The value of Control OVRLAP is greater than three (3).

Program action: A warning diagnostic is generated.

Fix: Check the value of the Control NWIDTH for all Dependent Variables. If the values for Control NWIDTH are correct, check the rates of the Dependent Variables. Perhaps the rates have inadvertently been set lower than the input rates. If you are convinced that you want to remove Spikes that cover more than three (3) Cycles, proceed and ignore the above warning.
Comments: Normally this Control is internally computed in the DESP Operation based upon the values of the Controls related to Spike removal. A user would override that computation (setting Control OVRLAP to a positive value) only if there is a specific reason to do so.

2.13 PROJECT

Purpose: Provides a textual description or title of the project, which is used to label the various output generated by this Operation.

Form: PROJECT = string

where 'string' may be up to forty (40) characters in length.

Initialized Value: PROJECT = '

that is, forty (40) blank characters.
3. VARIABLE CONTROLS

3.1 DELTOL

Purpose: Defines a maximum allowable difference between two adjacent values considered to be within the tolerance range of this Dependent Variable. All differences above this value are considered to be either Spikes or Level Shifts.

Form: DELTOL = real

where 'real' must be one of:

0. No removal of Spikes is to be performed, that is, this value is equivalent to a maximum allowable difference of positive infinity

>0. Defines the maximum allowable difference between two adjacent values

Initialized Value: DELTOL = 0.

Diagnostics:

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 13 FROM SUBPROGRAM INDESP EX STAGE IS 2
CONTROL 'DELTOL' VALUE '$' ITEM '$' INVALID; ASSUMING '0'
***FATAL*******FATAL******FATAL******FATAL******FATAL*******FATAL***

Cause: The value of Control DELTOL is less than zero (0.0).

Program action: (1) The value of Control DELTOL is set to zero (0.0), (2) fatal error flag set, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: set Control DELTOL to a value greater than or equal to zero (0.0).

Dependencies: NWIDTH

Comments: If Spikes are to be removed, that is, a positive value is given to Control DELTOL, then a positive value for the maximum number of data values that a correctable Spike may contain (given in Control NWIDTH) must also be positive for any Spikes to be removed.

The maximum allowable difference between two adjacent values, or tolerance, is determined by factors such as speed of sensor response or probability density functions.
3.2 INDGET (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the input to this Operation for this call, is obtained.

Form: INDGET = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INDGET = 0

Diagnostics:

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDGET' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Control INDGET was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDGET is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The data indices associated with Control INDGET are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDGET is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indecies throughout the entire program flow. If an INDGET index lies outside of

May 1, 1983

Release 1.0
the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: INRATE

Comments: The value of Control INDGET is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control INRATE, and by the number of Cycles passed to this Operation in this call, are stored contiguously.

3.3 INDPUT (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the result from this Operation for this call, is stored.

Form:  INDPUT = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INDPUT = 0

Diagnostics:

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDPUT' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****

Cause: The Control INDPUT was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDPUT is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.
Cause: The data indices associated with Control INDPUT are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDPUT is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indices throughout the entire program flow. If an INDPUT index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: RATE

Comments: The value of Control INDPUT is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control RATE, and by the number of Cycles available to this Operation in this call, are stored contiguously.

3.4 INRATE (Derived)

Purpose: Specifies the number of data values per Cycle to be input to this Operation for this Dependent Variable.

Form: INRATE = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INRATE = 0

Diagnostics:
****** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INRATE' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '5' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The Control INRATE was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INRATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

****** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

****** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOBMV AT SECTION 1.2 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

Program action: (1) A fatal flag is set, (2) the data move is not performed.

Fix: Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.

Release 1.0
May 1, 1983
Dependencies: INDGET, RATE

Comments: The value assigned to Control INRATE is derived in the INITIO routine of the DRIVER from the value of Control RATE of the last Transformation Operation.

When the value of Control INRATE is zero (0), no data is to be passed to this Operation, and thus no function is performed for this Dependent Variable.

The number of data values per Cycle given in Control INRATE are stored contiguously, with the address of the first data value defined by Control INDGET.

3.5 LSHFTS (Derived)

Purpose: Counts the number of Level Shifts that have been detected for a Dependent Variable.

Form: LSHFTS = integer

where 'integer' is non-negative.

Initialized Value: LSHFTS = 0

Comments: Control LSHFTS is derived in the DESP Operation. It is initialized to zero (0) at the beginning of each Flush Period, and is incremented by one (1) each time a Level Shift is detected for this Dependent Variable.

The value of this Control is printed at the end of each Flush Period.

3.6 NAMVAR (Derived)

Purpose: Assigns a name to be associated with a Dependent Variable.

Form: NAMVAR = name

Initialized Value: NAMVAR = NAMVAR

Comments: The value of Control NAMVAR is indirectly derived in the Translator by the Area Vector directives which manipulate items of the Variable Area Vector.
3.7 NUMGAP (Derived)

Purpose: Counts the number of missing data values for a Dependent Variable.

Form: NUMGAP = integer

where 'integer' is non-negative.

Initialized Value: NUMGAP = 0

Dependencies: GAPVAL (see DRIVER Controls document)

Comments: Control NUMGAP is derived in the DESP Operation. It is initialized to zero (0) at the beginning of each Flush Period, and is incremented by one (1) each time a missing data value is detected for this Dependent Variable.

The value of this Control is printed at the end of each Flush Period.

A missing data value is set to the special value defined in DRIVER Control GAPVAL.

3.8 NUMPTS (Derived)

Purpose: Counts the number of data values which have been removed by interpolation for a Dependent Variable.

Form: NUMPTS = integer

where 'integer' is non-negative.

Initialized Value: NUMPTS = 0

Comments: Control NUMPTS is derived in the DESP Operation. It is initialized to zero (0) at the beginning of each Flush Period, and is incremented by a count of the number of data values included in the interpolation of a given Spike.

The value of this Control is printed at the end of each Flush Period.

3.9 NUMSPK (Derived)

Purpose: Counts the number of Spikes detected for this Dependent Variable.

Form: NUMSPK = integer

where 'integer' is non-negative.
Initialized Value: NUMSPK = 0

Comments: Control NUMSPK is derived in the DESP Operation. It is initialized to zero (0) at the beginning of each Flush Period, and is incremented by one (1) each time a Spike is detected for this Dependent Variable.

The value of this Control is printed at the end of each Flush Period.

3.10 NUMUNC (Derived)

Purpose: Counts the number of detected Spikes for a Dependent Variable which cannot be corrected through interpolation.

Form:   NUMUNC = integer

where 'integer' is non-negative.

Initialized Value: NUMUNC = 0

Dependencies: NWIDTH

Comments: Control NUMUNC is derived in the DESP Operation. It is initialized to zero (0) at the beginning of each Flush Period, and is incremented by one (1) each time a Spike detected for this Dependent Variable that cannot be corrected through interpolation.

The value of this Control is printed at the end of each Flush Period.

An uncorrectable Spike is defined to be a Spike which contains more data values than the maximum number specified in Control NWIDTH.

3.11 NWIDTH

Purpose: Declares the maximum number of data values that a correctable Spike may contain.

Form:   NWIDTH = integer

where 'integer' is one of:

0    Spikes having a width of zero (0) data values are to be removed, that is, no Spikes are to be removed

>0    Defines the maximum number of consecutive data values in a correctable Spike

Initialized Value: NWIDTH = 0
Diagnostics:

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 13 FROM SUBPROGRAM INDESP EX STAGE IS 2
CONTROL 'NWIDTH' VALUE '$' ITEM '$' INVALID; ASSUMING '0'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control NWIDTH is less than zero (0).

Program action: (1) The value of Control NWIDTH is set to zero (0).
(2) A fatal error flag is set, and (3) execution continues subject to
the value assigned to DRIVER Control IABORT.

Fix: Set the Control NWIDTH to a non-negative value.

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 20 FROM SUBPROGRAM INDESP EX STAGE IS 2
CONTROL 'NWIDTH' VALUE '$' ITEM '$' IS LARGE; IS THIS WHAT YOU INTENDED?
***WARNING******WARNING******WARNING******WARNING******WARNING******WARNING***

Cause: The value of Control NWIDTH is greater than 3*INRATE.

Program action: A warning message produced.

Fix: Set the value of Control Nwidth to a value less than or equal to
3*INRATE.

Dependencies: DELTOL

Comments: If the value of Control DELTOL is zero (0.), then the value of Con-
trol NWIDTH is ignored.

If the value of Control NWIDTH is a large value, then interpolation
would be performed over a large number of data values, causing possible
perturbations in the information spectrum.

3.12 RATE (Derived)

Purpose: Specifies the number of data values per Cycle to be stored by this
Operation for this Dependent Variable.

Form: RATE = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: RATE = 1
Diagnoses:

**** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'RATE' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

If NUMCYC = -1, or positive, the RATE Control must appear in the Programmer Directive file.

Cause: The Control RATE was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control RATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

**** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

**** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOMV AT SECTION 1.2 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

Cause: The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

Program action: (1) A fatal flag is set, (2) the data move is not performed.

Fix: Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls.
INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.

Dependencies: INRATE, INPDU

Comments: The number of data values per Cycle given in Control RATE are stored contiguously, with the address of the first data value defined by Control INPDU.

The value of Control RATE is set by the DESP Operation to be exactly the same value as that of Control INRATE.

When the value of Control RATE is zero (0), no data values are to be stored and therefore processing of this Dependent Variable is not done.

The number of data values per Cycle given in Control RATE are stored contiguously.

3.13 SOURCE (Linked)

Purpose: Names a Source Variable for this Dependent Variable.

A Source Variable is defined to be a Dependent Variable whose data values are obtained for processing in place of obtaining this Dependent Variable's data values.

Form: SOURCE = name

where 'name' must be one of:

NOLINK No Source Variable is employed for this Dependent Variable, that is, the data values from this Dependent Variable are obtained for processing

mmm A value which exactly matches the name of a Dependent Variable as given in the NAMVAR Control, to be used as the Source Variable for this Dependent Variable.

Initialized Value: SOURCE = NOLINK

Diagnostics:

***** OPERATION NO. $ ***** DESP ***** OPERATION NO. $ *****
MESSAGE NO. 15 FROM SUBPROGRAM INDESP EX STAGE IS 2
CONTROL 'SOURCE' VALUE 'S' ITEM 'S' INVALID: LEGAL RANGE '0' TO 'NUMVEC'
**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**

Cause: The value of Control SOURCE is not NOLINK or does not match a
name as given in the NAMVAR Control for any Dependent Variable available to this Operation. This usually means that a Dependent Variable name has been mis-spelled.

Program action: (1) The value of Control SOURCE is set to zero (0). (2) A fatal error flag is set, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Correct any mis-spelled Dependent Variable names, or replace the incorrect values with the value NOLINK.

Comments: Normally, the data values to be processed are obtained from the subscript index defined in Control INDGET for this Dependent Variable, and are stored at the subscript index defined in Control INDPUT for this Dependent Variable. Employing a Source Variable, the data values to be processed are obtained from the subscript index defined in Control INDGET for SOME OTHER DEPENDENT VARIABLE, and are stored at the subscript index defined in Control INDPUT for this Dependent Variable.

This Control therefore facilitates the creation of new Dependent Variables from existing ones.

When the value of Control SOURCE is not NOLINK, then a Source Variable is being employed. The implications of this statement are:

1. Since the value of Control INDGET for the Source Variable is being used, the value of Control INDGET for this Dependent Variable is ignored.

2. Since the value of Control INRATE for the Source Variable is being used, the value of Control INRATE for this Dependent Variable is ignored.

3. Since the value of Control INRATE for the Source Variable is being used, it must be evenly divisible by the value of Control RATE for this Dependent Variable.

3.14 TOTPTS (Derived)

Purpose: Counts the total number of data values involved in the search for data values.

Form: TOTPTS = integer

where 'integer' is non-negative.

Initialized Value: TOTPTS = 0

Diagnostics:
MESSAGE NO. 2 FROM SUBPROGRAM INDESP EX STAGE IS 2
CONTROL 'TOTPTS' MISSING FROM CONTROL ARRAY

***** FATAL*******FATAL********FATAL********FATAL********FATAL********FATAL***

** Cause: Control TOTPTS missing from the Programmer Directive file. 

** Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT. 

** Fix: Control TOTPTS must be in Programmer Directive file. 

** Comments: Control TOTPTS is derived in the DESP Operation. It is initialized to zero (0) at the beginning of each Flush Period, and is incremented by one (1) each time a data point is considered in the search for a Spike for this Dependent Variable.

The value of this Control is printed at the end of each Flush Period.

The number of data values involved in Spikes, given in Control NUMPTS, divided by the total number of data values considered in looking for data Spikes, given in Control TOTPTS, produces the total percentage of data values involved in data Spikes, giving a statistic on the noise level of this Dependent Variable in this Flush Period.

3.15 TRPTYP

** Purpose: Selects the type of interpolation to be used in removing a Spike from the data values of a Dependent Variable.

** Form: TRPTYP = integer

where 'integer' is be one of:

1  Linear interpolation is performed to remove a Spike
2  Quadratic interpolation is performed to remove a Spike
3  Cubic interpolation is performed to remove a Spike

** Initialized Value: TRPTYP = 1

** Diagnostics:

***** OPERATION NO. $ ***** D E S P ***** OPERATION NO. $ *****
MESSAGE NO. 14 FROM SUBPROGRAM INDESP EX STAGE IS 2
CONTROL 'TRPTYP' VALUE '$' ITEM '$' INVALID; LEGAL RANGE '0' TO '3'; ASSUMING '0'
***FATAL*******FATAL********FATAL********FATAL********FATAL********FATAL***

Release 1.0  May 1, 1983
Cause: The value of Control TRPTYP is not one of the legal values mentioned.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Set Control TRPTYP to one of the legal values mentioned.

Comments: Linear interpolation requires the least amount of calculation of the three types, and is therefore, the most efficient, but the least accurate. On the other extreme, cubic interpolation is the most accurate, being derivative continuous, but also the least efficient.

3.16 UNITS

Purpose: Defines the units of measurement of a Dependent Variable.

Form: UNITS = string

where 'string' may be up to eight (8) characters in length, where all characters are from the set of legal characters available on the target machine.

Initialized Value: UNITS = ' 

that is, eight (8) blank characters.
GENPRO INPUT Operation
REFERENCE MANUAL

Editors:  Bonnie Gacnik
         Bob Lackman

SCIENTIFIC COMPUTING DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO
INPUT CONTROLS

CONTENTS

1. INTRODUCTION ................................................................. 3-INPUT.1

2. GENERAL CONTROLS ............................................................... 3-INPUT.3
   2.1 BITKEY ................................................................. 3-INPUT.3
   2.2 CONV ................................................................. 3-INPUT.3
   2.3 DATBIT ................................................................. 3-INPUT.5
   2.4 DATLOG ................................................................. 3-INPUT.5
   2.5 DATOPT ................................................................. 3-INPUT.6
   2.6 DATSIZ ................................................................. 3-INPUT.7
   2.7 FRTIM ................................................................. 3-INPUT.7
   2.8 I (Derived) ............................................................. 3-INPUT.7
   2.9 IKOUNT (Derived) ..................................................... 3-INPUT.8
   2.10 ITYPOP (Derived) .................................................... 3-INPUT.9
   2.11 IVNAM (Linked) ...................................................... 3-INPUT.11
   2.12 IVPR ................................................................. 3-INPUT.13
   2.13 IVTYPE ............................................................... 3-INPUT.13
   2.14 JCL ................................................................. 3-INPUT.15
   2.15 KEOF ................................................................. 3-INPUT.17
   2.16 KIN ................................................................. 3-INPUT.17
   2.17 KMODE ............................................................... 3-INPUT.18
   2.18 KOUNT ............................................................... 3-INPUT.19
   2.19 KPOS ................................................................. 3-INPUT.19
   2.20 KSTOP (Derived) .................................................... 3-INPUT.20
   2.21 KTYPE ............................................................... 3-INPUT.21
   2.22 LOGBIT ............................................................... 3-INPUT.21
   2.23 MAXIV ................................................................. 3-INPUT.22
   2.24 MEDIA (Derived) ..................................................... 3-INPUT.22
   2.25 NAMEOP (Derived) ................................................... 3-INPUT.23
   2.26 NAMKEY (Linked) .................................................... 3-INPUT.23
   2.27 NAMLIB ............................................................. 3-INPUT.24
   2.28 NAMPD ............................................................... 3-INPUT.24
   2.29 NAMUD ............................................................... 3-INPUT.24
   2.30 NEWV (Derived) ..................................................... 3-INPUT.24
   2.31 NOLDV (Derived) .................................................... 3-INPUT.25
   2.32 NPHY (Derived) ..................................................... 3-INPUT.25
   2.33 NSEC ................................................................. 3-INPUT.25
   2.34 NTOT (Derived) ..................................................... 3-INPUT.26
   2.35 NUMCYC ............................................................. 3-INPUT.27
   2.36 NUMLOG (Derived) .................................................. 3-INPUT.28
   2.37 NUMV (Derived) .................................................... 3-INPUT.28
   2.38 NWOL ................................................................. 3-INPUT.29
   2.39 OFFSET ............................................................. 3-INPUT.30

3. VARIABLE CONTROLS ............................................................ 3-INPUT.31
   3.1 BITS ................................................................. 3-INPUT.31
   3.2 CONKEY ............................................................. 3-INPUT.31
   3.3 FACTOR .............................................................. 3-INPUT.32
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>FSTBIT</td>
</tr>
<tr>
<td>3.5</td>
<td>INDGET (Derived)</td>
</tr>
<tr>
<td>3.6</td>
<td>INDDUT (Derived)</td>
</tr>
<tr>
<td>3.7</td>
<td>INRATE (Derived)</td>
</tr>
<tr>
<td>3.8</td>
<td>NAMVAR (Derived)</td>
</tr>
<tr>
<td>3.9</td>
<td>RATE</td>
</tr>
<tr>
<td>3.10</td>
<td>SAMPLE</td>
</tr>
<tr>
<td>3.11</td>
<td>SCLKEY</td>
</tr>
<tr>
<td>3.12</td>
<td>SKIP</td>
</tr>
<tr>
<td>3.13</td>
<td>TERM</td>
</tr>
<tr>
<td>3.14</td>
<td>TITLE</td>
</tr>
<tr>
<td>3.15</td>
<td>UNITS</td>
</tr>
</tbody>
</table>
1. **INTRODUCTION**

The INPUT Operation provides the ability to access variables from a variety of data structures, making this data available for further processing by GENPRO. To do this, the user must specify: general information about the INPUT data volume, general information about the data structure, and specific information about the variables to be accessed from this data structure.

The **GENERAL INFORMATION** about the **INPUT DATA VOLUME** would include:

- Access information (name, unit, file number, record number, etc.)
- Type of information on the volume
- Binary (recommended)
- Positive Integer (recommended)
- Signed Integer (discouraged)
- Floating Point (strongly discouraged)
- Character (discouraged)
- ASCII
- DPC
- BCD
- EBCDIC

The **GENERAL INFORMATION** about the **DATA STRUCTURE** would include:

- Physical record information
- The number of bits of data in the physical record
- The number of bits in the physical record
- Logical record information
- The number of bits in the logical record
- The number of logical records in each physical record
- The interval (in units of the INPUT independent variable) between logical records
- INPUT independent variable information relative to the logical record
• INPUT independent variable is a Dependent Variable(s)

• The Dependent Variable(s) which make up the component(s) of the INPUT independent variable

• The computational factor(s) used to combine the component(s) into a composite value called the INPUT independent variable.

• INPUT independent variable is the logical record number.

• If data logical/physical records are not homogeneous, a set of bits in each logical record must indicate which type of logical record has been encountered

• Which datums are to be accessed from the INPUT data volume

• All datums of the data structure may be accessed

• Any subset of the datums of the data structure may be accessed

• All datums of the data structure may be accessed any number of times

• All datums of a variable may be accessed

• Only one datum of a variable may be accessed

• A subset of the total number of datums of a variable may be accessed as long as only one algorithm is used to get from one datum to the next

The **SPECIFIC INFORMATION** about each **Dependent Variable** to be accessed would include:

• The location in the logical record of the first datum of this Dependent Variable

• The number of bits defining an datum

• The number of datums for a Dependent Variable in each logical record

• The algorithm used to get from one datum of a Dependent Variable to the next in the logical record (i.e., the number of bits between consecutive datums)

• Conversion formulas, scaling factors, and ranging algorithms (if any) to be applied to the datum(s) of this Dependent Variable.
2. GENERAL CONTROLS

2.1 BITKEY

Purpose: Specifies the value of the record identifier of data logical records which are to be processed. The record identifier is a bit configuration included in every logical record which is used to differentiate the type of information contained in that record.

Form: BITKEY = integer

where 'integer' may take on any value.

Initialized Value: BITKEY = 0

Dependencies: NAMKEY

Comments: The record identifier, if it exists will become a Dependent Variable when input. The name of that Dependent Variable is given in Control NAMKEY.

The number of bits contained in the record identifier is provided by Control BITS and the location in the data logical record given by Control FSTBIT for the Dependent Variable defined in Control NAMKEY.

When a record identifier exists, it may also be used as a means to detect parity errors. Since the value of the record identifier is fixed for all data logical records, the value obtained from the data logical record can be compared to the value of Control BITKEY, and discrepancies will indicate parity errors.

When NAMKEY and BITKEY are being used in quality control, and when processing has terminated on the input data volume, this diagnostic is written

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 240 FROM SUBPROGRAM DCODE AT SECTION 16.1 EX STAGE IS 4
'$' BIT PATTERN MIS-MATCHES ENCOUNTERED ON UNIT '$'
******INFORMATION******INFORMATION******INFORMATION******INFORMATION******

2.2 CONV

Purpose: Lists the computational factors used to combine various components into a composite value called the INPUT independent variable. It should be noted that the INPUT independent variable is not the Independent Variable of the processor. See details in the Comments section.

Release 1.0

May 1, 1983
below.

Form: \( CONV = \text{real}1, \ldots, \text{real}N \)

Initialized Value: \( CONV = 1 \).

where \( N \) is the number of components used to form the composite INPUT
independent variable,

where 'real1' is the computational factor for the most significant com-
ponent,

and where 'realN' is the computational factor for the least significant
component.

Diagnostics:

***** OPERATION NO. $ ***** INPUT ***** OPERATION NO. $ *****
MESSAGE NO. 1 FROM SUBPROGRAM INIT2 AT SECTION 5.10 EX STAGE IS 2
CONTROL 'CONV' LENGTH '$' INCONSISTENT WITH CONTROL 'IVNAM' LENGTH '!$
***FATAL*******FATAL*********FATAL*********FATAL*********FATAL*********FATAL***

Cause: The number of Groups defined for Control CONV does not equal the
number of Groups defined for Control IVNAM.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the same number of Groups for the Controls CONV and IVNAM.

Dependencies: IVTYPE, IVNAM

Comments: The INPUT independent variable is different from the Independent
Variable of the processor. The differences are:

- The number of components for the INPUT independent variable is not
  necessarily the same as the number of components for the Indepen-
dent Variable of the processor.

- The computational factors used for the INPUT independent variable
  may differ.

- Each component of the INPUT independent variable is in the form of
  data of a Dependent Variable.

- The INPUT independent variable is however in the same units as the
  processor Independent Variable, and is used to synchronize the
  incoming data with the start value given by the Independent Vari-
  able DRIVER Control BEGIV.

If the INPUT independent variable type set through Control IVTYPE is
zero, then the logical record number is used as the INPUT independent variable. In this case, values of CONV and IVNAM are ignored.

Each component of the INPUT independent variable is a Dependent Variable, and has a name, given in Control IVNAM, and a corresponding computational factor, given in Control CONV. Therefore, it is clear that the number of names given must correspond to the number of computational factors defined, or an error diagnostic is generated.

2.3 DATBIT

Purpose: Specifies the number of data bits in a data physical record.

Form: DATBIT = integer

where 'integer' is positive.

Initialized Value: DATBIT = 1

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.10 EX STAGE IS 2
CONTROL 'DATBIT' VALUE '"$' INVALID; VALID RANGE '1', '+N'
**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**FATAL**

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control DATBIT.

Dependencies: DATOPT

Comments: If data logical records may not span data physical records (DATOPT=,NOSPAN), then the value of Control DATBIT is not used.

Currently, Control DATOPT must be set to (,NOSPAN), and Control DATBIT is ignored.

2.4 DATLOG

Purpose: Specifies the number of data logical records per data physical record.

Form: DATLOG = integer
where 'integer' is a positive value representing the number of data logical records per data physical record.

Initialized Value: DATLOG = 1

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.20 EX STAGE IS 2
CONTROL 'DATLOG' VALUE '$' INVALID; VALID RANGE '1', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control DATLOG.

Dependencies: DATSIZ, DATOPT, LOGBIT

Comments: If the data logical record spans the data physical record (DATOPT = (AUTO, SPAN)), and the number of bits in a data logical record (Control LOGBIT) is greater than the number of bits in a data physical record (Control DATSIZ), then the value of Control DATLOG is ignored.

2.5 DATOPT

Purpose: Specifies data record options.

Form: DATOPT = (name1,name2)

where 'name1' and 'name2' specify data record options, and

where 'name1' is not used by the INPUT module and thus a fixed value of AUTO is used,

and where 'name2' must be one of:

SPAN Data logical records may span data physical records; that is, the last data logical record in a data physical record may be continued on a data physical record. A data logical record may span more than one data physical record.

NOSPAN Data logical records may not span data physical records.

Initialized Value: DATOPT = (AUTO,NOSPAN)
MESSAGE NO. 6 FROM SUBPROGRAM INIT2 AT SECTION 7.10 EX STAGE IS 2
CONTROL 'DATOPT' VALUE '$' INVALID; VALID RANGE 'SPAN', 'NOSPAN'; ASSUME 'SPAN'

****WARNING********WARNING********WARNING********WARNING********WARNING*****

Cause: The value of the second Element of Group is not SPAN or NOSPAN.

Program action: The value of second Group Element is set to NOSPAN.

Fix: Define the second Group Element of Control DATOPT to be NOSPAN.

Comments: Currently only the (AUTO, NOSPAN) option has been implemented.

2.6 DATSIZ

Purpose: Specifies the size of a data physical record in bits.

Form: DATSIZ = integer

where 'integer' is equal to the size of the data physical record in bits and must be a number greater than zero (0).

Initialized Value: DATSIZ = 10000

Diagnostics:

**** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.30 EX STAGE IS 2
CONTROL 'DATSIZ' VALUE '$' INVALID; VALID RANGE '1', '+N'

****FATAL********FATAL********FATAL********FATAL********FATAL********FATAL*****

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control DATSIZ.

Dependencies: DATOPT, DATLOG

2.7 FRTIM

Purpose: The interval between data logical records, measured in units of the INPUT independent variable.
Form: \( FRTIM = \text{real} \)

where 'real' is a positive value representing the interval between data logical records in units of the \text{INPUT} independent variable.

Initialized Value: \( FRTIM = 1 \).

Diagnostics:

***** OPERATION NO. $ **** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.40 EX STAGE IS 2
CONTROL 'FRTIM' VALUE '$' INVALID; VALID RANGE '.N', '+N'
***FATAL******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The value is less than or equal to zero (0.0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control FRTIM.

***** OPERATION NO. $ **** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM INIT2 AT SECTION 14.2 EX STAGE IS 2
CONTROL 'FRTIM' VALUE '$' INCONSISTENT WITH CONTROL 'DELIV' VALUE '$'
***FATAL******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: 1) The value of Control FRTIM must evenly devide the Control DELIV (see DRIVER Controls document), and 2) the value the Control FRTIM must be less than or equal to the Control DELIV (see DRIVER Controls document).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the Controls DELIV (see DRIVER Controls document) and FRTIM in such a way as to insure that FRTIM evenly devides DELIV, and FRTIM is less than or equal to DELIV.

Dependencies: SAMPLE, RATE, DELIV (see DRIVER Controls document)

Comments: The Cycle interval, expressed in Driver Control DELIV, must be greater than or equal to, and be evenly divisible by the data logical record interval, defined in Control FRTIM.

2.8 I \( (\text{Derived}) \)

Purpose: Provides the number of the data logical record just processed.
2.9 IKOUNT (Derived)

Purpose: A count of the number of parity errors encountered on the input data volume.

Form: IKOUNT = integer

where 'integer' is non-negative.

Initialized Value: IKOUNT = 0

Dependencies: NAMKEY, BITKEY

Comments: When NAMKEY and BITKEY are being used for quality control information on the input data volume, each time a logical record is encountered in which the Dependent Variable specified in NAMKEY does not contain the value specified in BITKEY a diagnostic message may be written by the INPUT Operation. IKOUNT is initialized at the beginning of execution to a value of zero (0), and incremented each time a parity error is encountered in the input data volume.

2.10 ITYPOP (Derived)

Purpose: Specifies the type of Operation.

Form: ITYPOP = integer

where 'integer' is one of:

-1 In-Place Transformation Operation
0 Snapshot Operation
1 Input Operation
2 Transformation Operation

Initialized Value: ITYPOP = 1

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'ITYPOP' MISSING FROM CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Control ITYPOP was not found in the PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control ITYPOP is not in the PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control ITYPOP is less than a minus one (-1) or greater than two (2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***
Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies: NUMCYC

Comments: The value of this Control is derived from the Programmer Directive file.

2.11 IVNAM (Linked)

Purpose: Lists the names of Dependent Variables which are the components of the INPUT independent variable. It should be noted that this INPUT independent variable is not the Independent Variable used throughout the processor. See details in the Comments section below.

Form: IVNAM = name1, ..., nameN

where N is the number of components used for the composite INPUT independent variable, and is greater than, or equal to, one (1),

where 'name1' is the name for the most significant component,

where 'nameN' is the name for the least significant component,

and where each 'name' matches an item of the Variable Area Vector.

If the INPUT independent variable is not a composite of Dependent Variables, then the value NOLINK is given to 'name1'.

Initialized Value: IVNAM = TIME

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 1 FROM SUBPROGRAM INIT2 AT SECTION 5.10 EX STAGE IS 2
CONTROL 'CONV' LENGTH '�新 inconsistent with CONTROL 'IVNAM' LENGTH '�新
"FATAL"FATAL"FATAL"FATAL"FATAL"FATAL"FATAL"FATAL"FATAL"

Release 1.0 May 1, 1983
Cause: The number of Groups defined for Control IVNAM does not equal the number of Groups defined for Control CONV.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the same number of Groups for the Controls IVNAM and CONV.

**** OPERATION NO. $***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM INIT2 AT SECTION 6.20 EX STAGE IS 2
CONTROL 'IVTYPE', VALUE '$' INCONSISTENT WITH CONTROL 'IVNAM', VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value(s) are not Element(s) of the Area Vector and Control IVTYPE is equal to one (1).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the value for Control IVTYPE to be zero (0) or respecify Control IVNAM to contain name(s) found in the Area Vector.

Dependencies: IVTYPE, CONV

Comments: The INPUT independent variable is a different value than the Independent Variable of the processor. The differences are:

- The number of components for the INPUT independent variable is not necessarily the same as the number of components for the processor Independent Variable.

- The computational factors used for the INPUT independent variable, given by Control CONV, may differ.

- The components of the INPUT independent variable are in the form of data of a Dependent Variable.

- The INPUT independent variable is however in the same units of the processor Independent Variable, and is used to synchronize the incoming data with the start up time given by the processor Independent Variable.

If the INPUT independent variable type set through Control IVTYPE is zero, then the logical record number is used as the INPUT independent variable. In this case, values of IVNAM and CONV are not needed and are thus ignored.

Each component of the INPUT independent variable is a Dependent Variable, and has a name, given in Control IVNAM, and a corresponding computational factor, given in Control CONV. Therefore, it is clear that the number of names given must correspond to the number of...
computational factors defined, or an error diagnostic is generated.

2.12 IVPR

Purpose: Print flag turning on/off the display of INPUT independent variable decode information.

Form: \[ \text{IVPR} = \text{integer} \]

where 'integer' is one of:

0 No display.

1 Activates display of INPUT independent variable decode information and the display of the INPUT data to be passed on to the next Operation.

2 Activates display of the INPUT data to be passed on to the next Operation.

Initialized Value: \( \text{IVPR} = 0 \)

Dependencies: KCHECK (see DRIVER Controls Document)

Comments: These printouts are helpful in tracing setup errors.

These displays are written to the FORTRAN unit defined by DRIVER Control KCHECK.

If a requested display does not occur check the assignment to unit KCHECK.

2.13 IVTYPE

Purpose: Defines the type of the INPUT independent variable.

Form: \[ \text{IVTYPE} = \text{integer} \]

where 'integer' is one of:

0 The INPUT independent variable is the logical record number

1 The INPUT independent variable is to be computed from component(s) obtained from specified Dependent Variable(s).

Initialized Value: \( \text{IVTYPE} = 0 \)
Diagnostics:

**** OPERATION NO. $ **** I N P U T **** OPERATION NO. $ ****
MESSAGE NO. 6 FROM SUBPROGRAM INIT2 AT SECTION 6.10 EX STAGE IS 2
CONTROL 'IVTYPE' VALUE '$' INVALID; VALID RANGE '0', '1'; ASSUMING '0'
***WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The value is less than zero (0) or greater than one (1).

Program action: The value is reset to zero (0).

Fix: Define the value for Control IVTYPE to be zero (0) or one (1).

**** OPERATION NO. $ **** I N P U T **** OPERATION NO. $ ****
MESSAGE NO. 9 FROM SUBPROGRAM INIT2 AT SECTION 6.20 EX STAGE IS 2
CONTROL 'IVTYPE', VALUE '$' INCONSISTENT WITH CONTROL 'IVNAM', VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is equal to one (1) and Group(s) of Control IVNAM contain information not found the Area Vector.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the value for Control IVTYPE to be zero (0) or respecify Control IVNAM to contain names found in the Area Vector.

Dependencies: CONV, IVNAM

Comments: The INPUT independent variable is a different value than the Independent Variable of the processor. The differences are:

- The number of components for the INPUT independent variable is not necessarily the same as the number of components for the processor Independent Variable.
- The conversion factors used for the INPUT independent variable may differ.
- The components of the INPUT independent variable are in the form of data of a Dependent Variable.
- The INPUT independent variable is however in the same units of the Independent Variable, and is used to synchronize the incoming data with the start up time given by the Independent Variable of the processor.

When Control IVTYPE is set to zero (0), the values for Controls CONV and IVNAM are not needed and are thus ignored.
`14 JCL
Purpose: Provides the arguments of the *VOLUME JCL instruction.

******************************************************************************
This Control is a machine dependent Control used only for the CDC 7600.
******************************************************************************

See the Comments section below for further discussion of the usage of
the *VOLUME JCL instruction on the CDC 7600.

Form: JCL = string1,...,stringK, string K+1,...,stringK+L, stringK+L+1,...,
stringK+L+M, ...

where 'string1' to 'stringK' form all the arguments for the first
*VOLUME JCL instruction,

JCL instruction, etc.,

where each 'string' is an argument for the *VOLUME JCL instruction,
where an argument is here defined as being the character string delim-
ited by commas from other arguments, and

where the last argument for a *VOLUME JCL instruction is the terminator
'0'.

For example, assume that the following two *VOLUME JCL instructions
were required for running on the CDC 7600:

*VOLUME,15,VSN=TAPE1,TAPE=16,9,STAGEIN=RT
*VOLUME,15,VSN=TAPE2,TAPE=16,9,STAGEIN=MA

Then Control JCL would be set to the following:

JCL = '*VOLUME','15',VSN=TAPE1','TAPE=16,9','STAGEIN=RT','0',

Initialized Value: JCL = '0'

Diagnostics:

The errors listed below will occur only when running on the NCAR system
CDC 7600. For all other machine configurations, Control JCL is not
used and thus no error diagnostics relating to it will be generated.

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 1 FROM SUBPROGRAM INIT2 AT SECTION 12.1 EX STAGE IS 2
CONTROL 'NWVOL' LENGTH '$' INCONSISTENT WITH CONTROL 'JCL' LENGTH '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The number of sets of Groups terminated by a Group of value '0'
for the Control JCL is not equal to the number of Groups for the
Control NWVOL.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Define the number of sets of Groups terminated by a Group of value '0' for Control JCL to equal the number of Groups for the Control NWVOL.

**Dependencies:** KPOS, NWVOL, NUMV, KIN

The names of the volumes which are to be processed are given, in order, in Control NWVOL. There must be a one-to-one correspondence between a volume names so given and the complete *VOLUME instruction given in Control JCL.

Similarly, the starting file and record location for each volume is given in Control KPOS, and naturally, there must be a one-to-one correspondence to the volume names and the complete *VOLUME instructions given in Controls NWVOL and JCL respectively.

The number of numeric storage units required to contain a complete *VOLUME instruction is defined in Control NUMV. Currently, this Control has only one value, that is, EACH *VOLUME instruction must be the same length.

**Comments:** Control JCL is highly machine dependent, created only for use on the NCAR system CDC 7600.

In addition, it was created only as a temporary solution to the problem of communicating to the operating system during execution time the names of succeeding volumes of data to be mounted and read.

The *VOLUME instruction which is to be input follows exactly the format described in the NCAR Terabit Memory System manual of July 1978, with the following restrictions;

- Each argument of the *VOLUME instruction (each Group) must be less than or equal to ten (10) characters in length.

- The logical unit number must be an integer between 11 and 20, inclusive, and all *VOLUME instructions must employ the same logical unit number.

- The number of arguments (Groups) for each *VOLUME instruction must be the same.
.15 KEOF

Purpose: An endfile flag which may or may not terminate input data volume processing when an endfile is encountered.

Form: KEOF = integer

where 'integer' is one of:

0 Input data volume processing continues when an endfile is encountered.
1 Input data volume processing stops when an endfile is encountered.

Initialized Value: KEOF = 1

Diagnostics:

***** OPERATION NO. $ ***** INPUT ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.11 EX STAGE IS 2
CONTROL 'KEOF' VALUE '$' INVALID; VALID RANGE '1', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value is less than zero (0) or greater than one (1).

Program action: KEOF is set to the value of one (1).

Fix: Define a value for Control KEOF to be zero (0) or one (1).

2.16 KIN

Purpose: Designates the logical unit number of the input device.

Form: KIN = integer

where 10 < 'integer' < 21.

Initialized Value: KIN = 20

Diagnostics:

***** OPERATION NO. $ ***** INPUT ***** OPERATION NO. $ *****
MESSAGE NO. 6 FROM SUBPROGRAM INIT2 AT SECTION 3.50 EX STAGE IS 2
CONTROL 'KIN' VALUE '$' INVALID; VALID RANGE '11', '20'; ASSUMING '15'
****WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The value is less than 11 or greater than 20.
Program action: The value of Control KIN is set to 15.

Fix: Define a value for Control KIN which is greater than 10 and less than 21.

Dependencies: JCL

Comments: The logical unit number specified as part of the complete *VOLUME JCL instruction must contain the same logical unit number defined in Control KIN. If the CDC 7600 is not the host machine, there is no JCL dependency.

2.17 KMODE

Purpose: Specifies the mode of all data which is to be accessed, that is, input by the INPUT Operation. The Control KMODE assumes that the NCAR tape utility RDTAPE is available.

Form: KMODE = integer

where 'integer' is one of:

0  ASCII character data is converted to DPC.
1  Data is read binary bit serial.
2  EBCDIC character data is converted to DPC.

Initialized Value: KMODE = 1

Diagnostics:

***** OPERATION NO.  $ *****  I N P U T  ***** OPERATION NO.  $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.60 EX STAGE IS 2
CONTROL 'KMODE' VALUE '$' INVALID; VALID RANGE '0', '2'
***FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value is less than zero (0) or greater than two (2).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a value for Control KMODE to be zero (0) or one (1) or two (2).
.18 KOUNT

Purpose: A quality control print flag.

Form: KOUNT = integer

where 'integer' is one of:

0  No diagnostic message is written.

1  A diagnostic message is written each time a logical record is encountered of questionable quality.

Initialized Value: KOUNT = 0

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.12 EX STAGE IS 2
CONTROL 'KOUNT' VALUE '$' INVALID; VALID RANGE '1', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than zero (0) or greater than one (1).

Program action: KOUNT is set to the value of zero (0).

Fix: Define a value for Control KOUNT to be zero (0) or one (1).

Dependencies: NAMKEY, BITKEY

Comments: Control KOUNT is derived in the INPUT Operation. When NAMKEY and BITKEY are being used for quality control information on the input data volume, each time a logical record is encountered in which the Dependent Variable specified in NAMKEY does not contain the value specified in BITKEY a diagnostic message may be written by the INPUT Operation. KOUNT determines whether or not this diagnostic message is written each time such a logical record is encountered.

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 237 FROM SUBPROGRAM MINID AT SECTION 7.10 EX STAGE IS 2,3,4
PARITY ERROR; VOLUME '$', FILE '$', RECORD '$', FRAME '$', UNIT '$'
****WARNING**********WARNING**********WARNING**********WARNING**********WARNING****

2.19 KPOS

Purpose: Describes the initial positioning on each volume.
Form: \( \text{KPOS} = (\text{integer}_1, \text{integer}_2), \ldots, (\text{integer}_{N-1}, \text{integer}_N) \)

where the number of Groups represents the number of volumes to be input,

where the first Element of the Group represents the number of files to be skipped from the beginning of the volume,

and where the second Element of the Group represents the number of records to be skipped from the beginning of the positioned file.

Initialized Value: \( \text{KPOS} = (0, 0) \)

Diagnostics:

***** OPERATION NO. $ ****
I N P U T **** OPERATION NO. $ ****
MESSAGE NO. 1 FROM SUBPROGRAM INIT2 AT SECTION 12.1 EX STAGE IS 2
CONTROL 'NWVOL' LENGTH '$' INCONSISTENT WITH CONTROL 'KPOS' LENGTH '$'
FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The number of Groups of Control NWVOL must equal the number of Groups of Control KPOS.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the Controls NWVOL and KPOS to have the same number of Groups.

Dependencies: NWVOL, JCL

Comments: The names of the volumes which are to be processed are given (in order) in Control NWVOL. There must be a one-to-one correspondence between a volume names given and the starting file and record location for that volume given in Control KPOS.

Similarly the number of sets of starting file and record location information given in Control KPOS must be equal to the number of complete 'VOLUME JCL instructions (when the host machine is the CDC 7600).

2.20 KSTOP (Derived)

Purpose: Indicates when there is no more data to be input.

Form: \( \text{KSTOP} = \text{integer} \)

where 'integer' is non-negative.
Initialized Value: $KSTOP = 0$

Comments: Control $KSTOP$ is derived in the INPUT Operation. It is initialized at the beginning of execution to a value of zero (0), and is set to one (1) when there is no more data to be input.

2.21 **KTYPE**

Purpose: Declares the type of all data volumes to be input. The Control $KTYPE$ assumes that the NCAR tape utility RDTAPE is available.

Form: $KTYPE = \text{integer}$

where 'integer' is one of:

- 0 This volume has been created on the NCAR system where partial words are ignored.
- 1 Same as the zero (0) specification.
- 2 This volume has been created outside of the NCAR system where partial words are not ignored.

Initialized Value: $KTYPE = 1$

Diagnostics:

***** OPERATION NO. $**** I N P U T ***** OPERATION NO. $*****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.70 EX STAGE IS 2
CONTROL 'KTYPE' VALUE '$' INVALID; VALID RANGE '0', '2'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than zero (0) or greater than two (2).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a value for Control $KTYPE$ to be zero (0) or one (1) or two (2).

Comments: The options described apply to 7 and 9 track tape images.

2.22 **LOGBIT**

Purpose: Specifies the number of bits in a data logical record.
Form: \[ \text{LOGBIT} = \text{integer} \]

where 'integer' is positive.

Initialized Value: \[ \text{LOGBIT} = 1000 \]

Diagnostics:

**** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.80 EX STAGE IS 2
CONTROL 'LOGBIT' VALUE '$' INVALID; VALID RANGE '1', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control LOGBIT.

2.23 \text{MAXIV}

Purpose: Specifies the maximum value that the INPUT independent variable may attain.

Form: \[ \text{MAXIV} = \text{real} \]

where 'real' is any real number.

Initialized Value: \[ \text{MAXIV} = 86400 \]

2.24 \text{MEDIA (Derived)}

Purpose: Represents the media number in the header.

Form: \[ \text{MEDIA} = \text{integer} \]

where 'integer' will be the media number in the header.

Initialized Value: \[ \text{MEDIA} = 1 \]

Comments: This Control is currently non-functional.
2.25 NAMEOP (Derived)

Purpose: Assigns a unique name to this Operation, which is used in normal and diagnostic prints.

Form: NAMEOP = name

Initialized Value: NAMEOP = INPUT

Comments: The value of this Control is derived from the Programmer Directive file.

2.26 NAMKEY (Linked)

Purpose: Assigns a Dependent Variable name to the record identifier. A record identifier is defined to be a bit configuration included in every logical record which is used to differentiate the type of information contained in that record.

Form: NAMKEY = name

where 'name' is either the name of a Dependent Variable contained in the Variable Area Vector which is to be assigned to the record identifier, or in the case that no record identifier exists, is the value NOLINK.

Initialized Value: NAMKEY = NOLINK

Dependencies: BITKEY

Comments: The value of the record identifier is provided in Control BITKEY.

Control NAMKEY allows the record identifier to enter into the stream of Dependent Variables. This accordingly allows the Controls associated with Dependent Variables for the INPUT Operation to be used to further define how the record identifier is to be input. These Controls specify the location and number of bits of the record identifier.

When NAMKEY and BITKEY are being used in quality control, and when processing has terminated on the input data volume, this diagnostic is written

**** OPERATION NO. $ ****
INPUT **** OPERATION NO. $ ****
MESSAGE NO. 240 FROM SUBPROGRAM DCODE AT SECTION 16.1 EX STAGE IS 4
'$' BIT PATTERN MIS-MATCHES ENCOUNTERED ON UNIT '$'
**** INFORMATION **** INFORMATION **** INFORMATION **** INFORMATION****
2.27 NAMLIB

Purpose: Provides file name(s) which comprise all required code for this Operation if stored on some permanent storage medium (such as PLIB on the CDC 7600).

Form: NAMLIB = string1, ..., stringN

where each 'string' is a file name.

Initialized Value: NAMLIB = 'NONE'

Comments: This Control is currently nonfunctional.

2.28 NAMPD

Purpose: Provides the name of the Programmer Directive file for this Operation.

Form: NAMPD = string

Initialized Value: NAMPD = 'INPUT'

Comments: This Control is currently nonfunctional.

2.29 NAMUD

Purpose: Provides the name of the User Directive file for this module.

Form: NAMUD = string

Initialized Value: NAMUD = 'INPUT'

Comments: This Control is currently nonfunctional.

2.30 NEWV (Derived)

Purpose: Contains the number of data volumes remaining to be input.

Form: NEWV = integer

where 'integer' is non-negative.

Initialized Value: NEWV = 0

Dependencies: NWVOL
OPERATION CONTROLS

Comments: Control NEWV is derived in the INPUT Operation. It is initialized to the number of volumes to be input given in Control NWVOL minus one, and is decremented as each new volume is mounted to be input.

2.31 NOLDV (Derived)

Purpose: Contains the number of data volumes which have been input.

Form: NOLDV = integer

where 'integer' is positive.

Initialized Value: NOLDV = 0

Comments: Control NOLDV is derived in the INPUT Operation. It is initialized to zero (0), and is incremented as each volume reaches the end of data on input.

2.32 NPHY (Derived)

Purpose: The number of physical records per logical record.

Form: NPHY = integer

where 'integer' is positive.

Initialized Value: NPHY = 1 LOGBIT, DATSIZ

Comments: Control NPHY is derived in the INPUT module. IF DATOPT is equal to zero (0), then NPHY is set to one (1). Otherwise, NPHY = LOGBIT/DATSIZ.

2.33 NSEC

Purpose: Specifies the number of consecutive occurrences of the INPUT independent variable required to initiate input of data. This allows user control of where data input is initiated on a volume, forcing it to begin only where a predefined number of consecutive values of the INPUT independent variable exist.

Form: NSEC = integer

where 'integer' is non-negative and represents the number of logical records that must contain consecutive values of the INPUT independent variable beyond the first logical record containing a value of the INPUT independent variable in the range of the processor Independent
Variable.

Initialized Value: NSEC = 0

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 7 FROM SUBPROGRAM INIT2 AT SECTION 3.90 EX STAGE IS 2
CONTROL 'NSEC' VALUE '$' INVALID; VALID RANGE '0', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value is less than zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control NSEC.

Comments: The idea behind this Control is to ensure that data input is not initiated in the midst of anomalies in the INPUT independent variable, that is, where gaps or redundancies occur. After a data logical record is found to have a value of the INPUT independent variable which is within the range of the processor Independent Variable, an algorithm is initiated which looks ahead at the next few data logical records to ensure that no anomalies occur in the values of the INPUT independent variable contained in those records. The number of data logical records involved in this look-ahead algorithm is determined by NSEC. Thus no look-ahead is initiated when the value of Control NSEC is zero (0).

2.34 NTOT (Derived)

Purpose: The bit position in the physical record of the next logical record to be processed.

Form: NTOT = integer

where 'integer' is non-negative.

Initialized Value: NTOT = 0

Comments: Control NTOT is derived in the INPUT Operation. It is initialized to zero (0), and is incremented by LOGBIT as each logical record within the physical record is ready to be processed.
2.35 NUMCYC

Purpose: Defines the amount of space, in units of the number of data Cycles, which is to be allocated for this Operation's use.

Form: NUMCYC = integer

where 'integer' is one of:

>0 That number of Cycles are allocated, suspending the automatic Cycle allocation algorithm of the DRIVER for this Operation.
0 Execution of this Operation is suspended for this run.
-1 The DRIVER optimizes the number of Cycles to be allocated based on the best flow for all Operations involved in this run.
-2 Available only for select Snapshot Operations, this value instructs the DRIVER that no space is to be allocated for data Cycles for this Operation.

Initialized Value: NUMCYC = -1

Diagnostics:

**** OPERATION NO. $ ****** I N P U T ****** OPERATION NO. $ ******
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'NUMCYC' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The value of Control NUMCYC is less than a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

**** OPERATION NO. $ ****** I N P U T ****** OPERATION NO. $ ******
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***
**Operation Controls**

**Cause:** The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).

**Program Action:** (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

**Dependencies:** ITYPOP, RATE

**Comments:**

- A positive value for Control NUMCYC results in the automatic optimal allocation of space to be turned off and that number of Cycles to be allocated instead. This option should be used with care, since it suspends the optimal flow of data in the processor and may thus cause bottlenecks in the flow and increase the execution time of the entire processor. by employing a positive value, there should be good reason for its use.

- A zero (0) value for Control NUMCYC causes this Operation to be turned off. This allows an Operation to be turned off without removing the Programmer Directive and the User Directive files.

### 2.36 NUMLOG (Derived)

**Purpose:** The logical record number.

**Form:** NUMLOG = integer

where 'integer' is non-negative.

**Initialized Value:** NUMLOG = 0

### 2.37 NUMV (Derived)

**Purpose:** The number of numeric storage units required to store a complete *VOLUME JCL instruction in memory.

**This Control is a machine dependent Control used only for the CDC 7600.**
**OPERATION CONTROLS**

Form: \( \text{NUMV} = \text{integer} \)

where 'integer' is positive.

Initialized Value: \( \text{NUMV} = 1 \)

Dependencies: JCL

Comments: Like Control JCL, this Control is machine dependent, being designed only for the CDC 7600.

2.38 NWVOL

Purpose: Lists the data volume names which are to be input.

Form: \( \text{NWVOL} = \text{string1}, \text{string2}, \ldots, \text{stringN} \)

where each 'string' is a data volume name,

where \( N \) is the number of data volumes to be input,

and where 'string1' is the first volume to be input, 'string2' is the second, etc., and 'stringN' is the last.

Initialized Value: \( \text{NWVOL} = \text{EOFTAP} \)

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****

MESSAGE NO. 1 FROM SUBPROGRAM INIT2 AT SECTION 11.1 EX STAGE IS 2
CONTROL 'NEWV' LENGTH '$' INCONSISTENT WITH CONTROL 'NWVOL' LENGTH '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The number of sets of Groups terminated by a Group of value '0'
for the Control JCL is not equal to the number of Groups for the Control NWVOL.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the number of sets of Groups terminated by a Group of value '0' for Control JCL to equal the number of Groups for the Control NWVOL.

Note: This error will occur only when running on the NCAR system CDC 7600. For all other machines configurations, Control JCL is not used and thus no error diagnostics relating to it will be generated.
MESSAGE NO. 1 FROM SUBPROGRAM INIT2 AT SECTION 12.1 EX STAGE IS 2
CONTROL 'NWVOL' LENGTH '$' INCONSISTENT WITH CONTROL 'KPOS' LENGTH '$'
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The number of Groups of Control NWVOL must equal the number of
Groups of Control KPOS.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Define the Controls NWVOL and KPOS to have the same number of
Groups.

Dependencies: JCL, KPOS

Comments: This Control is not machine dependent like Controls JCL and NUMV, and
thus is required for running on all machine configurations.

2.39 OFFSET

Purpose: An offset value added to the INPUT independent variable as a correc-
tion.

Form: OFFSET = real

where 'real' may be any real number.

Initialized Value: OFFSET = 0.0

Comments: Control OFFSET allows the user to change the time zone associated
with the data being processed, or to correct time offset errors on the
input data volume.
3. VARIABLE CONTROLS

3.1 BITS

Purpose: Designates the number of bits required to define one datum of a Dependent Variable in a data logical record.

Form: \[ \text{BITS} = \text{integer} \]

where 'integer' is positive.

Initialized Value: BITS = 16

Diagnostics:

***** OPERATION NO. $ **** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 15 FROM SUBPROGRAM INIT2 AT SECTION 4.10 EX STAGE IS 2
CONTROL 'BIT$' VALUE '$' ITEM '$' INVALID; VALID RANGE '1', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control BITS.

3.2 CONKEY

Purpose: Flags the type of conversion to be performed on the datums of this Dependent Variable.

Form: \[ \text{CONKEY} = \text{integer} \]

where 'integer' is one of:

1 No conversion is to be done.

2 For the bit string of the data logical record which represents the value of a Dependent Variable in integer mode (probably written on machine of a different word length than the host machine). The most significant (high order) bit signifies a sign bit: zero (0) for a positive number, one (1) for a negative. Conversion consists of detecting the negative values and representing these in terms of the corresponding negative value on the target machine.
3 For a 16 bit string with 8-bit bytes. This CONKEY inter-
switches the bytes.

4 For a bit string containing time in hours, minutes, and seconds
generated by summing the hours information multiplied by 10000,
the minutes information multiplied by 100 and the seconds infor-
mation (HHMMSS). This CONKEY converts the original string to a
value of time in seconds.

5 Sets a Dependent Variable to the value of the INPUT independent
variable.

6 This CONKEY complements the bit string (i.e., bit string =
.NOT.(bit string)).

7 This CONKEY applies the conversion algorithm described for CON-
KEY = 6 followed immediately by the conversion algorithm
described for CONKEY = 2.

Initialized Value: CONKEY = 1

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 15 FROM SUBPROGRAM INIT2 AT SECTION 4.20 EX STAGE IS 2
CONTROL 'CONKEY' VALUE '$' ITEM '$' INVALID; VALID RANGE '1', '+'N'
***FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control CONKEY.

Comments: Conversion, if required, is always done before scaling, if any.

3.3 FACTOR

Purpose: Specifies a factor used to scale the value of the Dependent Variable.

Form: FACTOR = real

where 'real' is any value other than zero (0.).

Initialized Value: FACTOR = 1.

Diagnostics:
***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 16 FROM SUBPROGRAM INIT2 AT SECTION 4.30 EX STAGE IS 2
CONTROL 'FACTOR' VALUE '$' ITEM '$' UNREASONABLE; ASSUMING '1.0'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is equal to zero (0.).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a non-zero value for Control FACTOR.

Dependencies: SCLKEY, TERM

Comments: If the value of Control SCLKEY is one (1), that is, no scaling is to be done on the datums of the Dependent Variable, then the value of Controls FACTOR and TERM are ignored.

If the value of Control SCLKEY is two (2), then datums of the Dependent Variable are divided by the value of Control FACTOR, and from that result is subtracted the value of Control TERM.

3.4 FSTBIT

Purpose: Specifies the location in the data logical record of the leftmost bit in the bit string representing the first datum of the Dependent Variable.

Form: FSTBIT = integer

where 'integer' is a positive value representing the location, in bits, of the leftmost bit of the bit string representing the first value of the Dependent Variable, where the first bit of the data logical record is one (1).

Initialized Value: FSTBIT = 1

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 15 FROM SUBPROGRAM INIT2 AT SECTION 4.40 EX STAGE IS 2
CONTROL 'FSTBIT' VALUE '$' ITEM '$' INVALID; VALID RANGE ',', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than or equal to zero (0.).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.
Fix: Define a positive value for Control FSTBIT.

3.5 **INDGET (Derived)**

Purpose: Designates a subscript index of a Dependent Variable where data, which is the input to this Operation for this call, is obtained.

Form: \[ \text{INDGET} = \text{integer} \]

where 'integer' is greater than or equal to zero (0).

Initialized Value: \( \text{INDGET} = 0 \)

Diagnostics:

```
***** OPERATION NO.       $       I N P U T       ***** OPERATION NO.       $       *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDGET' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********
```

Cause: The Control INDGET was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDGET is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

```
***** OPERATION NO.       $       I N P U T       ***** OPERATION NO.       $       *****
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********
```

Cause: The data indices associated with Control INDGET are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.
Fix: The value assigned to Control INDGET is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indexes throughout the entire program flow. If an INDGET index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: INRATE

Comments: The value of Control INDGET is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control INRATE, and by the number of Cycles passed to this Operation in this call, are stored contiguously.

3.6 INDPUT (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the result from this Operation for this call, is stored.

Form: INDPUT = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INDPUT = 0

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDPUT' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
**FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Control INDPUT was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDPUT is not in the Operation PD, that PD is a wrong release or it has been corrupted.
If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The data indices associated with Control INDPUT are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDPUT is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indices throughout the entire program flow. If an INDPUT index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: RATE

Comments: The value of Control INDPUT is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control RATE, and by the number of Cycles available to this Operation in this call, are stored contiguously.

3.7 INRATE (Derived)

Purpose: Specifies the number of data values per Cycle to be input to this Operation for this Dependent Variable.

Form: \( \text{INRATE} = \text{integer} \)

where 'integer' is greater than or equal to zero (0).

Initialized Value: \( \text{INRATE} = 0 \)

Diagnostics:
***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INRATE' MISSING FROM CONTROL ARRAY

MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0

***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The Control INRATE was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INRATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOBMV AT SECTION 1.2 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

Program action: (1) A fatal flag is set, (2) the data move is not performed.

Fix: Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.
Dependencies: INDGET, RATE

Comments: The value assigned to Control INRATE is derived in the DRIVER from the value of Control RATE of the last Transformation Operation.

When the value of Control INRATE is zero (0), no data is to be passed to this Operation, and thus no function is performed for this Dependent Variable.

The number of data values per Cycle given in Control INRATE are stored contiguously, with the address of the first data value defined by Control INDGET.

3.8 NAMVAR (Derived)

Purpose: Assigns a name to be associated with a Dependent Variable.

Form: NAMVAR = name

Initialized Value: NAMVAR = NAMVAR

Comments: The value of Control NAMVAR is indirectly derived in the Translator by the Area Vector directives which manipulate items of the Variable Area Vector.

3.9 RATE

Purpose: Specifies the number of data values per Cycle to be stored by this Operation for this Dependent Variable.

Form: RATE = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: RATE = 1

Diagnostics:

***** OPERATION NO.  $ ***** I N P U T ***** OPERATION NO.  $ *****
MESSAGE NO. 2  FROM SUBPROGRAM LODCOM AT SECTION 4.2  EX STAGE IS 2
CONTROL 'RATE' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2  EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

If NUMCYC = -1, or positive, the RATE Control must appear in the Programmer Directive file.
Cause: The Control RATE was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control RATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ ***** IMPUT ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'
**FATAL** **FATAL** **FATAL** **FATAL** **FATAL** **FATAL**

***** OPERATION NO. $ ***** IMPUT ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOBMV AT SECTION 1.2 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'
**FATAL** **FATAL** **FATAL** **FATAL** **FATAL** **FATAL**

Cause: The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

Program action: (1) A fatal flag is set, (2) the data move is not performed.

Fix: Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.

Dependencies: INRATE, INDPUT

***** OPERATION NO. $ ***** IMPUT ***** OPERATION NO. $ *****
MESSAGE NO. 15 FROM SUBPROGRAM INIT2 AT SECTION 4.50 EX STAGE IS 2
CONTROL 'RATE' VALUE '$' ITEM '$' INVALID; VALID RANGE '1', '+N'
**FATAL** **FATAL** **FATAL** **FATAL** **FATAL** **FATAL**

Cause: The value is less than zero (0).

Release 1.0

May 1, 1983
Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a non-negative value for Control RATE.

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ ***** MESSAGE NO. 18 FROM SUBPROGRAM INIT2 AT SECTION 15.1 EX STAGE IS 2 CNTRL 'SAMPLE' VALU '$' ITEM '$' INCNSISTNT WITH CNTRL 'RATE' VALU '$' ITEM '$' ***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The value of Control SAMPLE, when multiplied by the value of Controls DELIV (see DRIVER Controls document) devided by FRTIM, must exactly equal the value of Control RATE. There is some incompatibility among these Controls.

Program action: The value of Control RATE is set to (DELIV/FRTIM)*SAMPLE.

Fix: Define the Controls DELIV, FRTIM, SAMPLE, and RATE in a way to insure that the above mentioned relationships are honored.

Dependencies: SAMPLE, FRTIM, DELIV (DRIVER Control), INRATE, INDPUT

Comments: When the value of Control RATE is zero (0), no data is to be stored by this Operation, and thus no function is performed for this Dependent Variable.

The number of data values per Cycle given in Control RATE are stored contiguously, with the address of the first data value defined by Control INDPUT.

The value of Control SAMPLE, when multiplied by the value of Controls DELIV (see DRIVER Controls document) devided by FRTIM, must exactly equal the value of Control RATE.

3.10 SAMPLE

Purpose: Specifies the number of values per data logical record of a Dependent Variable.

Form: SAMPLE = integer

where 'integer' is positive.

Initialized Value: SAMPLE = 1

Diagnostics:
OPERATION CONTROLS

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 15 FROM SUBPROGRAM INIT2 AT SECTION 4.60 EX STAGE IS 2
CONTROL 'SAMPLE' VALUE '$' ITEM '$' INVALID; VALID RANGE,'1', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control SAMPLE.

Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 18 FROM SUBPROGRAM INIT2 AT SECTION 15.1 EX STAGE IS 2
CNTRL 'SAMPLE' VALU '$' ITEM '$' INCNSISTNT WITH CNTRL 'RATE' VALU '$' ITEM '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control SAMPLE, when multiplied by the value of Controls DELIV (see DRIVER Controls document) devided by FRTIM, must exactly equal the value of Control RATE. There is some incompatibility among these Controls.

Program action: The value of Control RATE is set to (DELIV/FRTIM)*SAMPLE.

Fix: Define the Controls DELIV, FRTIM, SAMPLE , and RATE in a way to insure that the above mentioned relationships are honored.

Dependencies: RATE, FRTIM, DELIV (DRIVER Control)

Comments: The value of SAMPLE*(DELIV/FRTIM) must equal the value of Control RATE.

3.11 SCLKEY

Purpose: Flags the type of scaling to be performed on the values of this Dependent Variable. Scaling consists of algebraically modifying a data value after that value has gone through the conversion process, if any.

Form: SCLKEY = integer

where 'integer' is one of:

1 No scaling is to be performed (i.e., the bit string is already in floating point mode).

Release 1.0 May 1, 1983
Each data value is divided by a specified factor and from this quotient is subtracted a specified term.

Initialized Value: SCLKEY = 2

Diagnostics:

**** OPERATION NO. $ **** I N P U T **** OPERATION NO. $ ****
MESSAGE NO. 15 FROM SUBPROGRAM INIT2 AT SECTION 4.70 EX STAGE IS 2
CONTROL 'SCLKEY' VALUE '$' ITEM '$' INVALID; VALID RANGE '1', '2'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value less than one (1) or greater than two (2).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a value of one (1) or two (2) for Control SCLKEY.

Dependencies: FACTOR, TERM

Comments: Scaling, if required, is always done after conversion, if any.

If the value of SCLKEY is one (1), then Controls FACTOR and TERM are ignored.

NOTE: Floating point information on the input data volume is strongly discouraged, since floating point information is represented in a variety of ways from machine to machine.

If the value of SCLKEY is two (2), then the value of Control FACTOR is the divisor, and the value of Control TERM is the term which is subtracted from the quotient (i.e., unscaled value = \(\text{scaled input value} / \text{FACTOR} - \text{TERM} \)).

3.12 SKIP

Purpose: The number of bits between the rightmost bit of the previous datum of a bit string representing a value of this Dependent Variable and the leftmost bit of the next datum of a bit string representing a value of this Dependent Variable.

Form: SKIP = integer

where 'integer' is a non-negative number.

Initialized Value: SKIP = 0
Diagnostics:

***** OPERATION NO. $ ***** I N P U T ***** OPERATION NO. $ *****
MESSAGE NO. 15 FROM SUBPROGRAM INIT2 AT SECTION 4.80 EX STAGE IS 2
CONTROL 'SKIP' VALUE '$' ITEM '$' INVALID; VALID RANGE '1', '+N'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value is less than or equal to zero (0).

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Define a positive value for Control SKIP.

Dependencies: SAMPLE, BITS, FSTBIT

3.13 TERM

Purpose: Specifies a term used to scale the value of the Dependent Variable.

Form: \[ \text{TERM} = \text{real} \]

Initialized Value: TERM = 0.0

Dependencies: SCLKEY, TERM

Comments: If the value of Control SCLKEY is one (1), that is, no scaling is to be done for values of the Dependent Variable, then the value of Controls FACTOR and TERM are ignored.

If the value of Control SCLKEY is two (2), then values of the dependent variable are divided by the value of Control FACTOR, and from that result is subtracted the value of Control TERM.

3.14 TITLE

Purpose: Provides a textual description or title for a Dependent Variable.

Form: \[ \text{TITLE} = \text{string} \]

where 'string' may be up to forty (40) characters in length, where all characters are from the set of legal characters available on the target machine.

Initialized Value: TITLE = 'that is, forty (40) blank characters.'
3.15 **UNITS**

**Purpose:** Defines the units of measurement of a Dependent Variable.

**Form:** 

\[ \text{UNITS} = \text{string} \]

where 'string' may be up to eight (8) characters in length, where all characters are from the set of legal characters available on the target machine.

**Initialized Value:** \[ \text{UNITS} = ' ' \]

that is, eight (8) blank characters.
GENPRO STATS Operation
REFERENCE MANUAL

Editors: Bonnie Gacnik
Bob Lackman

GENeral Scientific Data PROcessor

SCIENTIFIC COMPUTING DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO
1. INTRODUCTION .................................... 3-STATS.1

2. GENERAL CONTROLS .................................... 3-STATS.2
   2.1 BEGSPN ........................................... 3-STATS.2
   2.2 ENDSNP ........................................... 3-STATS.6
   2.3 FLUSHP ........................................... 3-STATS.10
   2.4 ITYPOP (Derived) ................................ 3-STATS.11
   2.5 KOUT ............................................ 3-STATS.13
   2.6 LOUT ............................................ 3-STATS.14
   2.7 MAXCHR .......................................... 3-STATS.17
   2.8 NAMEOP (Derived) ................................ 3-STATS.17
   2.9 NAMLIB .......................................... 3-STATS.17
   2.10 NAMPD .......................................... 3-STATS.17
   2.11 NUMCYC ......................................... 3-STATS.18
   2.12 PROJECT ......................................... 3-STATS.19

3. VARIABLE CONTROLS ..................................... 3-STATS.20
   3.1 DELSIG (Derived) ................................ 3-STATS.20
   3.2 DLMAX (Derived) ................................ 3-STATS.20
   3.3 DLMEN (Derived) ................................ 3-STATS.20
   3.4 DLMIN (Derived) ................................ 3-STATS.21
   3.5 INDGET (Derived) ................................ 3-STATS.21
   3.6 INPUT (Derived) .................................. 3-STATS.22
   3.7 INRATE (Derived) ................................ 3-STATS.24
   3.8 KURTOSIS (Derived) ................................ 3-STATS.25
   3.9 NAMVAR (Derived) ................................ 3-STATS.26
   3.10 NDEL (Derived) .................................. 3-STATS.26
   3.11 NPTOT (Derived) ................................ 3-STATS.26
   3.12 NUMGAP (Derived) ................................ 3-STATS.27
   3.13 PTSAV (Derived) ................................ 3-STATS.27
   3.14 RATE ............................................ 3-STATS.27
   3.15 SIGMA (Derived) .................................. 3-STATS.29
   3.16 SKEW (Derived) ................................... 3-STATS.29
   3.17 TMSAV (Derived) ................................... 3-STATS.30
   3.18 UNITS ............................................. 3-STATS.30
   3.19 XMAX (Derived) ................................... 3-STATS.30
   3.20 XMEAN (Derived) ................................... 3-STATS.31
   3.21 XMIN (Derived) ................................... 3-STATS.31
1. INTRODUCTION

The purpose of the STATS Operation is to calculate and display a variety of statistical information regarding the data values of the Dependent Variables of this Operation. In addition to the Dependent Variable symbolic names, units, and rates, any or all of the following statistics may be selected for display:

- Number of points used in the calculations
- Maximum value
- Minimum value
- Mean value
- Sigma (n times the sum of the squares then subtract the sum squared and divide the result by n) **1/2
- Skew (n squared times sum of the cubes then subtract the sum cubed and divide by n cubed then divide by sigma cubed)
- Kurtosis (n cubed times sum of the quadruples then subtract the sum quadrupled and divide by n quadrupled then divide by sigma to the fourth power)
- Maximum delta between adjacent points
- Minimum delta between adjacent points
- Mean delta
- Sigma delta (see sigma)
- Number of wild or missing Data values which have been replaced by the value of DRIVER Control GAPVAL. These replacements, if any, would be done by other GEN-PRO Operations such as the INPUT Operation.
2. GENERAL CONTROLS

2.1 BEGSNP

Purpose: Marks the beginning of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: BEGSNP = (real1,...,realN),...,(real1,...,realN)

where the number of Elements in each Group, N, is defined by Control IVDIM,

and where each Element represents a component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then, BEGSNP = (3., 51., 0.) would cause a Snapshot Period to begin at hour 3, minute 51, and second 0.

Initialized Value: BEGSNP = (0.0, 0.0, 0.0)

This value represents the beginning of one Snapshot of an Independent Variable with three components. For a positive valued Independent Variable it also represents a Snapshot which will begin at the first Cycle of the data for any Case.

Diagnostics:

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'BEGSNP' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Group length of Control BEGSNP is not equal to the number of dimensioned components of the Independent Variable, $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM.
***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
FATAL**********FATAL**********FATAL**********FATAL**********FATAL********

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP,
the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for BEGSNP and ENDSNP.

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 10 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH 'ENDSNP' VALUE '$'; ASSUMING '$'
WARNING**********WARNING**********WARNING**********WARNING**********WARNING

Cause: The value of Control BEGSNP is less than the ENDSNP of the pre-
vious Snapshot Period.

Program action: A warning message is printed.

Fix: Respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
INCONSISTENT SNAPSHOT INTERVALS '$' THROUGH '$' HAVE BEEN TURNED OFF.
FATAL**********FATAL**********FATAL**********FATAL**********FATAL********

Cause: A previous fatal error was detected in either Control BEGSNP or
ENDSNP.

Program action: All Snapshot Periods starting with the first one to
produce a fatal error condition are deactivated.

Fix: Respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAP INTERVAL 'I' 'BEGSNP' = '$' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE.
DID YOU WANT TO SKIP A CYCLE?
WARNING**********WARNING**********WARNING**********WARNING**********WARNING

Cause: The value of Control BEGSNP is one Cycle larger than the ENDSNP
of the previous Snapshot Period.
Program action: A warning message is printed.

Fix: If the intent is to display all Cycles Respecify the BEGSNP of interval I equal to the ENDSNP of interval I-1.

***** OPERATION NO. $ ***** STATS ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
MESSAGE NO. 61 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, (2) the Operation in question is removed from processing.

Fix: Since these intervals were previously examined in DRIVER routine LODCOM, the code of the Operation in question might have modified them, or random data stores might have overwritten them. See DRIVER Controls DEBUG and DUMP for debug aid.

Multiple Snapshot Periods (Groups) can be defined on the total run interval from BEGIV to ENDIV; however, these intervals cannot overlap although the start of the current interval should repeat the end of the last interval for continuous processing. For example,
BEGIV = (12.,20.,00.) $ ENDIV = (14.,24.,00.)

Over this run interval a legal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,30.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(14.,20.,00.)</td>
</tr>
</tbody>
</table>

Note that the intervals from 13.,30.,00. to 14.,00.,00. and 14.,20.,00. to 14.,24.,00. will not generate a display.

Also note that the first Snapshot Period is less than BEGIV for this Case. The DRIVER will reset it to BEGIV. If an ENDSNP value extends beyond ENDIV, a similar truncation would occur.

Finally note that the start of interval two equals the end of interval one. This insures continuous output. A warning is issued if the start of a subsequent interval is exactly one time step (see DRIVER Control DELIV) larger than the ending of a previous interval.

Over this run interval an illegal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,20.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(13.,20.,00.)</td>
</tr>
</tbody>
</table>

The second interval above is illegal because the BEGSNP of (12.,20.,00.) is less than the ENDSNP of interval one.

Also, the third interval above is illegal because the BEGSNP of (14.,00.,00.) is larger than the ENDSNP of (13.,20.,00.).

NOTE: These are fatal conditions and no processing of a display period with a fatal error nor any subsequent intervals are done even if the processor has been instructed to continue in spite of fatal errors (see DRIVER Control IABORT, value 5).

Dependencies: ENDSNP, IVDIM (see DRIVER Controls Document)

For Control BEGSNP, the number of Elements in a Group is defined by the number of components of the Independent Variable, given by DRIVER Control IVDIM.

Controls BEGSNP and ENDSNP work as ordered pairs describing the beginning and the ending of a Snapshot Period, that is, the Mth Group of Control BEGSNP marks the beginning of the Mth Snapshot Period, and the Mth Group of Control ENDSNP marks the ending of that same period. Therefore, these two Controls must have the same number of Groups.
In addition, Snapshot Periods may not overlap. Therefore, the next Group of Control BEGSNP must specify an Independent value greater than that of the previous Group of Control ENDSNP.

Comments: The number of Elements in a Group for Control BEGSNP is initially defined in the Programmer Directive file for this Operation to be three (3).

The Snapshot Periods must be an increasing function of the Independent Variable. Therefore, the Groups of Control BEGSNP must be arranged in increasing order.

All Snapshot Periods must lie within the interval of the Independent Variable delimited by Controls BEGIV and ENDIV.

2.2 ENDSNP

Purpose: Marks the ending of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: ENDSNP = (real1,...,realN),..., (real1,...,realN)

where the number of Elements in each Group, N, is defined by Control IVDIM,

and where each Element represents a component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then,

ENDSNP = (17., 6., 30.)

would cause a Snapshot Period to end at hour 17, minute 6, and second 30.

Initialized Value: ENDSNP = (99., 99., 99.)

This value represents the ending of one Snapshot of an Independent Variable with three components.

Diagnostics:

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
"CONTROL 'ENDSNP' LENGTH '$_' IS INVALID. CHANGE LENGTH TO '$_' IN PD FILE"
"*FATAL*********FATAL*********FATAL*********FATAL*********FATAL*********FATAL*********FATAL*********

May 1, 1983
Release 1.0
**Cause:** The Group length of Control ENDSNP is not equal to the number of dimensioned components of the Independent Variable, $IVDIM.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM of the associated code release.

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****

MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
***FATAL*******FATAL**********FATAL**********FATAL**********FATAL*****

**Cause:** The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP.

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****

MESSAGE NO. 70 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAPSHOT INTERVAL '$' 'ENDSNP' = '$' EXCEEDS 'ENDSNP' ='$' OF PRIOR INTERVAL
***FATAL**********FATAL**********FATAL**********FATAL**********FATAL*****

**Cause:** The value of Control ENDSNP is less than the ENDSNP of the previous Snapshot Period.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Respecify consistent values for the Snapshot Periods.

***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****

MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
INCONSISTENT SNAPSHOT INTERVALS '$' THROUGH '$' HAVE BEEN TURNED OFF.
***FATAL**********FATAL**********FATAL**********FATAL**********FATAL*****

**Cause:** A previous fatal error was detected in either Control BEGSNP or ENDSNP.

**Program action:** All Snapshot Periods starting with the first one to produce a fatal error condition are deactivated.

**Fix:** Respecify consistent values for BEGSNP and ENDSNP intervals.

---

**Release 1.0 May 1, 1983**
**** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAP INTERVAL 'I' 'BEGSNP' = '$' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE.
DID YOU WANT TO SKIP A CYCLE?
****WARNING********WARNING********WARNING********WARNING********WARNING****

Cause: The value of Control BEGSNP is one Cycle larger than the ENDSNP of the previous Snapshot Period.

Program action: A warning message is printed.

Fix: If the intent is to display all Cycles respecify the BEGSNP of interval I equal to the ENDSNP of interval I-1.

**** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
MESSAGE NO. 61 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL*********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, (2) the Operation in question is removed from processing.

Fix: Since these intervals were previously examined in DRIVER routine LODCOM, the code of the Operation in question might have modified them, or random data stores might have overwritten them. See DRIVER Controls DEBUG and DUMP for debug aid.

Multiple Snapshot Periods (Groups) can be defined on the total run interval from BEGIV to ENDIV; however, these intervals cannot overlap, but the start of the current interval should repeat the end of the last interval for continuous processing. For example,
BEGIV = (12.,20.,00.) $ ENDIV = (14.,24.,00.)

Over this run interval a legal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>Interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(12.,00.,00.)</td>
<td>(12.,30.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(99.,99.,99.)</td>
</tr>
</tbody>
</table>

Note that the intervals from 13.,30.,00. to 14.,00.,00. and 14.,20.,00. to 14.,24.,00. will not generate a display.

Also note that the last Snapshot Period exceeds ENDIV for this Case. The DRIVER will reset it to ENDIV. A similar truncation of a BEGSNP that starts before BEGIV would occur.

Finally note that the start of interval two equals the end of interval one. This insures continuous output. A warning is issued if the start of a subsequent interval is exactly one time step (see DRIVER Control DELIV) larger than the ending of a previous interval.

Over this run interval an illegal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>Interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,20.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(13.,20.,00.)</td>
</tr>
</tbody>
</table>

The second interval above is illegal because the BEGSNP of (12.,20.,00.) is less than the ENDSNP of interval one.

Also, the third interval above is illegal because the BEGSNP of (14.,00.,00.) is larger than the ENDSNP of (13.,20.,00.).

NOTE: These are fatal conditions and no processing of a display period with a fatal error nor any subsequent intervals is done even if the processor has been instructed to continue in spite of fatal errors (see DRIVER Control IABORT, value 5).

Dependencies: BEGSNP, IVDIM (see DRIVER Controls Document)

For Control ENDSNP, the number of Elements in a Group is defined by DRIVER Control IVDIM.

Controls BEGSNP and ENDSNP work as ordered pairs describing the beginning and the ending of a Snapshot Period, that is, the Mth Group of Control BEGSNP marks the beginning of the Mth Snapshot Period, and the Mth Group of Control ENDSNP marks the ending of that same period. Therefore, these two Controls must have the same number of Groups.

In addition, Snapshot Periods may not overlap. This implies that the
next Group of Control BEGSNP must specify an Independent Variable value
greater than that of the previous Group of Control ENDSNP.

Comments: The number of Elements in a Group for Control ENDSNP is initially
defined in the Programmer Directive file for this Operation to be three (3).

The Snapshot Periods must be an increasing function of the Independent
Variable. This implies that the Groups of the Control ENDSNP must be
arranged in increasing order.

All Snapshot Periods must lie within the interval of the Independent
Variable delimited by DRIVER Controls BEGIV and ENDIV. NOTE: The last
Cycle to be displayed is one Cycle BEFORE
the value given in Control ENDSNP.

2.3 FLUSHP

Purpose: Sets the length of a Flush Period.

A Flush Period is a sub-interval of the Snapshot Period marking the
frequency of a specific action. At the end of the specific action, all
data within the Flush Period is discarded, or "flushed", and execution
on subsequent data within the Snapshot Period is continued.

Form: FLUSHP = integer

where 'integer' is the number of Cycles in the Flush Period.

Initialized Value: FLUSHP = 900

For data with an Independent Variable of Time with each Cycle
representing one second in time, this number represents fifteen minutes
of data.

Diagnostics:

**** OPERATION NO. $ ***** STATS ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'FLUSHP' MISSING FROM CONTROL ARRAY
***FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****

Cause: Control FLUSHP is missing from the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution con-
tinues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number.
PD release numbers must match the code release numbers. Then verify
that the ordering of the PDs and UDs is correct in the set of Directive files. If Control FLUSHP is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ **** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 6 FROM SUBPROGRAM LODCOM AT SECTION 3.5 EX STAGE IS 2
CONTROL 'FLUSHP' VALUE '$' INVALID; VALID RANGE '$' TO '$'; ASSUMING '$'

Cause: The value of Control FLUSHP is less than or equal to zero (0).

Program action: (1) A warning message is printed, and (2) the interval for displays, FLUSHP, is reset to the full run interval of the Case.

Fix: If specific intervals for display are wanted, respecify and correlate Controls, BEGSNP, ENDSNP, and FLUSHP.

Dependencies: BEGSNP, ENDSNP, DELIV (see DRIVER Controls Document)

Controls BEGSNP and ENDSNP define the Snapshot Periods of which the Flush Period is a sub-interval.

Comments: The Flush Period begins with the first Cycle of data in the Snapshot Period. A counter is employed to count the number of Cycles from the beginning of the Flush Period to the length of the Flush Period defined in Control FLUSHP or until the end of the Snapshot Period. If the number of Cycles remaining in a Snapshot Period is greater than the Flush Period, then a "flush" is performed, the count of Cycles in a Flush Period is reset to zero (0), and a new Flush Period is begun. If the number of Cycles remaining in the Snapshot Period is less than the Flush Period, then a "flush" is performed on an abbreviated Flush Period, marked by the end of the Snapshot Period.

2.4 ITYPOP (Derived)

Purpose: Specifies the type of Operation.

Form: ITYPOP = integer

where 'integer' is one of:

-1 In-Place Transformation Operation
0 Snapshot Operation
1 Input Operation

2 Transformation Operation

Initialized Value: ITYPOP = 0

Diagnostics:

***** OPERATION NO. $ *****  STATS ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 3.0  EX STAGE IS 2
CONTROL 'ITYPOP' MISSING FROM CONTROL ARRAY
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The Control ITYPOP was not found in the PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UD sets is correct in the set of Directive files. If Control ITYPOP is not in the PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ *****  STATS ***** OPERATION NO. $ *****
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4  EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4  EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The value of Control ITYPOP is less than a minus one (-1) or greater than two (2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

May 1, 1983

3-STATS.12 REFERENCE MANUAL
Release 1.0
***** OPERATION NO. $ ***** S T A T S ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies: NUMCYC

Comments: The value of this Control is derived from the Programmer Directive file.

2.5 KOUT

Purpose: Designates the logical names of the devices on which the statistics generated by this Operation are output.

Form: KOUT = name1, ..., nameN

where N is the number of devices on which the statistics are to be output, and is greater than, or equal to zero (0),

where 'name1' through 'nameN' must be one of:

KPROC The generated statistics will be written to the unit specified by the DRIVER Control KPROC

KPRINT The generated statistics will be written to the unit specified by the DRIVER Control KPRINT

KFILM The generated statistics will be written to the unit specified by the DRIVER Control KFILM
**KFICHE** The generated statistics will be written to the unit specified by the DRIVER Control KFICHE.

**NONE** This value is used as an optional terminator to the list of logical names of devices which, when present, will cause all values given after it to be ignored.

**Initialized Value: KOUT = KPRINT**

**Diagnostics:**

```
***** OPERATION NO. $ ***** STATS ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM STATS AT SECTION 3.10 EX STAGE IS 5
CONTROL 'KOUT' MISSING FROM CONTROL ARRAY
***FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****FATAL*****
```

**Cause:** Control KOUT missing from the Programmer Directive file.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Control KOUT must be in Programmer Directive file.

**Dependencies:** KPROC, KPRINT, KFILM, KFICHE (see DRIVER Controls document)

**Comments:** Control KOUT allows the statistics generated in the STATS Operation to be written out to any of a number of logical units. Even multiple copies may be sent to the same device by specifying that logical unit name more than once.

The assignments of the logical unit names to a logical unit number is made in the DRIVER Controls listed above. The meanings ascribed to each name is also given in the documentation for those Controls.

### 2.6 **LOUT**

**Purpose:** Enumerates a set of Controls for which values of every Dependent Variable will be printed in columnar form in the output generated by the STATS Operation.

**Form:**

```
LOUT = name1, ..., nameN
```

where N is the number of Controls which are to head columns in the printed output,

and where each 'name' is one of:

**STOP** This value is used as an optional terminator to the list of names of Controls which, when present, will cause all
values given after it to be ignored

DELSIG The value delta sigma for each Dependent Variable will be printed

DLMAX The value of the maximum delta for each Dependent Variable will be printed

DLMIN The value of the minimum delta for each Dependent Variable will be printed

DLMEN The value of the mean delta for each Dependent Variable will be printed

FDEL The value of the flag which determines whether gaps in the data (marked by the value given by DRIVER Control GAPVAL) are to be included in statistical calculations for each Dependent Variable will be printed

NDEL The total number of delta values involved in delta calculations for each Dependent Variable will be printed

NPTOT The total number of data values involved in computing statistics for each Dependent Variable will be printed

NUMGAP The number of gaps in the data (marked by the value given by DRIVER Control GAPVAL) for each Dependent Variable will be printed

PTSAV The last data value from the last Cycle in the last call to the STATS Operation for each Dependent Variable will be printed

RATE The number of data values per Cycle to be used in computing statistics for each Dependent Variable will be printed

SIGMA The value of sigma for each Dependent Variable will be printed

SKEW The value of the skew for each Dependent Variable will be printed

UNITS The units of measurement for each Dependent Variable will be printed

KURTOSIS The value of the kurtosis for each Dependent Variable will be printed

XMAX The maximum value for each Dependent Variable will be printed
XMEAN  The mean value for each Dependent Variable will be printed

XMIN  The minimum value for each Dependent Variable will be printed

Initialized Value: LOUT = RATE, UNITS, NPTOT, XMAX, XMIN, XMEAN, SIGMA, SKEW, KURTOSIS, NDEL DLMAX, DLMIN, DLMEN, DELSIG, NUMGAP, STOP

Diagnostics:

***** OPERATION NO. $ *****   S T A T S   ***** OPERATION NO. $ *****

MESSAGE NO. 2  FROM SUBPROGRAM STATS AT SECTION 3.10  EX STAGE IS 5
CONTROL 'LOUT' MISSING FROM CONTROL ARRAY

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: Control LOUT missing from the Programmer Directive file.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Control LOUT must be in Programmer Directive file.

***** OPERATION NO. $ *****   S T A T S   ***** OPERATION NO. $ *****

MESSAGE NO. 4  FROM SUBPROGRAM INSTAT AT SECTION 0.50  EX STAGE IS 2
CONTROL 'LOUT' VALUE '$' NOT IMPLEMENTED

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The list of Controls specified in LOUT contains a Control name which is not an existing Dependent Variable Control. This usually means a Control name has been mis-spelled.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the LOUT list and verify that all Control names found in this list are valid for this Operation.

The initialized value causes the most useful statistics to be printed.

Comments: The name of each Dependent Variable, given by Control NAMVAR, will automatically appear as the first column in all printed output, and thus, does not need to be specified as a value of the LOUT Control.

The order of values for Control LOUT is not significant. The order that the columns of values for Controls will take is dependent on the order that Controls appear in the Control Array.

The values given to Control LOUT must match exactly the desired Control given in the Variable Controls section of the Control Array for the STATS Operation.
2.7 MAXCHR

Purpose: Specifies the maximum number of characters which can be printed on one line of the output device to which tabular output is to be written.

Form: \( \text{MAXCHR} = \text{integer} \)

where 'integer' is positive and is dependent on the physical limitation of the output device.

Initialized Value: \( \text{MAXCHR} = 128 \)

Comments: This Control is useful for statistical and tabular displays. Standard output to a paper printer might use 128 characters per line while terminal output might be restricted to 78 characters per line.

2.8 NAMEOP (Derived)

Purpose: Assigns a unique name to this Operation, which is used in informational and diagnostic prints.

Form: \( \text{NAMEOP} = \text{name} \)

Initialized Value: \( \text{NAMEOP} = \text{STATS} \)

Comments: The value of this Control is derived from the Programmer Directive file.

2.9 NAMLIB

Purpose: Provides file name(s) which comprise all required code for this Operation if stored on some permanent storage medium (such as PLIB on the CDC 7600).

Form: \( \text{NAMLIB} = \text{string}_1, \ldots, \text{string}_N \)

where each 'string' is a file name.

Initialized Value: \( \text{NAMLIB} = \text{'NONE'} \)

Comments: This Control is currently nonfunctional.

2.10 NAMPD

Purpose: Provides the name of the Programmer Directive file for this Operation.
Form:  NAMPD = string

Initialized Value:  NAMPD = 'NONE'

Comments:  This Control is currently nonfunctional.

2.11 NUMCYC

Purpose:  Defines the amount of space, in units of the number of data Cycles, which is to be allocated for this Operation's use.

Form:  NUMCYC = integer

where 'integer' is one of:

>0  That number of Cycles are allocated, suspending the automatic Cycle allocation algorithm of the DRIVER for this Operation.

0  Execution of this Operation is suspended for this run.

-1  The DRIVER optimizes the number of Cycles to be allocated based on the best flow for all Operations involved in this run.

-2  Available only for select Snapshot Operations, this value instructs the DRIVER that no space is to be allocated for data Cycles for this Operation.

Initialized Value:  NUMCYC = -2

Diagnostics:

***** OPERATION NO.  $ *****  S T A T S  ***** OPERATION NO.  $ *****
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4  EX STAGE IS 2  CONTROL 'NUMCYC' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4  EX STAGE IS 2  OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause:  The value of Control NUMCYC is less than a minus two (-2).

Program action:  (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix:  Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.
***** OPERATION NO. $ *****

MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies: ITYPOP, RATE

Comments: A positive value for Control NUMCYC results in the automatic optimal allocation of space to be turned off and that number of Cycles to be allocated instead. This option should be used with care, since it suspends the optimal flow of data in the processor and may thus cause bottlenecks in the flow and increase the execution time of the entire processor. by employing a positive value, there should be good reason for its use.

A zero (0) value for Control NUMCYC causes this Operation to be turned off. This allows an Operation to be turned off without removing the Programmer Directive and the User Directive files.

2.12 PROJECT

Purpose: Provides a textual description or title of the project, which is used to label the various output generated by this Operation.

Form: PROJECT = string

where 'string' may be up to forty (40) characters in length.

Initialized Value: PROJECT = '

that is, forty (40) blank characters.
3. VARIABLE CONTROLS

3.1 DELSIG (Derived)

Purpose: Contains the statistic "delta sigma" for a Dependent Variable for a Flush Period.

Form:  DELSIG = real

Initialized Value: DELSIG = 0.0

Comments: Control DELSIG is derived in the STATS Operation. It is initialized to zero (0) at the beginning of each Flush Period, contains the running sum of delta squared points up to the Flush Period boundary, at which time the value of delta sigma is computed and put into it.

3.2 DLMAX (Derived)

Purpose: Contains the statistic "delta maximum" for a Dependent Variable for a Flush Period.

Form:  DLMAX = real

Initialized Value: DLMAX = -9.E32

Comments: Control DLMAX is derived in the STATS Operation. It is initialized to a very large negative number at the beginning of each Flush Period, and is updated to the largest delta found to date.

3.3 DLMEN (Derived)

Purpose: Contains the statistic "mean delta" for a Dependent Variable for a Flush Period.

Form:  DLMEN = real

Initialized Value: DLMEN = 0.0

Comments: Control DLMEN is derived in the STATS Operation. It is initialized to zero (0.0) at the beginning of each Flush Period, and is updated to the mean delta for the Flush Period.
3.4 **DLMIN (Derived)**

Purpose: Contains the statistic "delta minimum" for a Dependent Variable for a Flush Period.

Form: \( \text{DLMIN} = \text{real} \)

Initialized Value: \( \text{DLMIN} = 9.E32 \)

Comments: Control DLMIN is derived in the STATS Operation. It is initialized to a very large positive number at the beginning of each Flush Period, and is updated to the smallest delta found to date.

3.5 **INDGET (Derived)**

Purpose: Designates a subscript index of a Dependent Variable where data, which is the input to this Operation for this call, is obtained.

Form: \( \text{INDGET} = \text{integer} \)

where 'integer' is greater than or equal to zero (0).

Initialized Value: \( \text{INDGET} = 0 \)

Diagnostics:

**** OPERATION NO. $ **** S T A T S **** OPERATION NO. $ ****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDGET' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL*******FATAL*******FATAL*******FATAL*******FATAL*******FATAL***

Cause: The Control INDGET was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDGET is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY

****FATAL*******FATAL********FATAL********FATAL********FATAL*******FATAL***

Cause: The data indices associated with Control INDGET are out of range
of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDGET is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indecies throughout the entire program flow. If an INDGET index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: INRATE

Comments: The value of Control INDGET is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control INRATE, and by the number of Cycles passed to this Operation in this call, are stored contiguously.

3.6 INDPUT (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the result from this Operation for this call, is stored.

Form: INDPUT = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INDPUT = 0

Diagnostics:

MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDPUT' MISSING FROM CONTROL ARRAY
OPERATION '$' TERMINATED; NUMCYC SET TO 0

****FATAL*******FATAL********FATAL********FATAL********FATAL*******FATAL***

May 1, 1983
Cause: The Control INDPUT was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDPUT is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

****** OPERATION NO. $ *****  S T A T S  ***** OPERATION NO. $ *****
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The data indices associated with Control INDPUT are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDPUT is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indices throughout the entire program flow. If an INDPUT index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Dependencies: RATE

Comments: The value of Control INDPUT is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control RATE, and by the number of Cycles available to this Operation in this call, are stored contiguously.
3.7 INRATE (Derived)

Purpose: Specifies the number of data values per Cycle to be input to this Operation for this Dependent Variable.

Form: \[ \text{INRATE} = \text{integer} \]

where 'integer' is greater than or equal to zero (0).

Initialized Value: \[ \text{INRATE} = 0 \]

Diagnostics:

***** OPERATION NO. $ *****  S T A T S  ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2  EX STAGE IS 2
CONTROL 'INRATE' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2  EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Control INRATE was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INRATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ *****  S T A T S  ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5  EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

***** OPERATION NO. $ *****  S T A T S  ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOBMV AT SECTION 1.2  EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

Program action: (1) A fatal flag is set, (2) the data move is not performed.
Fix: Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.

Dependencies: INDGET, RATE

Comments: The value assigned to Control INRATE is derived in the INITIO routine of the DRIVER from the value of Control RATE of the last Transformation Operation.

When the value of Control INRATE is zero (0), no data is to be passed to this Operation, and thus no function is performed for this Dependent Variable.

The number of data values per Cycle given in Control INRATE are stored contiguously, with the address of the first data value defined by Control INDGET.

3.8 KURTOSIS (Derived)

Purpose: Contains the statistic "kurtosis" for a Dependent Variable for a Flush Period.

Form: KURTOSIS = real

Initialized Value: KURTOSIS = 0.0

Diagnostics:

**** OPERATION NO.  4  *****  S T A T S  ***** OPERATION NO.  4  *****
MESSAGE NO. 205 FROM SUBPROGRAM STATS  AT SECTION  3.5  EX STAGE IS  5
CALCULATIONS WERE NOT COMPLETED ON SOME VERY LARGE OR VERY SMALL VALUES
**** WARNING  ******** WARNING  ******** WARNING  ******** WARNING  ****

Cause: A machine underflow or overflow would occur if statistical computations were to continue on this Dependent Variable.

Program action: (1) A warning message is printed, and (2) GAPVAL is substituted for the value of KURTOSIS.

Fix: Generally this message can be avoided by reducing the number of Cycles in a normal Flush Period defined by Control FLUSHP.
Comments: Control KURTOSIS is derived in the STATS Operation. It is initialized to zero (0.0) at the beginning of a Flush Period, and thereafter is the sum of the fourth power of the data values. At the end of the Flush Period, a final computation produces the statistical value "kurtosis".

A resultant value of -99. means that the kurtosis for this Dependent Variable for this Flush Period could not be determined.

3.9 NAMVAR (Derived)

Purpose: Assigns a name to be associated with a Dependent Variable.

Form: NAMVAR = name

Initialized Value: NAMVAR = NAMVAR

Comments: The value of Control NAMVAR is indirectly derived in the Translator by the Area Vector directives which manipulate items of the Variable Area Vector.

3.10 NDEL (Derived)

Purpose: Contains a count of the number of point to point differences, called deltas, for a Dependent Variable for a Flush Period.

Form: NDEL = integer

Initialized Value: NDEL = 0

Comments: Control NDEL is derived in the STATS Operation. It is initialized to zero (0) and is incremented by one (1) every time another delta (point to point difference) is computed.

3.11 NPTOT (Derived)

Purpose: Contains a count of the number of data values used in computing statistics for a Dependent Variable for a Flush Period.

Form: NPTOT = integer

Initialized Value: NPTOT = 0

Comments: Control NPTOT is derived in the STATS Operation. It is initialized to zero (0) and is incremented by one (1) for every data value included in the computation of the statistics, that is, for every data value included in the computation of the statistics, that is, for every data value included in the computation of the statistics, that is, for every data value included in the computation of the statistics, that is, for every data value
which is not a gap value.

3.12 NUMGAP (Derived)

Purpose: Contains a count of the number of gaps in the data for a Dependent Variable for a Flush Period.

Form: NUMGAP = integer

Initialized Value: NUMGAP = 0

Comments: Control NUMGAP is derived in the STATS Operation. It is initialized to zero (0) and is incremented by one (1) for every gap found in the data.

3.13 PTSAV (Derived)

Purpose: Contains the value of the last data value of the last Cycle from the previous call to the STATS Operation. This is used to save the last point so that a point to point difference, or delta, can be computed.

Form: PTSAV = real

Initialized Value: PTSAV = 0.0

Comments: Control PTSAV is derived in the STATS Operation. It is initialized to zero (0.0) and is assigned the last data value of the last Cycle processed in this call before returning. It is used in the next call to provide continuous point to point differencing.

3.14 RATE

Purpose: Specifies the number of data values per Cycle to be stored by this Operation for this Dependent Variable.

Form: RATE = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: RATE = 1

Diagnostics:
If NUMCYC = -1, or positive, the RATE Control must appear in the Programmer Directive file.

**Cause:** The Control RATE was not found in the Operation PD File.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

**Fix:** Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDIs is correct in the set of Directive files. If Control RATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

**Cause:** The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

**Program action:** (1) A fatal flag is set, (2) the data move is not performed.

**Fix:** Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion...
Operation, TERP, can convert any input rate to any output rate.

Dependencies: INRATE

Comments: The number of data values per Cycle given in Control RATE are stored contiguously, with the address of the first data value defined by Control INPRT.

3.15 SIGMA (Derived)

Purpose: Contains the statistic "sigma" for a Dependent Variable for a Flush Period.

Form: SIGMA = real

Initialized Value: SIGMA = 0.0

Comments: Control SIGMA is derived in the STATS Operation. It is initialized to zero (0.0) at the beginning of a Flush Period, and thereafter is the sum of the square of the data values. At the end of the Flush Period, a final computation produces the statistical value "sigma".

3.16 SKEW (Derived)

Purpose: Contains the statistic "skew" for a Dependent Variable for a Flush Period.

Form: SKEW = real

Initialized Value: SKEW = 0.0

Diagnostics:

**** OPERATION NO. 4 ***** STATS ***** OPERATION NO. 4 *****
MESSAGE NO. 205 FROM SUBPROGRAM STATS AT SECTION 3.5 EX STAGE IS 5
CALCULATIONS WERE NOT COMPLETED ON SOME VERY LARGE OR VERY SMALL VALUES
**** WARNING ********* WARNING ********* WARNING ********* WARNING ****

Cause: A machine underflow or overflow would occur if statistical computations were to continue on this Dependent Variable.

Program action: (1) A warning message is printed, and (2) GAPVAL is substituted for the value of SKEW.

Fix: Generally this message can be avoided by reducing the number of Cycles in a normal Flush Period defined by Control FLUSHP.
Comments: Control SKEW is derived in the STATS Operation. It is initialized to zero (0.0) at the beginning of a Flush Period, and thereafter is the sum of the cube of the data values. At the end of the Flush Period, a final computation produces the statistical value "skew".

A resultant value of -99. means that the skew for this Dependent Variable for this Flush Period could not be determined.

3.17 TMSAV (Derived)

Purpose: Contains the Independent Variable value for the last value of a Dependent Variable which was not equal to GAPVAL (see DRIVER document).

Form: TMSAV = real

Initialized Value: TMSAV = 0.0

Comments: Control TMSAV is derived in the STATS Operation. It is initialized to zero (0.0) and is reset to the current value of the Independent Variable each time a good data value for a Dependent Variable is encountered.

3.18 UNITS

Purpose: Defines the units of measurement of a Dependent Variable.

Form: UNITS = string

where 'string' may be up to eight (8) characters in length, where all characters are from the set of legal characters available on the target machine.

Initialized Value: UNITS = '    '

that is, eight (8) blank characters.

3.19 XMAX (Derived)

Purpose: Contains the statistic "maximum" for a Dependent Variable for a Flush Period.

Form: XMAX = real

Initialized Value: XMAX = -9.E32
Comments: Control XMAX is derived in the STATS Operation. It is initialized to a very large negative number, and thereafter is updated whenever a data value is found that is larger than the current maximum.

3.20 XMEAN (Derived)

Purpose: Contains the statistic "mean" for a Dependent Variable for a Flush Period.

Form: \( XMEAN = \text{real} \)

Initialized Value: \( XMEAN = 0.0 \)

Comments: Control XMEAN is derived in the STATS Operation. It is initialized to zero (0.0) and thereafter is the sum of the data values. At the end of a Flush Period, the sum is replaced by a computed mean.

3.21 XMIN (Derived)

Purpose: Contains the statistic "maximum" for a Dependent Variable for a Flush Period.

Form: \( XMIN = \text{real} \)

Initialized Value: \( XMIN = 9.\times10^32 \)

Comments: Control XMAX is derived in the STATS Operation. It is initialized to a very large positive number, and thereafter is updated whenever a data value is found that is smaller than the current minimum.
GENPRO PRINT Operation
REFERENCE MANUAL

Editors: Bonnie Gacnik
        Bob Lackman

GENeral Scientific Data PROcessor

SCIENTIFIC COMPUTING DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO
# PRINT CONTROLS

## CONTENTS

1. INTRODUCTION ........................................................... 3-PRINT.1

2. GENERAL CONTROLS .......................................................... 3-PRINT.2
   2.1 BEGSNP .......................................................... 3-PRINT.2
   2.2 ENDSNP .......................................................... 3-PRINT.6
   2.3 FLUSHP .......................................................... 3-PRINT.10
   2.4 IOFMT .......................................................... 3-PRINT.11
   2.5 ITYPOP (Derived) .................................................. 3-PRINT.12
   2.6 IVLAB .......................................................... 3-PRINT.14
   2.7 MAXCHR .......................................................... 3-PRINT.15
   2.8 MRATE .......................................................... 3-PRINT.15
   2.9 MSGOPT .......................................................... 3-PRINT.16
   2.10 NLINES .......................................................... 3-PRINT.17
   2.11 NOPTIN .......................................................... 3-PRINT.17
   2.12 NAMEOP (Derived) .................................................. 3-PRINT.18
   2.13 NAMLIB .......................................................... 3-PRINT.18
   2.14 NAMPD .......................................................... 3-PRINT.18
   2.15 NUMCYC .......................................................... 3-PRINT.18
   2.16 PROJECT .......................................................... 3-PRINT.20
   2.17 VCOEF .......................................................... 3-PRINT.20

3. VARIABLE CONTROLS .......................................................... 3-PRINT.22
   3.1 FORMAT .......................................................... 3-PRINT.22
   3.2 INDGET (Derived) .................................................. 3-PRINT.22
   3.3 INDPUT (Derived) .................................................. 3-PRINT.24
   3.4 INRATE (Derived) .................................................. 3-PRINT.25
   3.5 NAMVAR (Derived) .................................................. 3-PRINT.27
   3.6 PNAME .......................................................... 3-PRINT.27
   3.7 RATE .......................................................... 3-PRINT.27
   3.8 UNITS .......................................................... 3-PRINT.29

4. FUNCTION CONTROLS .......................................................... 3-PRINT.30
   4.1 NAMFUN (Derived) .................................................. 3-PRINT.30
   4.2 PGRT .......................................................... 3-PRINT.30
   4.3 NLIST (Linked) .................................................... 3-PRINT.31
1. **INTRODUCTION**

The PRINT Operation's purpose is to provide neat, tabular printed output of data values for Dependent Variables over specified intervals of the Independent Variable. The user has control over the number, size and location of intervals of the Independent Variable, the number and names of the Dependent Variables, the format specification of the data values, and the rate of data values to be printed.

The output of the PRINT Operation is divided into subparts called 'pages' which are tailored by the user through Controls to correspond to the physical device. Since normally the physical device would be a printer, then one PRINT Operation 'page' would correspond to one page of paper on the printer. However, output could be directed to a microfilm device, in which case a 'page' corresponds to a frame of microfilm.

Each PRINT Operation 'page' contains six lines of heading information at the top, which include a title, page number, and titles for each of the column headings. This is followed by the tabulated data values. All the Dependent Variables appearing on a given 'page' have the same rate so that the length of each column is identical in length.

The 'pages' of the PRINT Operation may be implicitly derived or explicitly specified. If implicitly derived, the Operation will search the Dependent Variables for common rates, gathering them together into a single 'page'. Explicit specification is performed through the optional use of the Function Controls. Each 'page' is represented by one function and the rate and a list of Dependent Variables to be printed on that page are defined through Function Controls.
2. GENERAL CONTROLS

2.1 BEGSNP

Purpose: Marks the beginning of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: \( \text{BEGSNP} = (\text{real}_1, \ldots, \text{real}_N), \ldots, (\text{real}_1, \ldots, \text{real}_N) \)

where the number of Elements in each Group, \( N \), is defined by Control IVDIM,

and where each Element represents a component of the Independent Variable from the most significant component, \( 'real_1' \), to the least significant, \( 'real_N' \).

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then, \( \text{BEGSNP} = (3., 51., 0.) \)

would cause a Snapshot Period to begin at hour 3, minute 51, and second 0.

Initialized Value: \( \text{BEGSNP} = (0.0, 0.0, 0.0) \)

This value represents the beginning of one Snapshot of an Independent Variable with three components. For a positive valued Independent Variable it also represents a Snapshot which will begin at the first Cycle of the data for any Case.

Diagnostics:

***** OPERATION NO. $ ***** P R I N T ***** OPERATION NO. $ *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'BEGSNP' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
***FATAL***FATAL***FATAL***FATAL***FATAL***FATAL***FATAL***FATAL***

Cause: The Group length of Control BEGSNP is not equal to the number of dimensioned components of the Independent Variable, $IVDIM.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM.
of the associated code release.

***** OPERATION NO. $ ***** P R I N T ***** OPERATION NO. $ ***** MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2 CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for BEGSNP and ENDSNP.

***** OPERATION NO. $ ***** P R I N T ***** OPERATION NO. $ ***** MESSAGE NO. 10 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2 CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH 'ENDSNP' VALUE '$'; ASSUMING '$'
*****WARNING********WARNING********WARNING********WARNING********WARNING*****

Cause: The value of Control BEGSNP is less than the ENDSNP of the previous Snapshot Period.

Program action: A warning message is printed.

Fix: Respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** P R I N T ***** OPERATION NO. $ ***** MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2 INCONSISTENT SNAPSHOT INTERVALS '$' THROUGH '$' HAVE BEEN TURNED OFF.
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: A previous fatal error was detected in either Control BEGSNP or ENDSNP.

Program action: All Snapshot Periods starting with the first one to produce a fatal error condition are deactivated.

Fix: Respecify consistent values for BEGSNP and ENDSNP intervals.

***** OPERATION NO. $ ***** P R I N T ***** OPERATION NO. $ ***** MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2 SNAP INTERVAL 'I' 'BEGSNP' = '$' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE. DID YOU WANT TO SKIP A CYCLE?
*****WARNING********WARNING********WARNING********WARNING********WARNING*****

Cause: The value of Control BEGSNP is one Cycle larger than the ENDSNP of the previous Snapshot Period.
Program action: A warning message is printed.

Fix: If the intent is to display all Cycles respecify the BEGSNP of interval I equal to the ENDSNP of interval I-1.

***** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
MESSAGE NO. 61 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, (2) the Operation in question is removed from processing.

Fix: Since these intervals were previously examined in DRIVER routine LODCOM, the code of the Operation in question might have modified them, or random data stores might have overwritten them. See DRIVER Controls DEBUG and DUMP for debug aid.

Multiple Snapshot Periods (Groups) can be defined on the total run interval from BEGIV to ENDIV; however, these intervals cannot overlap although the start of the current interval should repeat the end of the last interval for continuous processing. For example,

May 1, 1983

Release 1.0
BEGIV = (12.,20.,00.) $ \quad ENDIV = (14.,24.,00.)$

Over this run interval a legal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,30.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(14.,20.,00.)</td>
</tr>
</tbody>
</table>

Note that the intervals from 13.,30.,00. to 14.,00.,00. and 14.,20.,00. to 14.,24.,00. will not generate a display.

Also note that the first Snapshot Period is less than BEGIV for this Case. The DRIVER will reset it to BEGIV. If an ENDSNP value extends beyond ENDIV, a similar truncation would occur.

Finally note that the start of interval two equals the end of interval one. This insures continuous output. A warning is issued if the start of a subsequent interval is exactly one time step (see DRIVER Control DELIV) larger than the ending of a previous interval.

Over this run interval an illegal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,20.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(13.,20.,00.)</td>
</tr>
</tbody>
</table>

The second interval above is illegal because the BEGSNP of (12.,20.,00.) is less than the ENDSNP of interval one.

Also, the third interval above is illegal because the BEGSNP of (14.,00.,00.) is larger than the ENDSNP of (13.,20.,00.).

NOTE: These are fatal conditions and no processing of a display period with a fatal error nor any subsequent intervals are done even if the processor has been instructed to continue in spite of fatal errors (see DRIVER Control IABORT, value 5).

Dependencies: IVDIM, ENDSNP (see DRIVER Controls Document)

For Control BEGSNP, the number of Elements in a Group is defined by the number of components of the Independent Variable, given by DRIVER Control IVDIM.

Controls BEGSNP and ENDSNP work as ordered pairs describing the beginning and the ending of a Snapshot Period, that is, the Mth Group of Control BEGSNP marks the beginning of the Mth Snapshot Period, and the Mth Group of Control ENDSNP marks the ending of that same period. Therefore, these two Controls must have the same number of Groups.
In addition, Snapshot Periods may not overlap. Therefore, the next Group of Control BEGSNP must specify an Independent value greater than that of the previous Group of Control ENDSNP.

Comments: The number of Elements in a Group for Control BEGSNP is initially defined in the Programmer Directive file for this Operation to be three (3).

The Snapshot Periods must be an increasing function of the Independent Variable. Therefore, the Groups of Control BEGSNP must be arranged in increasing order.

All Snapshot Periods must lie within the interval of the Independent Variable delimited by Controls BEGIV and ENDIV.

2.2 ENDSNP

Purpose: Marks the ending of a Snapshot Period.

A Snapshot Period is an arbitrary interval of the Independent Variable during which the function of this Operation will be applied to the data.

Form: ENDSNP = (real1,...,realN),..., (real1,...,realN)

where the number of Elements in each Group, N, is defined by Control IVDIM,

and where each Element represents a component of the Independent Variable from the most significant component, 'real1', to the least significant, 'realN'.

EXAMPLE: If the Independent Variable is time recorded in components of hour, minute, and second, then, ENDSNP = (17., 6., 30.) would cause a Snapshot Period to end at hour 17, minute 6, and second 30.

Initialized Value: ENDSNP = (99., 99., 99.)

This value represents the ending of one Snapshot of an Independent Variable with three components.

Diagnostics:

***** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 23 FROM SUBPROGRAM LODCOM AT SECTION 3.0 EX STAGE IS 2
'CONTROL 'ENDSNP' LENGTH '$' IS INVALID. CHANGE LENGTH TO '$' IN PD FILE
*FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***
Cause: The Group length of Control ENDSNP is not equal to the number of dimensioned components of the Independent Variable, $IVDIM.$

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. The number of Elements in the DIMGEN PD Directive must equal the IFTRAN global $IVDIM$ of the associated code release.

***** OPERATION NO. $ *****   PRINT   ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for BEGSNP and ENDSNP.

***** OPERATION NO. $ *****   PRINT   ***** OPERATION NO. $ *****
MESSAGE NO. 70 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAPSHOT INTERVAL '$' 'ENDSNP' = '$' EXCEEDS 'ENDSNP' '='$' OF PRIOR INTERVAL
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control ENDSNP is less than the ENDSNP of the previous Snapshot Period.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Respecify consistent values for the Snapshot Periods.

***** OPERATION NO. $ *****   PRINT   ***** OPERATION NO. $ *****
MESSAGE NO. 63 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
INCONSISTENT SNAPSHOT INTERVALS '$' THROUGH '$' HAVE BEEN TURNED OFF.
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: A previous fatal error was detected in either Control BEGSNP or ENDSNP.

Program action: All Snapshot Periods starting with the first one to produce a fatal error condition are deactivated.

Fix: Respecify consistent values for BEGSNP and ENDSNP intervals.
MESSAGE NO. 68 FROM SUBPROGRAM LODCOM AT SECTION 3.6 EX STAGE IS 2
SNAP INTERVAL 'I' 'BEGSNP' = '$' EXCEEDS INTERVAL 'I-1' 'ENDSNP' BY 1 CYCLE.
DID YOU WANT TO SKIP A CYCLE?

**WARNING**

Cause: The value of Control BEGSNP is one Cycle larger than the ENDSNP of the previous Snapshot Period.

Program action: A warning message is printed.

Fix: If the intent is to display all Cycles Respecify the BEGSNP of interval I equal to the ENDSNP of interval I-1.

MESSAGE NO. 9 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
CONTROL 'BEGSNP' VALUE '$' INCONSISTENT WITH CONTROL 'ENDSNP' VALUE '$'
MESSAGE NO. 61 FROM SUBPROGRAM NEWSNP AT SECTION 1.3 EX STAGE IS 5
OPERATION '$' TERMINATED; NUMCYC SET TO 0

**FATAL**

Cause: The value of Control BEGSNP is greater than or equal to ENDSNP, the end time of this Snapshot Period.

Program action: (1) A fatal error flag is set, (2) the Operation in question is removed from processing.

Fix: Since these intervals were previously examined in DRIVER routine LODCOM, the code of the Operation in question might have modified them, or random data stores might have overwritten them. See DRIVER Controls DEBUG and DUMP for debug aid.

Multiple Snapshot Periods (Groups) can be defined on the total run interval from BEGIV to ENDIV; however, these intervals cannot overlap, but the start of the current interval should repeat the end of the last interval for continuous processing. For example,
Over this run interval a legal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>Interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(12.,00.,00.)</td>
<td>(12.,30.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(99.,99.,99.)</td>
</tr>
</tbody>
</table>

Note that the intervals from 13.,30.,00. to 14.,00.,00. and 14.,20.,00. to 14.,24.,00. will not generate a display.

Also note that the last Snapshot Period exceeds ENDIV for this Case. The DRIVER will reset it to ENDIV. A similar truncation of a BEGSNP that starts before BEGIV would occur.

Finally note that the start of interval two equals the end of interval one. This insures continuous output. A warning is issued if the start of a subsequent interval is exactly one time step (see DRIVER Control DELIV) larger than the ending of a previous interval.

Over this run interval an illegal set of Snapshot Periods would be:

<table>
<thead>
<tr>
<th>Interval</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGSNP</td>
<td>(00.,00.,00.)</td>
<td>(12.,20.,00.)</td>
<td>(14.,00.,00.)</td>
</tr>
<tr>
<td>ENDSNP</td>
<td>(12.,30.,00.)</td>
<td>(13.,30.,00.)</td>
<td>(13.,20.,00.)</td>
</tr>
</tbody>
</table>

The second interval above is illegal because the BEGSNP of (12.,20.,00.) is less than the ENDSNP of interval one.

Also, the third interval above is illegal because the BEGSNP of (14.,00.,00.) is larger than the ENDSNP of (13.,20.,00.).

NOTE: These are fatal conditions and no processing of a display period with a fatal error nor any subsequent intervals is done even if the processor has been instructed to continue in spite of fatal errors (see DRIVER Control IABORT, value 5).

Dependencies: BEGSNP, IVDIM (see DRIVER Controls Document)

For Control ENDSNP, the number of Elements in a Group is defined by DRIVER Control IVDIM.

Controls BEGSNP and ENDSNP work as ordered pairs describing the beginning and the ending of a Snapshot Period, that is, the Mth Group of Control BEGSNP marks the beginning of the Mth Snapshot Period, and the Mth Group of Control ENDSNP marks the ending of that same period. Therefore, these two Controls must have the same number of Groups.

In addition, Snapshot Periods may not overlap. This implies that the
next Group of Control BEGSNP must specify an Independent Variable value greater than that of the previous Group of Control ENDSNP.

Comments: The number of Elements in a Group for Control ENDSNP is initially defined in the Programmer Directive file for this Operation to be three (3).

The Snapshot Periods must be an increasing function of the Independent Variable. This implies that the Groups of the Control ENDSNP must be arranged in increasing order.

All Snapshot Periods must lie within the interval of the Independent Variable delimited by DRIVER Controls BEGIV and ENDIV. **NOTE:** The last Cycle to be displayed is one Cycle **BEFORE** the value given in Control ENDSNP.

2.3 **FLUSHP**

Purpose: Sets the length of a Flush Period.

A Flush Period is a sub-interval of the Snapshot Period marking the frequency of a specific action. At the end of the specific action, all data within the Flush Period is discarded, or "flushed", and execution on subsequent data within the Snapshot Period is continued.

Form: \[ \text{FLUSHP} = \text{integer} \]

where 'integer' is the number of Cycles in the Flush Period.

Initialized Value: \[ \text{FLUSHP} = 900 \]

For data with an Independent Variable of Time with each Cycle representing one second in time, this number represents fifteen minutes of data.

Diagnostics:

***** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 2.2 EX STAGE IS 2
CONTROL 'FLUSHP' MISSING FROM CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: Control FLUSHP is missing from the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify
that the ordering of the PDs and UDs is correct in the set of Directive files. If Control FLUSHP is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENFRO consultant.

***** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 6 FROM SUBPROGRAM LODCOM AT SECTION 3.5 EX STAGE IS 2
CONTROL 'FLUSHP' VALUE '$' INVALID; VALID RANGE '$' TO '$'; ASSUMING '$'
*****WARNING*******WARNING*******WARNING*******WARNING*******WARNING*******

Cause: The value of Control FLUSHP is less than or equal to zero (0.).

Program action: (1) A warning message is printed, and (2) the interval for displays, FLUSHP, is reset to the full run interval of the Case.

Fix: If specific intervals for display are wanted, respecify and correlate Controls, BEGSNP, ENDSNP, and FLUSHP.

Dependencies: BEGSNP, ENDSNP, DELIV (see DRIVER Controls Document)

Controls BEGSNP and ENDSNP define the Snapshot Periods of which the Flush Period is a sub-interval.

Comments: The Flush Period begins with the first Cycle of data in the Snapshot Period. A counter is employed to count the number of Cycles from the beginning of the Flush Period to the length of the Flush Period defined in Control FLUSHP or until the end of the Snapshot Period. If the number of Cycles remaining in a Snapshot Period is greater than the Flush Period, then a "flush" is performed, the count of Cycles in a Flush Period is reset to zero (0), and a new Flush Period is begun. If the number of Cycles remaining in the Snapshot Period is less than the Flush Period, then a "flush" is performed on an abbreviated Flush Period, marked by the end of the Snapshot Period.

2.4 IOFMT

Purpose: Flags whether the Dependent Variable format specifications are to be contained in memory or can be temporarily stored on disk to save memory space.

Form: IOFMT = integer where 'integer' must be one of:

0 All format specifications will be contained in memory

1 All format specifications will be stored on disk until needed
Initialized Value: IOFMT = 0

Diagnostics:

XXXXXXXXXX FATAL-FATAL-FATAL XXXXXXXXXX

SUB RDOPFL, OP 'S' -- UNABLE TO FIND IOFMT
IN GENERAL CONTROL SECTION OF CONTROL ARRAY --

Cause: Control missing from Control Array.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control BEGIV is not in the DRIVER PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

Comments: If memory availability becomes a problem, the space normally used for storage of the display formats can in effect be eliminated, by storing the formats on disk until needed at flush time. The trade-off of course, will be greater execution time due to added disk accesses.

2.5 ITYPOP (Derived)

Purpose: Specifies the type of Operation.

Form: ITYPOP = integer

where 'integer' is one of:

-1 In-Place Transformation Operation
0 Snapshot Operation
1 Input Operation
2 Transformation Operation

Initialized Value: ITYPOP = 0

Diagnostics:
MESSAGE NO. 2 FROM SUBPROGRAM LOCDOM AT SECTION 3.0 EX STAGE IS 2
CONTROL 'ITYPOP' MISSING FROM CONTROL ARRAY
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The Control ITYPOP was not found in the PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control ITYPOP is not in the PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

MESSAGE NO. 24 FROM SUBPROGRAM LOCDOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INVALID.

MESSAGE NO. 61 FROM SUBPROGRAM LOCDOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The value of Control ITYPOP is less than a minus one (-1) or greater than two (2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

MESSAGE NO. 9 FROM SUBPROGRAM LOCDOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'
MESSAGE NO. 61 FROM SUBPROGRAM LOCDOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL******FATAL******FATAL******FATAL******FATAL******FATAL***

Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).
Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is an Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

Dependencies: NUMCYC

Comments: The value of this Control is derived from the Programmer Directive file.

2.6 IVLAB

Purpose: Provides the titles of the column headings for the display variable components.

The display variable is a variable converted from the Independent Variable for the purpose of clarity in interpreting the output.

Form: $\text{IVLAB} = (\text{string}_1, \ldots, \text{string}_N)$

where $N$ is the number of components in the Display variable,

where 'string$_1$' is the title of the column heading of the most significant component of the Display variable, etc.,

and where 'string$_N$' is the title of the column heading of the least significant component of the Display variable.

Initialized Value: $\text{IVLAB} = ("HR", "MN", "SEC")$

Diagnostics:

$\text{xxxxxxxxxxx  FATAL-FATAL-FATAL  xxxxxxxxxx}
\text{SUB RDOPFL, OP '$'}$

IVLAB AND VCOEF SCALING N-TUPLES DO NOT AGREE IN NUMBER.
NUMBER OF VCOEF FACTORS = '$$
NUMBER OF IVLAB LABELS = '$$
$\text{xxxxxxxxxx  xxxxxxxxxx}

Cause: The number of Groups in Controls IVLAB and VCOEF are not equal.
Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: When specifying the values for Controls IVLAB and VCOEF, make sure the number of Groups specified for each Control are equal.

Dependencies: VCOEF

Comments: The values of Control VCOEF provide the conversion factors for each component of the Display variable. The number of values for Controls IVLAB and VCOEF must be equivalent or an error message is generated.

The format field widths for the components of the Display variable are computed in the PRINT Operation. For aesthetic purposes the number of characters in each string of Control IVLAB should be adjusted to correspond to the number of spaces reserved in each format field width. In general \((\text{the number of characters specified in each element of the IVLAB n-tuple}) + 2\) will be the field width of the format generated for the first \(n-1\) elements of the n-tuple. The format generated for the \(n\)th element is more dependent on the requested print rates.

2.7 MAXCHR

Purpose: Specifies the maximum number of characters which can be printed on one line of the output device to which tabular output is to be written.

Form: \(\text{MAXCHR} = \text{integer}\)

where 'integer' is positive and is dependent on the physical limitation of the output device.

Initialized Value: \(\text{MAXCHR} = 128\)

Comments: This Control is useful for statistical and tabular displays. Standard output to a paper printer might use 128 characters per line while terminal output might be restricted to 78 characters per line.

The PRINT Operation uses MAXCHR to regulate the columnar width of its tabular display.

2.8 MRATE

Purpose: Selects a single, master rate at which all Dependent Variables available to the PRINT Operation will be printed.

Form: \(\text{MRATE} = \text{integer}\)

where 'integer' must be one of:
0 No master rate is specified. Print rate will default to: 1) the rate specified by PGRT if Page Functions are defined, or 2) the value specified by the RATE Control for the Dependent Variable.

>0 This value is to be taken as the master rate for all Dependent Variables

Initialized Value: MRATE = 0

Diagnostics:

'\$' HAS INPUT RATE '\$'; NOT AN EVEN MULTIPLE OF THE ASKED RATE '\$'

XXXXXXXXXX WARNING-WARNING-WARNING XXXXXXXXX

BECAUSE OF THE ABOVE INPUT RATE INCONSISTENCIES
THE ASKED RATE FOR THIS REQUEST HAS BEEN RESET TO
THE G.C.D. 1

Cause: The master rate given in Control MRATE is not a common denominator of all Dependent Variables of the Variable Area Vector.

Program action: (1) A warning message is issued and (2) Control MRATE is set to one (1).

Fix: Set Control MRATE to a common denominator of all Dependent Variables of the Area Vector.

Dependencies: RATE

Comments: The master rate given in Control MRATE must be a common denominator of all Dependent Variables available to the PRINT Operation, or an error message will be generated and a value which is a common denominator will be substituted.

2.9 MSGOPT

Purpose: Print flag which enables the user to display different amounts information.

Form: IVPR = integer

where 'integer' is one of;

0 Only error messages are printed

1 Error messages and informational tables are printed
2 Error messages, informational tables, generated formats, Page Function requests, and diagnostic information related to file and Cycle allocation

Initialized Value: MSGOPT = 0

2.10 N L I N E S

Purpose: Specifies the maximum number of lines which can be printed on one page of the output device to which tabular output is to be written.

Form: \[ NLINES = \text{integer} \]

where 'integer' is positive and is dependent on the physical limitation of the output device minus six (6).

Initialized Value: \( NLINES = 60 \)

The six line heading at the top of every page generated by the PRINT Operation, which includes a title, labels, and units, is not included in the count of lines given in Control N L I N E S: its value is based on the number of lines of data values which may be printed on a page.

2.11 N O P T I N

Purpose: Flags whether all Dependent Variables, or only those included in a page list provided by Function Controls, will be printed.

Form: \[ NOPTIN = \text{integer} \]

where 'integer' must be one of:

0 Only Dependent Variables explicitly stated in the NLIST Function Control for a given function (page) will be printed

1 The above will be printed, but also all other Dependent Variables not mentioned in the NLIST Function Control of any function (page) will be printed at the rate of one data value per Cycle

Initialized Value: \( NOPTIN = 0 \)

Comments: The value of Control NOPTIN will be ignored if no pages are specifically requested through the use of Function Controls. See the Function Controls below for a more detailed description.
.12 NAMEOP (Derived)

Purpose: Assigns a unique name to this Operation, which is used in informational and diagnostic prints.

Form: \[ \text{NAMEOP} = \text{name} \]

Initialized Value: \[ \text{NAMEOP} = \text{PRINT} \]

Comments: The value of this Control is derived from the Programmer Directive file.

2.13 NAMLIB

Purpose: Provides file name(s) which comprise all required code for this Operation if stored on some permanent storage medium (such as PLIB on the CDC 7600).

Form: \[ \text{NAMLIB} = \text{string}_1, \ldots, \text{string}_N \]

where each 'string' is a file name.

Initialized Value: \[ \text{NAMLIB} = \text{'}NONE\text{'} \]

Comments: This Control is currently nonfunctional.

2.14 NAMPD

Purpose: Provides the name of the Programmer Directive file for this Operation.

Form: \[ \text{NAMPD} = \text{string} \]

Initialized Value: \[ \text{NAMPD} = \text{'}NONE\text{'} \]

Comments: This Control is currently nonfunctional.

2.15 NUMCYC

Purpose: Defines the amount of space, in units of the number of data Cycles, which is to be allocated for this Operation's use.

Form: \[ \text{NUMCYC} = \text{integer} \]

where 'integer' is one of:
Execution of this Operation is suspended for this run.

-1 The DRIVER optimizes the number of Cycles to be allocated based on the best flow for all Operations involved in this run.

Initialized Value: NUMCYC = 0

Diagnostics:

***** OPERATION NO. $ ***** P R I N T ***** OPERATION NO. $ *****
MESSAGE NO. 24 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'NUMCYC' VALUE '$' INVALID.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control NUMCYC is less than a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.

***** OPERATION NO. $ ***** P R I N T ***** OPERATION NO. $ *****
MESSAGE NO. 9 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
CONTROL 'ITYPOP' VALUE '$' INCONSISTENT WITH CONTROL 'NUMCYC' VALUE '-2'.
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 3.4 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The value of Control ITYPOP is greater than zero (0) and NUMCYC equals a minus two (-2).

Program action: (1) A fatal error flag is set, (2) the involved Operation is turned off by setting NUMCYC = 0, and (3) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: There is a Operation incompatibility. ITYPOP greater than zero defines a Transformation Operation, whereas, NUMCYC = -2 is only valid for Snapshot Operations. One is invalid. Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Also check the ordering and completeness of the PD and UD Directive sets. If the problem does not surface, contact a GENPRO consultant.
Dependencies: ITYPOP, RATE

Comments: A zero (0) value for Control NUMCYC causes this Operation to be turned off. This allows an Operation to be turned off without removing the Programmer Directive and the User Directive files.

2.16 PROJECT

Purpose: Provides a textual description or title of the project, which is used to label the various output generated by this Operation.

Form: PROJECT = string

where 'string' may be up to forty (40) characters in length.

Initialized Value: PROJECT = 'that is, forty (40) blank characters.

2.17 VCOEF

Purpose: Provides the conversion factors of the Display Variable components.

The Display Variable is a variable converted from the Independent Variable for the purpose of clarity in interpreting the output.

Form: VCOEF = (real1, ..., realN)

where N is the number of components in the Display Variable,

where 'real1' is the conversion factor of the most significant component of the Display variable, etc.,

and where 'stringN' is the conversion factor of the least significant component of the Display variable.

Initialized Value: VCOEF = (3600., 60., 60.)

Diagnostics:

xxxxxxxxxxxxx FATAL-FATAL-FATAL xxxxxxxxxx
SUB RDOPFL, OP '$'
IVLAB AND VCOEF SCALING N-TUPLES DO NOT AGREE IN NUMBER.
NUMBER OF VCOEF FACTORS = '$'
NUMBER OF IVLAB LABELS = '$'
xxxxxxxxxxxxx xxxxxxxxxx
Cause: The number of Groups for Controls IVLAB and VCOEF are not equal.

Program action: (1) A fatal error flag is set, and (2) Execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: When specifying the values for Controls IVLAB and VCOEF, make sure the number of Groups specified for each Control are equal.

Dependencies: IVLAB

Comments: The values of Control IVLAB provide the titles to the column heading for each component of the Display variable. The number of values for Controls IVLAB and VCOEF must be equivalent or an error message is generated.
3. VARIABLE CONTROLS

3.1 FORMAT

Purpose: Describes the format specification for outputing this Dependent Variable.

Form:  \texttt{FORMAT = string}

where \texttt{string} must be one of the following FORTRAN formats:

- \texttt{Ew.d} Single precision floating point value with exponent
- \texttt{Fw.d} Single precision floating point value without exponent
- \texttt{Gw.d} Most significant digit editing
- \texttt{Iw} Decimal integer
- \texttt{Ow} Octal integer
- \texttt{Aw} Alphanumeric
- \texttt{Lw} Logical

Initialized Value: \texttt{FORMAT = 'F7.2'}

Comments: The format that is selected for a Dependent Variable must be consistent with the magnitude and type of the data values of that Dependent Variable.

The selected format must be available in the FORTRAN compiler of the target machine.

Apostrophe ('), quote ('"), and asterisk ('*') editing descriptors, Tn and Xn positional editing descriptors, and kP scaling, are not implemented at this time, and are thus not allowable.

3.2 INDGET (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the input to this Operation for this call, is obtained.

Form:  \texttt{INDGET = integer}

where \texttt{integer} is greater than or equal to zero (0).
Initialized Value: INDGET = 0

Diagnostics:

**** OPERATION NO. $ ***** PRINT **** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDGET' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
****FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The Control INDGET was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDGET is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
****FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********

Cause: The data indices associated with Control INDGET are out of range of the SPACE partition where the data is stored.

Program action: (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

Fix: The value assigned to Control INDGET is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indices throughout the entire program flow. If an INDGET index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.
Dependencies: INRATE

Comments: The value of Control INDGET is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control INRATE, and by the number of Cycles passed to this Operation in this call, are stored contiguously.

3.3 INDPUT (Derived)

Purpose: Designates a subscript index of a Dependent Variable where data, which is the result from this Operation for this call, is stored.

Form: INDPUT = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: INDPUT = 0

Diagnostics:

***** OPERATION NO. $ *****  PRINT  ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INDPUT' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The Control INDPUT was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INDPUT is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

***** OPERATION NO. $ *****  PRINT  ***** OPERATION NO. $ *****
MESSAGE NO. 69 FROM SUBPROGRAM SETIND AT SECTION 2.6 EX STAGE IS 2
DATA '$' INDICES OUT OF RANGE FOR '$' VARIABLES ON CALL '$';
OFTEN DUE TO AN OVERWRITTEN CONTROL ARRAY
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***
**Cause:** The data indices associated with Control INDPUT are out of range of the SPACE partition where the data is stored.

**Program action:** (1) A fatal flag is set, (2) execution continues subject to the value of DRIVER Control IABORT.

**Fix:** The value assigned to Control INDPUT is derived in the SETIND routine of the DRIVER, which dynamically updates these subscript indecies throughout the entire program flow. If an INDPUT index lies outside of the range of the data array being addressed, this usually means that some new code being added to GENPRO has generated wild stores which have overwritten the valid indices.

To pursue the problem activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

**Dependencies:** RATE

**Comments:** The value of Control INDPUT is the subscript index of the first data value of this Dependent Variable. Other data values, defined by the rate of this Dependent Variable in Control RATE, and by the number of Cycles available to this Operation in this call, are stored contiguously.

### 3.4 INRATE (Derived)

**Purpose:** Specifies the number of data values per Cycle to be input to this Operation for this Dependent Variable.

**Form:** INRATE = integer

where 'integer' is greater than or equal to zero (0).

**Initialized Value:** INRATE = 0

**Diagnostics:**

```
***** OPERATION NO. $ *****  P R I N T  ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'INRATE' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0
***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***
```

**Cause:** The Control INRATE was not found in the Operation PD File.

**Program action:** (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.
Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control INRATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

**** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'
FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL

**** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOBMV AT SECTION 1.2 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'
FATAL********FATAL********FATAL********FATAL********FATAL********FATAL********FATAL

Cause: The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

Program action: (1) A fatal flag is set, (2) the data move is not performed.

Fix: Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion Operation, TERP, can convert any input rate to any output rate.

Dependencies: INDGET, RATE

Comments: The value assigned to Control INRATE is derived in the DRIVER from the value of Control RATE of the last Transformation Operation.

When the value of Control INRATE is zero (0), no data is to be passed to this Operation, and thus no function is performed for this Dependent Variable.

The number of data values per Cycle given in Control INRATE are stored contiguously, with the address of the first data value defined by Control INDGET.
3.5 NAMVAR (Derived)

Purpose: Assigns a name to be associated with a Dependent Variable.

Form: NAMVAR = name

Initialized Value: NAMVAR = NAMVAR

Comments: The value of Control NAMVAR is indirectly derived in the Translator by the Area Vector directives which manipulate items of the Variable Area Vector.

3.6 PNAME

Purpose: Designates a character string to be used as an alternate to the name (given in Control NAMVAR) in labeling the column of data values of a Dependent Variable.

Form: PNAME = 'string'

where 'string' is up to eight (8) characters in length.

If the 'string' of characters in Control PNAME is all blank, then the value of Control NAMVAR will be used to label the column of the data values of this Dependent Variable.

Initialized Value: PNAME = '

Comments: This option could be used as a simple way of adding a descriptive label to Dependent Variables, as they move through various Transformation Operations.

3.7 RATE

Purpose: Specifies the number of data values per Cycle to be stored by this Operation for this Dependent Variable.

Form: RATE = integer

where 'integer' is greater than or equal to zero (0).

Initialized Value: RATE = 1

Diagnostics:
"**** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 2 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
CONTROL 'RATE' MISSING FROM CONTROL ARRAY
MESSAGE NO. 61 FROM SUBPROGRAM LODCOM AT SECTION 4.2 EX STAGE IS 2
OPERATION '$' TERMINATED; NUMCYC SET TO 0

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

If NUMCYC = -1, or positive, the RATE Control must appear in the Programmer Directive file.

Cause: The Control RATE was not found in the Operation PD File.

Program action: (1) A fatal error flag is set, and (2) execution continues subject to the value assigned to DRIVER Control IABORT.

Fix: Check the Programmer Directive File for a valid release number. PD release numbers must match the code release numbers. Then verify that the ordering of the PDs and UDs is correct in the set of Directive files. If Control RATE is not in the Operation PD, that PD is a wrong release or it has been corrupted.

If the Directives are correct, activate the DRIVER Control DEBUG, option CONTROLS. Also consider the use of DRIVER Control DUMP. If the problem does not surface take your printouts to a GENPRO consultant.

"**** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM DATAMV AT SECTION 1.5 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

"**** OPERATION NO. $ ***** PRINT ***** OPERATION NO. $ *****
MESSAGE NO. 17 FROM SUBPROGRAM ATOBMV AT SECTION 1.2 EX STAGE IS 2
CONTROL 'INRATE' VALUE '$' ITEM '$' INCONSISTENT WITH CONTROL 'RATE' VALUE '$'

***FATAL********FATAL********FATAL********FATAL********FATAL********FATAL***

Cause: The ratio of INRATE/RATE is not an integer for some Dependent Variable (ITEM).

Program action: (1) A fatal flag is set, (2) the data move is not performed.

Fix: Activate DRIVER Control PRINT, option (RATE). The resulting display will show the rates of all Dependent Variables being processed at the entry and exit of each Operation in the Operation Sequence. Correct the rate inconsistency of the appropriate Operations.

Most Snapshot Operations can do an integral point pickoff. Most Transformation Operations require the rates associated with Controls INRATE (rate of data input to the Operation) and RATE (rate of data output by that Operation) to be the same. The rate conversion
Operation, TERP, can convert any input rate to any output rate.

Diagnostics:

```
XXX WARNING-WARNING-WARNING XXXXXXXXXX
SUB FILE, OP '$'
'$$' HAS A INPUT RATE OF '$'
WHICH IS NOT AN EVEN DIVISOR OF THE ASKED RATE OF '$'
```

**Cause:** Control RATE does not evenly divide Control INRATE

**Program action:** (1) A warning message is printed, and (2) Control RATE is set to Control INRATE

**Fix:** Define Control RATE such that it evenly divides Control INRATE

**Dependencies:** INRATE

**Comments:** When the value of Control RATE is zero (0), no data is to be printed and therefore this Dependent Variable is skipped.

The number of data values per Cycle given in Control RATE are stored contiguously, with the address of the first data value defined by Control INDPUT.

### 3.8 UNITS

**Purpose:** Defines the units of measurement of a Dependent Variable.

**Form:** \[ \text{UNITS} = \text{string} \]

where 'string' may be up to eight (8) characters in length, where all characters are from the set of legal characters available on the target machine.

**Initialized Value:** \[ \text{UNITS} = ' ' \]

that is, eight (8) blank characters.
4. FUNCTION CONTROLS

4.1 NAMFUN (Derived)

Purpose: Assigns a name to be associated with a function.

Form: NAMFUN = name

Initialized Value: NAMFUN = NAMFUN

Comments: The value of Control NAMFUN is indirectly derived in the Translator by the Area Vector directives which manipulate items of the Function Area Vector.

4.2 PGRT

Purpose: Declares the rate at which data values of all Dependent Variables will be output for this page.

Form: PGRT = integer

where 'integer' is non-negative and is a common denominator of all Dependent Variables to be included on this page.

Initialized Value: PGRT = 0

Diagnostics:

'$' HAS INPUT RATE '$'; NOT AN EVEN MULTIPLE OF THE ASKED RATE '$'

XXXXXXXXXX WARNING-WARNING-WARNING XXXXXXXXXX

BECAUSE OF THE ABOVE INPUT RATE INCONSISTENCIES
THE ASKED RATE FOR THIS REQUEST HAS BEEN RESET TO
THE G.C.D. 1

Cause: The value of Control PGRT is not a common denominator of all Dependent Variables to be included on this page (defined in Control NLIST).

Program action: (1) A diagnostic message is generated, and (2) the value of Control PGRT is then reset to the greatest common denominator of all the Dependent Variables to be included on this page.

Fix: Define Controls PGRT, NLIST, and RATE such that all rate are compatible

May 1, 1983

Release 1.0
OPERATION CONTROLS

Dependencies: NLIST, RATE

Comments: The value of Control PGRT must be a common denominator of the rates, given in Control RATE, of all the Dependent Variables to be included on this page, given in Control NLIST.

4.3 NLIST (Linked)

Purpose: Lists the names of the Dependent Variables to be included on this page.

Form: NLIST = name1, ..., nameN

where N is the total number of Dependent Variables to be included on this page,

and where each 'name' must be one of:

NOLINK No Dependent Variables are included on this page, and therefore, this page is skipped

nnnn A set of characters, which exactly matches the value of Control NAMVAR for the desired Dependent Variable to be included on this page

Initialized Value: NLIST = NOLINK

Diagnostics:

xxxxxxxxxxxx xxxxxxxxxxxxxxxxx

SUB RDOPFL, OP'$'
PAGE RATE '$' '$' WAS REQUESTED IN PAGE OPTION,
BUT WAS NOT DEFINED IN OPERATIONS VECVAR
--- DELETED FROM LIST ---

xxxxxxxxxxxx xxxxxxxxxxxxxxxxx

Cause: A value of Control NLIST does not exactly match the value of Control NAMVAR for any Dependent Variable, usually this indicated that a name in the list has been mis-spelled

Program action: (1) A diagnostic message is generated, and (2) the value that does not correspond is then removed from the list of Dependent Variables to be included on this page

Fix: Take care not to mis-spell the names contained in the Control NLIST

May 1, 1983
REFERENCE MANUAL 3-PRINT. 31
Release 1.0

May 1, 1983
REFERENCE MANUAL 3-PRINT. 31
Release 1.0
Dependencies: NAMVAR

Comments: Each value of Control NLIST must match the value of Control NAMVAR for some Dependent Variable, or a diagnostic message will be generated, and that value removed from the list of values to be included in this page.
The GENPRO System
Implementor's Guide

Editor: Bob Lackman

GENeral Scientific Data PROcessor

SCIENTIFIC COMPUTING DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO
NCAR is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation.
# TABLE OF CONTENTS

**GENPRO SYSTEM IMPLEMENTATION GUIDE** .......................... 1-1

**HOST ENVIRONMENT** ............................................. 1-1

- Machine Dependencies ........................................... 1-1
- Uniqueness of Names ........................................... 1-2
- Computational Accuracy due to Word Length ..................... 1-2
- Work Array Dimensions ........................................ 1-2
- Graphics .......................................................... 1-3
- Holerith Data .................................................... 1-3
- Character Manipulations ...................................... 1-3
- Bit Manipulations ............................................... 1-3
- Tape or Disk I/O .................................................. 1-4
- Random I/O ....................................................... 1-4
- Miscellaneous .................................................... 1-4

**SOURCE LANGUAGE** ............................................... 1-4

**GENPRO FILES** ................................................... 1-5

**FILE ASSIGNMENTS** ............................................... 1-6

- Error Messages ................................................... 1-6
- Directives ........................................................ 1-6
- Input Data ........................................................ 1-13
- Output Data ...................................................... 1-13
- Intermediate Data .............................................. 1-13
- Graphics Output ................................................ 1-13
- Tabular Data Listing .......................................... 1-14
- Printouts ........................................................ 1-14

**COMPILING** ........................................................ 1-14

**LINKING** .......................................................... 1-14

**LOADING and RUNNING** ............................................ 1-14

- CRAY COS JCL ..................................................... 1-15
- VAX Command File .............................................. 1-16

**CRACKING TAPE FORMATS** ........................................ 1-17

Table 1. GENPRO System Files on Magnetic Tape .............. 1-5
Table 2. Directives of a Typical GENPRO Case ................ 1-7
Table 3. Independent Variable Control Set ................... 1-18

NCAR Graphics Interface ............................................ Appendix A.
Utility Interfaces ................................................... Appendix B.
IFTRAN Preprocessor .............................................. Appendix C.
HOST ENVIRONMENT

All or substantial parts of GENPRO have been compiled and executed on the following list of computers:

1. CRAY1/COS/CFT
2. CDC 7600/NCAR/NCAR FORTRAN
3. CDC 203,205/NOS/FORTRAN 66
4. DEC VAX 11/780/VMS/FORTRAN 4+...
5. IBM 3081,4341/VM-CMS/FORTGI,FORTVS

Machine Dependencies

As a whole, GENPRO is a large system. A number of options do exist, however, to alter the actual size which must be loaded to accomplish a given application, or class of applications. These include the judicious selection of array dimension sizes, the number of functional Operations which can be performed in a single Case, and the number of code modules and support subroutines to be loaded. The amount of virtual or actual memory needed to load all of the code with array dimensions large enough to handle most applications is approximately 1.5 Mbytes. This figure also includes the usual system library support routines such as SIN, COS, etc.

On today's paging and virtual memory operating systems, the simplest approach is just to load all code, whether it is used in the current application or not. This is the most efficient approach from the manpower point of view because one need not recompile and relink for each new application. Moreover, it is efficient from the machine point of view in that only those pages of code which are being executed will actually be held in memory.

Because it is a large total system, the Scientific Computing Division of NCAR does not support porting of the system to smaller machines than those with 32 bit index addressing and 2 Mbytes of real or virtual memory. Thus GENPRO is targeted for machines of the size of a VAX or an IBM 4341, or larger. In fact, because of the functional stacking approach of GENPRO, it is an extremely efficient data processing system even on super-computers such as the CRAY.

Within the range of applicable machines, there are a number of machine specific topics. The two major ones are word (NSU) size in bytes and bits, and low level system utilities.
Word size inconsistencies are handled by using a code preprocessor called IFTRAN. The specific features of IFTRAN which allow the output FORTRAN to be tailored to a particular machine are conditional compilation and token replacements. Sites to receive GENPRO can request either IFTRAN source and the portable (written in FORTRAN) IFTRAN preprocessor, or specific FORTRAN targeted for their machine.

The lower level system utilities to support such functions as bit manipulations, tape or disk I/O, and graphics must be provided by the site if appropriate versions are not already available.

**Uniqueness of Names**

Names of variables defined in User Directives (UDS), as well as the Controls (option switches) of the Operations, must be alphanumeric, start with a letter, and must be unique in the number of characters of the host computer. That is, on a VAX or IBM machine these names must be unique in the first 4 characters. On the CRAY they must be unique in the first 8 characters, etc. For example, the variables PRES1, and PRES2 could not be processed on a VAX (4 byte NSU machine) because they would have the same internal name PRES. A user could select the alternative names PR1 and PR2 instead.

All Controls of the existing Operations have already been selected such that they are unique to 4 characters. Only if a user is adding an Operation of their own do they need to worry about the uniqueness of Control names.

**Computational Accuracy due to Word Length**

The accuracy of computations such as those performed in the STATS (statistics) Operation are affected by the computer word (NSU) size. That Operation generates a warning when overflows or underflows would occur.

Another area that can present problems is in the I/O scaling of data from floating point to positive integer and vice versa. These conversions involve Controls called TERM (additive term) and FACTOR (multiplicative factor) in the INPUT and OUTPUT Operations.

**Work Array Dimensions**

All data being processed by GENPRO is stored within partitions of a single large array called SPACE. It is equvalenced to the additional aliases of CON, ICON, DATA, IDATA, SAV, ISAV, BUF, IBUF, and SCR, ISCR. When the system is generated from the IFTRAN, the size of this array ($ISPACE) is selected. For smaller systems such as a VAX a size of 50000 words is typically chosen. On CRAY and other large systems 200000 words has been used. The optimum number to be used is dependent upon the number of Operations to be stacked into a single Case, the sample rates of the variables, and the number of variables to be processed.
GENPRO can process any length time series because the Operations are called as many times as is necessary with a moving Window of the data being processed. In general, the larger the data Window, the more efficient the processor runs because there are fewer calls and larger data strings to operate upon. Thus a larger $ISPACE$ array dimension means a larger data Window and improved efficiency.

Graphics
Graphics are generally available through the portable NCAR system graphics package. This package has been ported and installed at approximately 1300 national and international sites. GENPRO uses this graphics facility exclusively.

A list of sites which has obtained the graphics package is available for each computer vendor, such as IBM, CDC, PRIME, etc. A potential site might be able to receive help from one of these sites. To obtain such a list contact:

Sue Long
National Center for Atmospheric Research
Box 3000
Boulder, Colorado 80303

The NCAR graphics software interface is described in Appendix A.

Holerith Data
GENPRO was originally written in FORTRAN 66. The only required extension to the FORTRAN 77 standard is the holerith data type. At the time of writing of this document the IBM FORTRAN 77 compiler did not have this extension; thus, the FORTRAN 66 option was required.

DEC, CRAY, CDC, and PRIME compilers do have the holerith extension so GENPRO can be compiled under FORTRAN 77 on those machines.

Character Manipulations
ENCODE and DECODE are system functions provided on the DEC, CRAY, CDC, and PRIME systems for character and format manipulations. These system functions do not exist on IBM systems. An IBM simulation of this facility is available for code loading and checkout on IBM systems; however, it is much too inefficient for actual production usage.

A potential IBM site would thus have to acquire or develop a more efficient ENCODE/DECODE facility.

The ENCODE/DECODE software interface is described in Appendix B.
Bit Manipulations

Because GENPRO is a data processing system, it goes without saying that utilities must be available on the host machine for manipulating data on the bit level such as shifting, masking, getting bit fields and storing bit fields. GENPRO uses a pair of functions for accessing and storing an arbitrary string of bits called GBYTE/SBYTE. Versions of these routines are available for the CRAY, DEC, IBM, CDC, and UNIVAC machines.

The GBYTE/SBYTE software interface is described in Appendix B.

Tape or Disk I/O

For sequential access of data from tape or a disk file, GENPRO uses read/write software called RDTAPE/WRTAPE. Versions of these routines are available for CRAY, CDC, IBM, and DEC machines.

The RDTAPE/WRTAPE software interface is described in Appendix B.

Random I/O

Some random file manipulations are useful within GENPRO for data management. A general interface to routines called BRANRD/BRANWT is provided. These routines are available for CRAY, CDC, IBM, and DEC machines.

The BRANRD/BRANWT software interface is described in Appendix B.

Miscellaneous

There are several other local software facilities which are assumed to be available. They include DATE and TIME functions, a random number generator (QDIN Operation), and all of the standard mathematical type routines such as SIN, COS, ALOG, etc.

SOURCE LANGUAGE

The GENPRO codes can exist in any of three language levels. The highest level is IFTRAN, a preprocessing structured language which creates FORTRAN 66 or FORTRAN 77 as output. The second level is standard FORTRAN which can be FORTRAN 66, or FORTRAN 77 with the holerith data extension. The lowest level code is object or binary code which is unique to the operating system on which it was generated. Users are discouraged from porting the code at this level.

A document describing the IFTRAN preprocessor is included as Appendix C.
GENPRO FILES Included on the tape used to port GENPRO are the following collection of files:

All files except 26 are in ASCII unless the target host is an IBM or IBM compatible machine. In that case the files will be in EBCDIC.

---

### Table 1. GENPRO System Files on Magnetic Tape

<table>
<thead>
<tr>
<th>FILE</th>
<th>NAME</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DRIVER</td>
<td>Source Code (IFTRAN or FORTRAN)</td>
</tr>
<tr>
<td>2</td>
<td>TRANSLATOR</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>3</td>
<td>LEX SCANNER</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>4</td>
<td>SUPPORT SUB LIB</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>5</td>
<td>RANDOM FILE SIM</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>6</td>
<td>CALIBRATION OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>7</td>
<td>DESPIKE OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>8</td>
<td>FILTER OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>9</td>
<td>INPUT OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>10</td>
<td>OUTPUT OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>11</td>
<td>PLOT OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>12</td>
<td>PRINT OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>13</td>
<td>SETRNG OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>14</td>
<td>SHIFT OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>15</td>
<td>SPECTRA OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>16</td>
<td>STATISTICS OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>17</td>
<td>INTERPOLATION OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>18</td>
<td>GAPFIL OPERATION</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>19</td>
<td>ERROR MESSAGE FILE</td>
<td>300 RECORDS OF 120 CHARACTERS EACH (ASCII)</td>
</tr>
<tr>
<td>20</td>
<td>PROGRAMMER DIRECTIVES</td>
<td>DATA (PROGRAM DIRECTIVES)</td>
</tr>
<tr>
<td></td>
<td>FOR ALL OPERATIONS (PDS)</td>
<td>(ASCII)</td>
</tr>
<tr>
<td>21</td>
<td>TUTORIAL TEST CASE 1</td>
<td>DATA (ASCII) PDS/UDS</td>
</tr>
<tr>
<td>22</td>
<td>TUTORIAL TEST CASE 2</td>
<td>DATA (ASCII) PDS/UDS</td>
</tr>
<tr>
<td>23</td>
<td>TUTORIAL TEST CASE 3</td>
<td>DATA (ASCII) PDS/UDS</td>
</tr>
<tr>
<td>24</td>
<td>TUTORIAL TEST CASE 4</td>
<td>DATA (ASCII) PDS/UDS</td>
</tr>
<tr>
<td>25</td>
<td>TUTORIAL TEST CASE 5</td>
<td>DATA (ASCII) PDS/UDS</td>
</tr>
<tr>
<td>26</td>
<td>METEOROLOGICAL DATA</td>
<td>BINARY DATA TO BE PROCESSED AS PART OF THE TUTORIALS</td>
</tr>
<tr>
<td>27</td>
<td>SYSTEM IMPLEMENTOR'S GUIDE</td>
<td>THIS DOCUMENT IN ASCII</td>
</tr>
</tbody>
</table>

* These files might not be available for the target host.
The GENPRO Operations all use a common error reporting process which links an error message number from 1 to 30 to the corresponding record in file 19 of Table 1. Arguments from the subroutine are inserted in place of $ symbols in the error record. In this way a single generic message can be used to report many different error conditions.

The error message file (19 in the above table) must be connected by host JCL to fortran unit 98, or the read from unit 98 in the COMERR Subroutine of the GENERAL SUB LIB must be changed to another appropriate unit (assuming your system does not provide unit 98).

A user directs the functioning of the GENPRO system through Directives. There are 2 types of Directives, Programmer Directives (PDS) and User Directives (UDS). A full set of Directives must be provided in order to execute each GENPRO Operation, and another set is required by the GENPRO DRIVER. Each Directive set must contain 1 and only 1 PD plus 1 or more UDS assembled in this order.

A full GENPRO Case can be executed by stacking sets of Directives. The set for the DRIVER must always come first, the set for some type of input Operation (read data to be processed) must come second, and then a sequence of arbitrary Operations can be performed in the order in which their associated Directive sets are stacked. For example, Table 2 shows a Case that includes reading and plotting meteorological variables from a tape (or disk file).
Table 2. Directives of a Typical GENPRO Case.

NOLIST
BEGINPD
/ XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/ X DRIVER OPERATION X
/ X PROGRAMMER DIRECTIVE FILE X
/ XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/
DEFGEN = NAMPD, NAMLIB
DIMGEN= 'NONE', 'NONE'
/
DEFGEN = BEGIV, ENDIV, DELIV
DIMGEN = (0.,0.,0.), (00.,01.,00.), 1.
/
DEFGEN = SCALIV, LABIV, IVDIM
DIMGEN = (3600.,60.,1.), (HRS,MNS,SEC), 3
/
DEFGEN = PROJECT
DIMGEN =
/
DEFGEN = PRTIME, PRDATE
DIMGEN = ('IIH','IIM','IIS'), ('DD','MON','YY')
/DIMGEN = ('IIH', 'IIM', 'IIS'), ('DD', 'MON', 'YY')
/
DEFGEN = NUMOPT, OVRFLO, GAPVAL, BITCHR, BITNSU, IABORT
DIMGEN = 1, -99999., 99999., 8, 64 , 4
/
DEFGEN = KCHECK, KDUMP, KERR, KFILM, KFICHE, KPRINT, KPROC
DIMGEN = 6, 4, 6, 4, 4, 6
/
DEFGEN = PRINT
DIMGEN = <(CASE),(RATE),(SNAP),(TIME),(SIZE)>
/
DEFGEN = DEBUG,
DIMGEN = <(NOSNAP),(NOFLOW),(NOCONTROLS)>, DUMP
DIMGEN = (0,1,1,1)
/
DEFGEN = NAMEOP, NUMCYC, ITYPOP
DIMGEN = INDR , 0 , 1
/
DEFGEN = BEGSNP, ENDSNP, FLUSHP
DIMGEN = (0.,0.,0.), (99.,99.,99. ), 900
/
DEFGEN = KUNIT , LIMREC, RECSIZ, DATLOG, NUMBIT, MAXCHR
DIMGEN = (9,10), 0 , 0 , 0 , 0 , 128
/
MODGEN = NAMPD , NAMLIB, NUMCYC,
BEGIV , ENDIV , DELIV , SCALIV , LABIV , IVDIM ,
PRTIME , PRDATE , PROJECT , PRINT , DEBUG , DUMP ,
NUMOPT, OVRFLO, GAPVAL, BITCHR, BITNSU, IABORT,
KCHECK, KDUMP, KERR, KFILM, KFICHE, KPRINT, KPROC,
BEGSNP, ENDSNP, FLUSHP,
KUNIT, LIMREC, RECSIZ, DATLOG, NUMBIT, MAXCHR
/
DEFVAR = NAMVAR,INDGET, INRATE, INDPUT, RATE
DIMVAR = NAMVAR, 0 , 0 , 0 , 1
MODVAR = RATE
/
ENDPD /***************************************************************
LIST
BEGINUD
/ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
/ X DRIVER OPERATION X
/ X USER DIRECTIVE FILE X
/ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
/
PROJECT='ALPEX P3 HIGH RES DATA (40HZ)'
PRDATE = ('15','APR','82')
PRTIME = ('08H','32M','45S')
BEGIN = (08.,32.,45.)
END = (14.,23.,40.)
BEGINP = (08.,32.,45.),(08.,45.,00.)
END = (08.,45.,00.),(14.,23.,40.)
FLUSH = 600
/
PRINT = (CASE),(RATE),(TIME),(SIZE),(STOP)
PRINT = (CASE),(TIME),(STOP)
DEBUG = (FLOW),(STOP)
IABORT = 5
KPROC = 6
KPRINT = 6
KERR = 6
KCHECK = 6
KDUMP = 6
ENDUD
NOLIST
BEGINPD
/ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
/ X INPUT OPERATION X
/ X PROGRAMER DIRECTIVE FILE X
/ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
/
DEFGEN = NAMEOP, NAMPD, NAMUD, NAMLIB, ITYPOP, NUMCYC
DIMGEN = INPUT, 'INPUT', 'INPUT', 'NONE', 1, -1
/
DEFGEN = CONV, FRTIM, IVTYPE, IVNAM
DIMGEN = 1. , 1.0 , 0 , NOLINK
LNKVAR=IVNAM
/
DEFGEN = JCL
DIMGEN = 'NO'
/
DEFGEN = KIN, KMODE, KPOS, KTYPE
DIMGEN = 20 , 1 , (0, 0), 1
/
DEFGEN = LOGBIT, DATLOG, BITKEY, DATSIZ, MEDIA, DATOPT
DIMGEN = 1000 , 1 , 0 , 10000 , 1 , (AUTO,NOSPAN)
DEFINE = NPHY, NSEC, NAMKEY, NWVOL
DIMGEN = 1, 0, NOLINK, 'EOFTAP';
LNKVAR=NAMKEY

DEFINE = NUMV, NEWV, NOLDV, NTOT, I, KSTOP, NUMLOG, IVPR, DATBIT
DIMGEN = 1, 0, 0, 0, 0, 0, 0, 0, 1

DEFINE = OFFSET, MAXIV, KEOF, KOUNT, IKOUNT
DIMGEN = 0.0, 86400., 1, 0, 0

MODGEN = NAMPD, NAMUD, NAMLIB, NUMCYC, CONV, FRTIM, IVTYPE, IVNAM, JCL, KIN, KMODE, KPOS, KTYPE, LOGBIT, DATLOG, BITKEY, NSEC, NAMKEY, NWVOL, DATSIZ, DATOPT, MEDIA, IVPR, DATBIT, OFFSET, MAXIV, KEOF, KOUNT

DEFVAR = NAMVAR, INDGET, INRATE, INDPUT
DIMVAR = NAMVAR, 0, 0, 0

DEFVAR = RATE, BITS, FSTBIT, SKIP, SAMPLE, CONKEY, SCLKEY, TERM, FACTOR
DIMVAR = 1, 16, 1, 0, 1, 1, 2, 0., 1.

DEFVAR = TITLE, UNITS
DIMVAR = 'TITLE', 'UNITS'

MODVAR = RATE, BITS, FSTBIT, SKIP, SAMPLE, CONKEY, SCLKEY, TERM, FACTOR, TITLE, UNITS

/ENDPD /************************************************************/

LIST
BEGINUD

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 X INPUT OPERATION X
 X USER DIRECTIVE FILE X
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ORDGEN = FRTIM, IVTYPE, IVPR
LETGEN = 1.0, 1, 0

ORDGEN = IVNAM, CONV, KEOF, KIN, KPOS, NWVOL, JCL
LETGEN = [TIME], [1.], 0, 13, (0,0), 'V00761', 'YES'

ORDGEN = LOGBIT, DATLOG, DATSIZ
LETGEN = 9344, 1, 9344

VECVAR = TIME, PMB, TASC, TASI, ALAT, ALON, HDNG, V1, U1, Z, YR, W, V, U, RHOV, T, TV, RADDWN

/EXTRACT THE DATA FIELDS FROM A PACKED BINARY TAPE

/ORDVAR = FSTBIT, BITS, SKIP, SAMPLE, CONKEY, TERM, FACTOR
LETVAR = 1, 32, 0, 1, 4, 0.0, 1.0000000, %FOR, TIME
LETVAR = 33, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, PMB
LETVAR = 65, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, TASC
LETVAR = 97, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, TASI

1-9
LETVAR = 129, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, ALAT
LETVAR = 161, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, ALON
LETVAR = 193, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, HDNG
LETVAR = 225, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, V1
LETVAR = 257, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, U1
LETVAR = 269, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, Z
LETVAR = 321, 32, 0, 1, 2, 0.0, 1000.0000, %FOR, YR
LETVAR = 385, 32, 0, 40, 2, 0.0, 1.E06, %FOR, W
LETVAR = 1665, 32, 0, 40, 2, 0.0, 1.E06, %FOR, V
LETVAR = 2945, 32, 0, 40, 2, 0.0, 1.E06, %FOR, U
LETVAR = 4225, 32, 0, 40, 2, 0.0, 1.E06, %FOR, RHOV
LETVAR = 5505, 32, 0, 40, 2, 0.0, 1.E06, %FOR, T
LETVAR = 6785, 32, 0, 40, 2, 0.0, 1.E06, %FOR, TV
LETVAR = 8065, 32, 0, 40, 2, 0.0, 1.E06, %FOR, RADDWN

RATE = 1, %FOR, TIME, PMB, TASC, TASI, ALAT, ALON, HDNG, V1, U1, Z, YR
RATE = 40, %FOR, W, V, U, RHOV, T, TV, RADDWN

SCLKEY = 2

ORDVAR = TITLE, UNITS /NAME
LETVAR = 'RAW INS LATITUDE', 'DEG ', %FOR, ALAT
LETVAR = 'RAW INS LONGITUDE', 'DEG ', %FOR, ALON
LETVAR = 'UNALTERED TAPE TIME', 'SECS ', %FOR, TIME
LETVAR = 'AIRCRAFT TRUE HEADING', 'DEG ', %FOR, HDNG
LETVAR = 'PRESSURE', 'MB ', %FOR, PMB
LETVAR = 'AIRCRAFT TRUE AIR SPEED', 'M ', %FOR, TASC, TASI
LETVAR = 'AMBIENT TEMPERATURE', 'DEG K ', %FOR, T
LETVAR = 'VIRTUAL TEMPERATURE', 'DEG K ', %FOR, TV
LETVAR = 'ALTITUDE', 'FEET ', %FOR, Z
LETVAR = 'YEAR', 'YEARS ', %FOR, YR
LETVAR = 'RADIOMETER DOWN', 'DEG C ', %FOR, RADDWN
LETVAR = 'WIND VECTOR EAST COMPONENT', 'M/SEC ', %FOR, U, U1
LETVAR = 'WIND VECTOR NORTH COMPONENT', 'M/SEC ', %FOR, V, V1
LETVAR = 'WIND VECTOR VERTICAL COMPONENT', 'M/SEC ', %FOR, W
LETVAR = 'WATER VAPOR DENSITY', 'G/M3 ', %FOR, RHOV

ENDUD
NOLIST
BEGINPD

DEFGEN = NAMEOP, NAMPD, NAMLIB, NUMCYC, ITYPOP
DIMGEN = PLOT, 'NONE', 'NONE', -1, 0

DEFGEN = PRINT
DIMGEN = '<(TABLE),(SUMMARY),(STOP),(CONTROLS),(IO)>

DEFGEN = BEGSNP, ENDSNP, FLUSHP
DIMGEN = (0., 0., 0.), (99., 99., 99.), 900

DEFGEN = PROJECT
DIMGEN = 2

DEFGEN = IVUNIT, IVTITLE
DIMGEN = 2
DEFGEN = KACCUM, RASTER, SIMPLE, LENARO, LENBRB
DIMGEN = (0,-1), (1024,1024.), MIX, 60, 60
/
DEFGEN = NCELLS, CSIZE, CAREA
DIMGEN = 15, (0.), (1.)
/
DEFGEN = OV1GRD, OV2GRD, OV3GRD, OV4XY, OV5XY, OV6XY
DIMGEN = (1.,1.), (1.,1.), (1.,1.), (1.,1.), (1.,1.), (1.,1.)
/
DEFGEN = ' OV1TXT', OV2TXT
DIMGEN =
/
DEFGEN = ' OV3TXT', OV4TXT
DIMGEN =
/
DEFGEN = ' OV5TXT', OV6TXT
DIMGEN =
/
DEFGEN = SIZTXT
DIMGEN = (2,2,2,2,2)
/
DEFVAR = NAMVAR, INRATE, INDGET, INDPUT, RATE
DIMVAR = NAMVAR, 0, 0, 0, 1
/
DEFVAR = ' TITLE', UNITS
DIMVAR =
/
DEFFUN = NAMFUN, PLCTYPE, PLTCYC, FRAME, SETCAL
DIMFUN = NAMFUN, XY, 0, YES, <(.15,.95),(.15,.95)> / (X,Y)
/
DEFFUN = IGRID, ISCALE, REVERSE, EQUAL, RNGTYP
DIMFUN = 5, 1, NONE, NO, (FULL,FULL) / (X,Y)
/
DEFFUN = BOTRNG, MAJMIN
DIMFUN = <(-99,-99),(-99,-99)>, <-(-99,-99),(-99,-99)> / (X,Y)
/
DEFFUN = REPLLOT, WINDOW, EXTREMA, SIGMAS, ADIBAT
DIMFUN = NO, NO, 0, 5.5, 0. / (X,Y)
/
DEFFUN = XAXIS, YAXIS, ARROWS, BRBTYPE, BRBVARS
DIMFUN = IV, NOLINK, NO, NONE, (NOLINK,NOLINK) / (U,V) OR (S,D)
/
DEFFUN = HSTYPE, PROBE, OVLAYS
DIMFUN = ACC, 1, (NOLINK,NOLINK) / (COORDS,TEXT) OR (COORDS,NOLINK)
/
MODGEN = BEGSNP, ENDSNP, FLUSHP, PROJECT, PRINT, IVUNIT, IVTITLE,
SIMPLE, KACCUM, RASTER, NUMCYC, LENARO, LENBRB, NCELLS,
CSIZE, CAREA, OV1GRD, OV2GRD, OV3GRD, OV4XY, OV5XY,
OV6XY, OV1TXT, OV2TXT, OV3TXT, OV4TXT, OV5TXT, OV6TXT, SIZTXT
/
MODVAR = RATE, TITLE, UNITS
/
1-11
MODFUN = PLTYPE, PLTCYC, FRAME, SETCAL, IGRID, ISCALE, REVERSE, EQUAL, RNGTYP, BOTRNG, MAJMIN, REPLOT, WINDOW, EXTREMA, SIGMAS, ADIBAT, XAXIS, YAXIS, ARROWS, BRBTYP, BRBVARS, HSTYPE, PROBE, OVLAYS

LNKGEN = OVLAYS
LNKVAR = XAXIS
LNKVAR = YAXIS
LNKVAR = BRBVARS

ENDPD
LIST
BEGINUD

PRINT=(SUMMARY),(STOP)
PROJECT = %D
BEGSNP = %D
ENDSNP = %D
FLUSHP = %D

VECVAR = %T

I AM PLOTTING A 1 PT PICKOFF (RATE=1). RATE = %T IS THE INPUT RATE.

RATE = 1
TITLE = %T
UNITS = %T

SIMPLE TIME PLOTS OF ALL VARIABLES

SIMPLE = ALL
ENDUD
ENDCASE
ENDJOB

PDS begin with the Directive, BEGINPD, and end with the Directive, ENDPD. UDS begin with the Directive, BEGINUD, and end with the Directive, ENDUD. GENPRO users should not be allowed to alter PDS. As a system administrator you should attempt to isolate the user as much as possible from even the awareness of their existence. UDS, on the other hand, are the process whereby the user makes his selection of which data to process, which variables are to be processed, the variable sample rates to be chosen, the functions (print, plot, filter, etc.) to be performed on the variables, and the options to be selected with respect to each function.

Each Case ends with the Directive ENDCASE, and each batch job ends with the Directive ENDJOB.

Normally all Directives such as tutorial files 21 through 25 of Table 1. are read from the standard fortran input (card reader, terminal) unit, 5.
All options available through User Directives are explained in the GENPRO User Guide.

**Input Data**

Scientific or engineering data to be processed is typically read using the INPUT Operation. This data is usually on magnetic tape or magnetic disk files.

This data is read from a FORTRAN unit selected by the user. It must lie in the range from 11 to 20. Refer to the KIN Control of the INPUT Operation in the GENPRO User Guide. Usually the user must provide host JCL which connects his tape or disk file to the FORTRAN unit number selected. See the JCL Control of the INPUT Operation in the GENPRO User Guide.

**Output Data**

Scientific or engineering data which has been processed by GENPRO can be written to magnetic tape or disk using the OUTPUT Operation.

This data is written onto a FORTRAN unit selected by the user. It must lie in the range from 21 to 30. Refer to the OUNIT Control of the OUTPUT Operation in the GENPRO User Guide. The user must provide host JCL which connects his tape or disk file to the FORTRAN unit number selected.

**Intermediate Data**

Data files are used internally to store program control information and to accumulate tabular or graphic displays.

On certain systems these files have been hidden from the user. On other systems they might need direct FORTRAN unit assignments.

In the case of the PRINT Operation, switching from a preassigned file through the BRANRD/BRANWT interface to direct disk unit assignments is handled through IFTRAN conditional compilation.

The PLOT and SPFFT (Spectra) Operations, on the other hand, provide a Control so the user can make this selection at run time. See the KACCUM Control for either Operation in the GENPRO User Guide.

If file 5 (RANDOM FILE SIM) of Table 1 is not available for your host machine you will either have to write one (see Appendix B for the interface), or make direct FORTRAN unit assignments which do not conflict with the other assignments of this section.

**Graphics Output**

Graphical output is generated by the PLOT and SPFFT Operations. The form of the output is metacode, as explained in the Graphics manual of Appendix A.
Unit assignment and storage of the metacode is a part of the system implementation of the NCAR Graphics as outlined in Appendix A.

Tabular Data Listing

The PRINT Operation provides the capability to generate grouped tabular displays of the variables being processed. This output is directed to a unit called KPRINT. The actual unit number to be used is selected in the UD of the DRIVER. See DRIVER Control KPRINT in the GENPRO Users Guide.

Options are available on most systems to send this output to either unit 6 (printer or terminal), a plotting device, or a disk or tape file.

Printouts

Standard GENPRO printouts include, error messages, diagnostics, job and data flow information, statistics, etc. The selection of FORTRAN unit numbers for various printouts is done in the DRIVER. See DRIVER Controls KERR, KCHECK, KDUMP, and KPROC in the GENPRO Users Guide.

Options are available on most systems to send this output to either unit 6 (printer or terminal), a plotting device, or a disk or tape file.

COMPILING

The GENPRO code is highly portable except for the system utilities described in Appendix B. It can be compiled under FORTRAN 66 or any FORTRAN 77 compiler which includes the holerith data extension.

If any compilation errors occur, please return a listing of these errors along with the name of the compiler used to:

Bob Lackman
National Center for Atmospheric Research
Box 3000
Boulder, Colorado 80307

LINKING

The DRIVER, General Support Library, Local System Math Library, and all compiled Operations selected from the available 13 must be linked along with compiled stub entries for the other available Operations. The stubs are needed because the OPCALL Subroutine of the DRIVER allows for calls to all existing GENPRO Operations.

LOADING and RUNNING

This part of the process is highly system dependent. Examples of a batch CRAY submission and a VAX command file are given.
CRAY COS JCL

**. READ PDS AND THE ERROR MESSAGE FILE**
**. PCOPY (/TB/GENPRO/REL02/ERRS,PDS ERRS,PDS)**
**. DELETE ANY PREVIOUS SET OF DIRECTIVES**
**. ACCESS,DN=G2DIR,ID=RL211,UQ,NA.**
**DELETE,DN=G2DIR,NA.**
**RELEASE,DN=G2DIR.**
**EDITOR.**
**. ACQUIRING THE GENPRO DRIVER AND OPERATION LIBRARIES**
**. ACQUIRE,DN=DRVLO4,MF=TB,ID=GENPRO,ED=1,W=GENPRO,M=GENPRO.**
**ACQUIRE,DN=OPRL04,MF=TB,ID=GENPRO,ED=1,W=GENPRO,M=GENPRO.**
**. ACQUIRING INPUT TAPE**
**ACQUIRE,DN=VO00761,DF=BB,MF=MT,^**
**TEXT='ERRMAX=9,EOTP=YES,NFILES=9,DEN=1600,BLKSIZE=12000'.**
**. ASSIGN INPUT TAPE TO UNIT 13**
**ASSIGN,DN=VO00761,A=FT13.**
**. READ DIRECTIVES FROM UNIT 5**
**. RELEASE,DN=$IN.**
**ASSIGN,DN=G2DIR,A=$IN.**
**ASSIGN,DN=G2DIR,A=FT05.**
**. ASSIGN OUTPUT TAPE TO UNIT 21**
**ASSIGN,DN=OUTPUT,A=FT21.**
**. TABULAR PRINTS TO UNIT 8**
**ASSIGN,DN=KPRINT,A=FT08,BFI=OFF.**
**. ASSIGN ERROR MESSAGES TO UNIT 98**
**ASSIGN,DN=ERRS,A=FT98,MR,BS=11.**
**. LOAD THE GENPRO LIBRARIES PRECEEDED BY ANY COMPILED CODES**
**LDR,DN=$BLD:DRVL04:OPRL04,MAP,L=MAPLIST,T=GENPRO,NA.**
**. THE FOLLOWING STATEMENT SENDS PLOTS TO DICOMED (FICHE)**
**NETDISP,DN=$PLT,DC=PT,DF=BI,MF=D1,NOACK,MDS.**
**DELETE,DN=G2DIR,NA.**

1-15
RELEASE,DN=G2DIR.
EXIT.
NETDISP,DN=$PLT,DC=PT,DF=BI,MF=D1,NOACK,MDS.
DELETE,DN=G2DIR,NA.
RELEASE,DN=G2DIR.
*
*. A stack of (PD,UD) sets are in the G2DIR Directive file.
*. EOF
EDIT,S=PDS,D=G2DIR
# readq G2DIR
EOD

The readq statement is a local NCAR command to read the file G2DIR consisting of the (PD,UD) sets of Directives. It contains a FETCH of the needed PD from the full set of PDS (file by that name), followed by a user created UD. Table 2 shows a typical Directive setup.

VAX Command File

On VMS systems a command file such as the following can be created to make unit assignments and initiate program execution:

$ ASSIGN/USER MODE G2.PLT IOP007
$ ASSIGN/USER MODE G2.PRT FOR008
$ ASSIGN/USER MODE G2.DIR FOR005
$ ASSIGN/USER MODE G2.OUT FOR006
$ ASSIGN/USER MODE G2.TAP IOP021
$ ASSIGN/USER MODE MFAO IOP013
$ ASSIGN/USER MODE MFAO LOGICAL MAGTAPE 00
$ ASSIGN/USER MODE ERR.DAT FOR098
$ DEFINE/USER MODE SYS$INPUT SYS$COMMAND:
$ RUN DRIVER
$ EXIT

This example command file assumes that the NCAR Graphics System with the IOP interface has been installed. In the example, metacode plots from the PLOT and SPFFT Operations are written on unit 7, tabular prints from the PRINT Operation are written on unit 8, Directives are input from unit 5, printouts go to unit 6, the data tape read by the INPUT Operation is assigned to unit 13, an output tape is written to unit 21 by the OUTPUT Operation, and the common error message file is connected to unit 98.

These assignments assume that the following Directives were used within the file called G2.DIR:
By far the most difficult aspect of running GENPRO is associated with extracting the proper variable fields from a data tape. If one of these fields is TIME, any error in the definition of the length, location, or scaling of the datum will result in a failure of the program to sync up to the requested run times set by the (BEGIV,ENDIV) Directives.

Therefore, whenever a new tape format is being decoded the Independent Variable chosen should be LOGICAL RECORD not TIME. Table 3 shows the Directives needed to switch between these two choices of Independent Variable.
Table 3. Independent Variable Control Set

<table>
<thead>
<tr>
<th>Operation(s)</th>
<th>LOGREC</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVER</td>
<td>BEGIV = (1.)</td>
<td>BEGIV = (08., 32., 45.)</td>
</tr>
<tr>
<td></td>
<td>ENDIV = (10.)</td>
<td>ENDIV = (08., 33., 00.)</td>
</tr>
<tr>
<td></td>
<td>SCALIV = (1.)</td>
<td>SCALIV = (3600., 60., 1.)</td>
</tr>
<tr>
<td></td>
<td>LABIV = (LOGREC)</td>
<td>LABIV = (HR, MIN, SEC)</td>
</tr>
<tr>
<td></td>
<td>IVDIM = 1</td>
<td>IVDIM = 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT</th>
<th>IVNAM is ignored</th>
<th>IVNAM = (HMS or TIME)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CONV is ignored</td>
<td>CONV value is ignored under</td>
</tr>
<tr>
<td></td>
<td>(no CONKEY for LOGREC)</td>
<td>CONKEY = 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT</th>
<th>IVTYPE=0</th>
<th>IVTYPE=1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FRTIME is ignored</td>
<td>FRTIME=1.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PLOT</th>
<th>IVUNIT=1</th>
<th>IVUNIT=2 (selects minute title</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>based upon LABIV = (HR, MIN, SEC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The 2nd of the 3 components</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRINT</th>
<th>VCOEF = (1000., 100., 100.)</th>
<th>VCOEF = (3600., 60., 60.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IVLAB = ('TH', 'HND', 'REC')</td>
<td>IVLAB = ('HR', 'MI', 'SEC')</td>
</tr>
</tbody>
</table>

| DRIVER       | BEGSNP = (1.)               | BEGSNP = (08., 32., 45.), (08., 45., 00.) |
| DESP         |                             |                                           |
| PRINT        |                             |                                           |
| PLOT         |                             | (Or any reasonable set of intervals)     |
| STATS        |                             |                                           |
| SPFFT        |                             |                                           |
| OUTPUT       |                             |                                           |
The Graphics System
Implementor's Guide

Editor: Gregory R. McArthur
In September of 1982, an article on additional information for the Implementor's Guide was published and distributed to all users in that month's issue of The Record. This information now appears at the end of this reprint of the manual.
# The Graphics System Implementor's Guide

## Table of Contents

### CHAPTER 1: SYSTEM PLOT PACKAGE PHILOSOPHY
- Overview .......................................................... 1-1
- Formation .......................................................... 1-1
- Transportation ..................................................... 1-2
- Enhancements ....................................................... 1-2

### CHAPTER 2: IMPLEMENTING THE SYSTEM PLOT PACKAGE ON A TARGET COMPUTER
- Acquisition ......................................................... 2-1
- Support ............................................................... 2-1
- Testing ............................................................... 2-1
- TEST.PLOT FRAME 1 .............................................. 2-2
- TEST.PLOT FRAME 2 .............................................. 2-3
- TEST.SSP PLOTS .................................................... 2-4

### CHAPTER 3: REQUIRED LOCALLY IMPLEMENTED SUPPORT Routines FOR TRANSPORTING THE NCAR SYSTEM PLOT PACKAGE
- Subroutine ULIBER .................................................. 3-1
- Subroutine WRITEB ................................................ 3-1
- Subroutine ENCODE ................................................ 3-1
- Subroutine GETCHR ............................................... 3-2
- Subroutine SETCHR ............................................... 3-2
- Subroutine PACKUM ............................................... 3-2
- Subroutine PERROR ............................................... 3-2
- Function LOC (X) .................................................. 3-2
- Function ISHIFT ................................................... 3-2
- Function IAND ...................................................... 3-3
- Function IOR ....................................................... 3-3
- Function INIT (X) .................................................. 3-3
- Function ILMACH (I) ............................................. 3-3
- Function RLMACH (I) ............................................. 3-5
- Routine Implementations .......................................... 3-6
- Routine Implementations (Cont) ................................ 3-7
- Routine Implementations (Cont) (Cont) ......................... 3-8

### CHAPTER 4: HOW TO WRITE A METACODE TRANSLATOR TO ADD A GRAPHICS DEVICE TO THE SYSTEM
# Table of Contents

## Introduction to Metacode
- Portable Translation ........................................ 4-1
- Metacode Instructions ........................................ 4-1
- Metacode Instructions (Con't) ............................... 4-3
- OP CODE Definitions ........................................... 4-4
- Paper Efficiency ................................................ 4-5

## CHAPTER 5: NCAR'S GRAPHICS METAFILE STRUCTURE
- Introduction ...................................................... 5-1
- Metafile Structure ............................................. 5-1
- Old vs. New Metafile Structure ............................... 5-2

## CHAPTER 6: UNDERSTANDING AN NCAR SYSTEM PLOT PACKAGE
- Purpose .......................................................... 6-1
- Formation ........................................................ 6-1
- Structure ........................................................ 6-1
  - Variables and Use Tables ................................. 6-2

## CHAPTER 7: PREPROCESSING TO CREATE A SYSTEM PLOT PACKAGE
- Introduction ...................................................... 7-1
- Deck Structure ................................................ 7-1
- Mods ............................................................... 7-1
- Examples ......................................................... 7-2

## CHAPTER 8: UNDERSTANDING THE MASTER SYSTEM PLOT PACKAGE
- Preprocessing ................................................... 8-1
- Parameters ....................................................... 8-1
  - Master System Plot Package Excerpts .................... 8-2
    - Comments About Flags .................................. 8-2
    - Comments About Sets and Macros ..................... 8-3
    - Sets and Macros Defined ............................... 8-6
    - Comments About Codes ................................ 8-9

## CHAPTER 9: SAMPLE PLOTS
| AUTOGRAPH | CONRAN | CONRAS | CONRAQ | CONREC | CONRECQCK | CONRECQSMTH | CONRECQSMTH | DASHCHR | DASHLINE | DASHSMITH | DASHSUPR | HAFTON | ISOSRF | ISOSRFHR | PWRTX | PWRTIT | PWRTI | PWRTIW | PWRTIW | PWRTJ | SCROLL | SRFACE | STRMLN | SUPMAP | THREEED | VELVCT | WINDOW |
|-----------|--------|--------|--------|--------|-----------|-------------|-------------|---------|---------|----------|---------|--------|--------|----------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
The system plot package is used to perform low-level graphics tasks, such as scaling, line drawing, and character drawing. It is written with the goal of being as fast and small as possible while constrained to provide adequate sophistication to support various scientific plotting requirements. It has been made portable so as to provide a common graphics interface on all computers at NCAR. This also allows this interface to be implemented at many other installations. The system plot package produces a high-level device-independent instruction set to support various plotters at NCAR. This instruction set can be interfaced to virtually all line-drawing plotters.

System plot packages are configured for various computers by using a preprocessor called FRED. The FRED preprocessor translates an extension of FORTRAN, called FRED into FORTRAN. The most important FRED features used in writing the master system plot package were macros, parameters, and conditional compilation. Included among the parameters in the master system plot package are vari-
ables describing the target machine (such as the number of bits in a character). Preprocessing the master system plot package with these variables properly set produces a system plot package for a given computer.

**TRANSPORTATION**

Many experts in the field of transporting software today define software as being portable if the ratio of the amount of work to implement the software from scratch on a new system to the amount of work to transport the software to the new system is a large number. By this standard, the system plot package is portable if the implementor obtains a version configured for the target machine. The transportation process consists of implementing the required support routines for bit manipulation and I/O, testing these routines with the portable test package, testing the plot package and support routines with the sample program, and implementing the device-independent code translator for some plotter. For a person familiar with the target computer who also has software which drives a plotter, this can generally be done with roughly two weeks effort.

**ENHANCEMENTS**

Once the system is working, efficiency can be upgraded by improving the required locally implemented routines, and flexibility can be added by expanding the translators. To reduce the core requirements and improve the running time of the system plot package, the support routines should be rewritten in tight machine language code. The routines which are used most often are, in order: INTT, IAND, ISHIFT, GETCHR, IOR, PACKUM, WRITEB, LOC, SETCHR, ENCODE, IMACH, and RMACH. The portable test program should be used to test the improved versions. As experience is gained with each translator, users will realize what improvements are needed to make optimum use of each plotter for various applications. Highly sophisticated translators can take several months to implement.
CHAPTER 2: IMPLEMENTING THE SYSTEM PLOT PACKAGE ON A TARGET COMPUTER

ACQUISITION

A system plot package must be configured with parts of the code tailored to match the hardware of the target computer. If the Computing Facility’s software request form is properly completed by someone outside NCAR who is requesting the system plot package, the code received will match the target computer. To generate at NCAR a system plot package configured for a particular computer, the master system plot package is preprocessed using FRED (see System Plot Package Philosophy, Preprocessing to Create a System Plot Package, and Understanding the Master System Plot Package in Part 2 of this manual; FRED write-up, Chapter 13 of the NCAR Software Support Library).

SUPPORT

To run a system plot package, 14 support routines must be implemented (see Required Locally Implemented Support Routines for Transporting the NCAR System Plot Package in Part 2 of this manual). The implementation of these routines must be tested using the portable test program (TEST.12 on PORTLIB at NCAR; this is automatically sent to anyone requesting the plot package via the software distribution procedure). Hereafter, the system plot package is assumed to include the tested support routines.

TESTING

There are two portable test programs which can be used to test the system plot package. One in TEST.PLOT and the other is TEST.SPP (both of these FORTRAN programs are on PORTLIB at NCAR and are automatically sent to anyone requesting the plot package through NCAR software distribution.) TEST.PLOT produces instructions for two sample pictures (see the sample pictures on the following pages) and, in the program documentation, provides hex and octal dumps of the instructions which should be created. TEST.PLOT should be run as a first test. The program TEST.SPP exercises every entry of the system plot package and produces printed messages as to the success of the tests as well as instructions for 27 plots. Examples of the correct plots are reproduced on the following pages.
NCAR PLOT PACKAGE TEST
DEMONSTRATION PLOT FOR PWRT
DEMONSTRATION PLOT FOR PLOTIT
DEMONSTRATION PLOT FOR LINE
DEMONSTRATION PLOT FOR CURVE
DEMONSTRATION PLOT FOR SET 2-9
IWP la
6
DI DIs I I - s L _ _
- I I -I -- I I- II -- s
II- I-I -P II IPI1 - IIPIYIP· ...

2-9
DEMONSTRATION PLOT FOR PERIM
DEMONSTRATION PLOT FOR HALIFAX
DEMONSTRATION PLOT FOR TİČK4
DEMONSTRATION PLOT FOR GRID
DEMONSTRATION PLOT FOR GRIDL
DEMONSTRATION PLOT FOR PERIML
DEMONSTRATION PLOT FOR LABMOD
DEMONSTRATION PLOT FOR POINT
DEMONSTRATION PLOT FOR POINTS
DEMONSTRATION PLOT FOR OPTN
IZE
SIZE 100
SIZE 90
SIZE 80
SIZE 70
SIZE 60
SIZE 50
SIZE 40
SIZE 30
SIZE 20
SIZE 10
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 0
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 1
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 2
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 4
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 5
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 6
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 8
DEMONSTRATION PLOT FOR GRIDAL

IGPH = 9
DEMONSTRATION PLOT FOR GRIDAL
IGPH = 10
DEMONSTRATION PLOT FOR FLASH1, FLASH2, FLASH3

FLASH SEQ. 0
FLASH SEQ. 1
FLASH SEQ. 2
FLASH SEQ. 3
FLASH SEQ. 4
FLASH SEQ. 5
FLASH SEQ. 6
FLASH SEQ. 7
FLASH SEQ. 8
FLASH SEQ. 9
FLASH SEQ. 10
DEMONSTRATION PLOT FOR FLASH4

FLASH4 TEST
CHAPTER 3: REQUIRED LOCALLY IMPLEMENTED SUPPORT Routines FOR TRANSPORTING THE NCAR SYSTEM PLOT PACKAGE

The following routines are needed by the portable system plot package.

SUBROUTINE ULIBER (IERR,MESS,LMESS)

Prints error message or prints error number and error message

IERR = error number (not printed if 0)
MESS = message to be printed including carriage control
LMESS = number of characters in MESS (≤ 130)

SUBROUTINE WRITEB (MBUFA,MBLEN,MUNIT)

Writes metacode to plotter or other unit which will serve as input to a metacode translator

MBUFA = variable containing the address of the buffer to be written (MBUFA = LOC(buffer))

MBLEN = number of buffer words that actually contain metacode data. If NCAR's standard metafile format is to be observed, WRITEB should do a fixed length binary write of 11,520 bits, regardless of the value of MBLEN. If MBLEN=0, a frame has been completed and the appropriate action is to put the data buffer out and backspace over it. In this way, the standard terminating record for a metafile will follow the last frame of the metafile.

MUNIT = unit number where instructions are to be written

SUBROUTINE ENCODE (NCHARS,IFMT,IRESLT,VAR)

The NCHARS characters that would result from writing VAR using format IFMT are stored in IRESLT. Thus,

CALL ENCODE(4,6H(F4.2),IRESLT,1.)

would result in IRESLT containing the 4 character string "1.0", left justified. The system plot package and NCAR utilities always call ENCODE with an E, F, G, or I immediately following the (, so it is possible to implement ENCODE without a full-blown FORMAT cracker.
SUBROUTINE GETCHR (ICHARS, LEN, N, JCHAR)

Extracts character number N from character string ICHARS which is LEN characters long and returns it in JCHAR, right-justified with zero fill. If N is less than or equal to zero or greater than LEN, GETCHR should return the right-justified, zero-filled representation for a blank.

SUBROUTINE SETCHR (ICHARS, LEN, N, JCHAR)

Takes a right-justified, zero-filled character stored in JCHAR and puts it in character position N of character string ICHARS which is LEN characters long. The rest of JCHAR is unchanged. If N is less than or equal to zero or greater than LEN, SETCHR should do nothing.

SUBROUTINE PACKM (MBPRS, NMBPRS, MBUFA)

Packs 16-bit bytes into a buffer

MBPRS = an array containing 16-bit bytes, one per word, right-justified, zero-filled

NMBPRS = number of such bytes. NMBPRS is always a number such that 16*NMBPRS bits fit exactly in an integral number of words

MBUFA = a variable containing the address of the start of the words into which the packed instructions are to be put. The bytes are not put in MBUFA; they are put starting at the location pointed to by MBUFA.

SUBROUTINE ERROR

A fatal error has occurred in the system plot package. This routine should implement some graceful error exit.

FUNCTION LOC (X)

Address of X relative to the start of the program. If I and J are default-size integers stored in sequential locations in memory, then LOC(J) − LOC(I) = 1. Similarly, if M is a default size-dimensioned integer variable, then LOC(M(2)) − LOC(M(1)) = 1. LOC is an intrinsic function on NCAR's 7600 system and is, therefore, not included in the listings that follow.

FUNCTION ISHIFT (IWORD, N)

IWORD shifted by N bits. If N > 0, a left circular shift (ROTATE); if N < 0, a right end off shift. |N| < word length

FUNCTION IAND (K1, K2)

The bit by bit logical product of K1 and K2

Each K2 bit
Each IAND 0 1
K1 0 0 0
bit 1 0:1

FUNCTION IOR (K1,K)

The bit by bit logical sum of K1 and K2
Each K2 bit

Each IAND 0 1
K1 0 0 1
bit 1 0:1

FUNCTION INTT (X) .TRUE. if X is an integer (Unless the user makes an error, the integers will be positive and $< 2^{15}$, and unless the user changes the plotter resolution with SETIT, the integers will be positive and $< 1024$.)

.FALSE. Otherwise

If a floating point zero cannot be distinguished from an integer zero, .FALSE. should be returned for zero.

On machines for which it is difficult or impossible to implement the function INTT, it is possible to eliminate it with a little work and with the sacrifice of some flexibility.

The function INTT can be left out of the package provided each plotting coordinate passed to a plot package entry is typed as a floating point or integer variable. The subroutine SETCOND (see The System Plot Package section of the Graphics Manual for documentation) can be used for this purpose. Two SETCOND keywords exist for specifying plotting coordinate types: XT (for XType) and YT (for YType). The value "2" indicates plotting coordinates will be integers, and the value "3" indicates plotting coordinates will be floating point numbers. For example:

CALL SETCOND (2XT, 2)
CALL SETCOND (2YT, 3)

would specify that subsequent X-coordinates will be integers and Y-coordinates will be floating point. The specifications for XT and YT remain in effect until SETCOND is called to reset them.

3-3
If one were willing to always restrict himself to floating point coordinate specifications, then the calls

CALL SETCND (2HXT,3)
CALL SETCND (2HYT,3)

at the beginning of the program would be the only required calls to SETCND.

At the present time, the higher level FORTRAN utilities (described in the Graphics Utilities section of this manual) do not specify the coordinate types of arguments to system plot package entries (this is where INIT is used in fact). In order to use a utility in a plot package implementation which does not have an INIT routine, it will be necessary to search that utility for calls to plot package entries, and invoke the appropriate SETCND call at those points.

**FUNCTION IIMACH (I)**

This function is used to set up 16 machine constants:

1. IIMACH(1) = the standard input unit
2. IIMACH(2) = the standard output unit
3. IIMACH(3) = the standard punch unit
4. IIMACH(4) = the standard error message unit
5. IIMACH(5) = the number of bits per integer storage unit
6. IIMACH(6) = the number of characters per integer storage unit

Assume integers are represented in the S-digit, base-A form

\[ \text{SIGN}(X(S-1)A^{(S-1)} + \ldots + X(1)A^0 + X(0)) \]

where 0 < X(I) < A for I = 0, ..., S-1.

1. IIMACH(7) = A, the base
2. IIMACH(8) = S, the number of base-A digits
3. IIMACH(9) = A**S-1, the largest magnitude

Assume floating point numbers are represented in the T-digit, base-B form

\[ \text{SIGN}(B^E)(X(1)/B + \ldots + (X(T)/B^T)) \]

where 0 < X(1), and EMIN.LE.E.LE.EMAX

4. IIMACH(10) = B, the base
Single precision:

I1MACH(11) = T, the number of base-B digits
I1MACH(12) = EMIN, the smallest exponent E
I1MACH(13) = EMAX, the largest exponent E

Double precision:

I1MACH(14) = T, the number of base-B digits
I1MACH(15) = EMIN, the smallest exponent E
I1MACH(16) = EMAX, the largest exponent E

**FUNCTION R1MACH (I)**

This function sets 5 single-precision machine constants.

R1MACH(1) = B**(EMIN-1), the smallest positive magnitude
R1MACH(2) = B**EMAX*(1-B**(-T)), the largest magnitude
R1MACH(3) = B**(-T), the smallest relative spacing
R1MACH(4) = B**(1-T), the largest relative spacing
R1MACH(5) = LOG10(B)

The following are implementations of these routines using NCAR system-dependent features. LOC is an inline function at NCAR, so it is not listed here.
******** THESE ROUTINES ARE NOT PORTABLE ********

SUBROUTINE LIBER(I ERR, MESS, LMESS)
DIMENSION MESS(1)
IF (I ERR .NE. 0) WRITE(6,1) I ERR
NWORDS = (LMESS - 1) / 10 + 1
WRITE(6,2) (MESS(I), I = 1, NWORDS)
RETURN
1 FORMAT(6H01 I ERR =, 15)
2 FORMAT(13A10)
END

SUBROUTINE WRITEB(MBUFA, MBLEN, MUNIT)
DIMENSION IDUMMY(1)
C 11520 BITS IS 192 60-BIT WORDS
C DATA NWORDS/192/
C ISTART = MBUFA - LOC(IDUMMY) + 1
IEND = ISTART + NWORDS - 1
BUFFER OUT(MUNIT, 1) (IDUMMY(ISTART), IDUMMY(IEND))
1 IF (UNIT, MUNIT) 1, 2, 3, 3
2 IF (MBLEN .EQ. 0) BACKSPACE MUNIT
RETURN
3 CALL ULIBER(0, 32HOPLOT PACKAGE BUFFER WRITE ERROR, 32)
STOP
END

SUBROUTINE ENCODE(NCHARS, IFMT, IRESLT, VAR)
DIMENSION IFMT(1)
ENCODE(NCHARS, IFMT, IRESLT) VAR
RETURN
END

SUBROUTINE GETCHR(ICHARS, LEN, N, JCHAR)
DIMENSION ICHARS(1)
IF (N, LT. 1, OR. N, GT LEN) GO TO 1
IWORD = (N - 1) / 10 + 1
IPOS = -(IWORD - 1) * 10
JCHAR = IAND(ISHIFT(ICHARS(IWORD), IPOS = 6-60), 77B)
RETURN
1 JCHAR = 55B
RETURN
END

SUBROUTINE SETCHR(ICHARS, LEN, N, JCHAR)
DIMENSION ICHARS(1)
IF (N, LT. 1, OR. N, GT LEN) RETURN
IWORD = (N - 1) / 10 + 1
IPOS = -(IWORD - 1) * 10
ICHARS(IWORD) = ISHIFT(1 OR JCHAR, IAND(I SHIFT(ICHARS(IWORD), 6 = IPOS),
C -77B)), 10 - IPOS + 6)
RETURN
END
SUBROUTINE PACKUM(MBPRS, NMBPRS, MBUFA)
DIMENSION IDUMMY(1), MBPRS(NMBPRS)
ISUB=MBUFA-LOC(IDUMMY)+1
CALL SBYTES('DUMMY(ISUB), MBPRS, 0, 16, 0, NMBPRS)
RETURN
END

SUBROUTINE PERROR
DATA IENTRY/0/
IF(IENTRY.NE.0) GO TO 2
IENTRY=1
CALL FLUSH
WRITE(6,3)
WRITE(6,4)
1 STOP
2 WRITE(6,3)
WRITE(6,5)
WRITE(6,4)
GO TO 1
3 FORMAT(2SM/FATAL ERROR IN PLOT PACKAGE)
4 FORMAT(32M IGNORE THE FOLLOWING ERROR TYPE)
5 FORMAT(23M COULD NOT FLUSH BUFFER)
END

FUNCTION ISMIFT(IWORD, N)
ISMIFT=IFT(IWORD, N)
RETURN
END

FUNCTION IAND(K1, K2)
IAND=AND(K1, K2)
RETURN
END

FUNCTION IOR(K1, K2)
IOR=OR(K1, K2)
RETURN
END

FUNCTION INTT(K)
LOGICAL INTT
DATA MASK/37770000000000000B/
INTT=IAND(IABS(K), MASK).EQ.0
IF(K.EQ.0) INTT=FALSE.
RETURN
END

C LOC IS AN INTRINSIC FUNCTION ON THE NCAR SYSTEM.
INTEGER FUNCTION IMACH(I)
INTEGER IMACH(16)
C MACHINE CONSTANTS FOR THE CDC 6000/7000 SERIES
DATA IMACH( 1) / 5 /
DATA IMACH( 2) / 6 /
DATA IMACH( 3) / 3581 /
DATA IMACH( 4) / 6 /
DATA IMACH( 5) / 60 /
DATA IMACH( 6) / 10 /
DATA IMACH( 7) / 2 /
DATA IMACH( 8) / 48 /
DATA IMACH( 9) / 00007777777777777777B/
DATA IMACH(10) / 2 /
DATA IMACH(11) / 48 /
DATA IMACH(12) / -974 /
DATA IMACH(13) / 1070 /
DATA IMACH(14) / 96 /
DATA IMACH(15) / -927 /
DATA IMACH(16) / 1070 /
IMACH = IMACH(1)
RETURN
END

REAL FUNCTION RMACH(I)
REAL RMACH(5)
DATA RMACH(1) / 0001400000000000000000B /
DATA RMACH(2) / 37767777777777777777B /
DATA RMACH(3) / 1640400000000000000000 B /
DATA RMACH(4) / 1641400000000000000000 B /
DATA RMACH(5) / 17164642023241175720 B /
RMACH = RMACH(1)
RETURN
END
INTRODUCTION TO METACODE

The device-independent instruction set (metacode) can be considered to be the hardware instructions for an imaginary plotter with extensive and extendable capabilities. This plotter has $2^{15}$ (or 32,768) addressable positions along each axis, multiple intensities and spot sizes, color, hardware characters in various sizes and fonts, and can have other required capabilities added on. The instruction set is 8-bit-byte-oriented and is more or less dictated by the nature of most minicomputers that will be used for the metacode translation.

PORTABLE TRANSLATION

A portable translator for translating NCAR metacode is available. This translator makes some efficiency sacrifices to achieve other goals. Because it is portable, the translator is easily moved from one host computer to another. Because it can reduce high-level metacode constructs to low-level ones, the least sophisticated plotters can be supported. Because the code has clearly marked interfaced points, a new plotter can be added to the system with a small effort.

METACODE INSTRUCTIONS

There are three types of instructions (see figure, next page). First is a four-byte positioning instruction to control pen movement containing a coordinate pair and a pen control bit. The next is a multibyte instruction, containing an operation code, the number of bytes to follow, and a character string. The last type of instruction is a two-byte increment instruction, for those users concerned with code compaction or very high resolution.

The positioning instruction is the basic line drawing instruction for the abstract plotter, reflecting the full precision of the device. This resolution corresponds to a picture size of almost three feet square with a resolution of one thousandth of an inch. The pen control bit position was chosen to facilitate translation on machines with short word lengths.

The multibyte instructions are used to control functions of the abstract plotter other than pen motion. Sixty-four operation codes are possible; half are reserved for use by the plot package designer, half are user-definable. The range of user-defined option codes is quite large, including picture descriptors, code compaction, links with interactive equipment, and metacode translator control. Users are, of course, required to add routines to the translator to react to these instructions; otherwise, they should be ignored.
The increment instruction can be used for code compaction, achieving up to a two-to-one compaction of the instructions for curves. (Other compaction methods are available through the multibyte instruction.)

**4 BYTE INSTRUCTION (ABSOLUTE POSITION)**

```
<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th></th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>Y</td>
</tr>
</tbody>
</table>
```

**MULTIBYTE INSTRUCTION**

```
<table>
<thead>
<tr>
<th>2</th>
<th>6</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>OPCODE</td>
<td>N</td>
<td>BYTE 1</td>
</tr>
</tbody>
</table>
```

**2 BYTE INSTRUCTION (RELATIVE POSITION; subtract 2^5 from X and Y)**

```
<table>
<thead>
<tr>
<th>2</th>
<th>6</th>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
```

Postprocessors for various plotters should be easy to write because of the limited form of the metacode input. No complicated utilities, such as axis drawing or scaling, need to be written for noncompacted metacode. For plotters where system plot packages already exist, a translator can be written using some of the existing code with minor modifications.

Authors of codes to translate metacode into machine instructions for a given plotter are strongly encouraged to parameterize operation codes (op codes), as these may be changed as experience is gained. Changes in instruction format are considered extremely unlikely.

†NOTE: In multibyte instructions, if N is odd, an extra byte is added to the end of the character string so all instructions are a multiple of 16 bits in length. This byte should be ignored.
Extract MXINC and MYINC. Subtract 32 from each and add their scaled* values to MX and MY. Also extract OP.

Extract OPCODE and N from 16 bits

Get next 16 bits; first 8 ➔ OPN Get option value and save. See OPCODE definitions

Store bits in MX. Get next 16 bits. First bit ➔ OP 15 bits ➔ MY Scale* MX and MY

Pen up*(MX, MY)

Pen down†(MX, MY)

Extract MXINC and MYINC. Subtract 32 from each and add their scaled* values to MX and MY. Also extract OP.

N (N is even) characters to plot; start at (MX, MY) with orientation IOR and width ISIZE. Get 16 bits N/2 times and plot them.

Advance frame, move paper and reset origin, or whatever to start new picture.

* 0 ≤ MX, MY ≤ 32,767. Scale these numbers to fit particular plotter used.

† It is often possible to use (a stripped down version of) the existing (locally written or manufacturer's) plot package to do these functions.
A pen control bit is found in the 2-byte instruction and the 4-byte instructions. An op code appears in the multibyte instructions. If the multibyte instruction is setting an option, the first byte of the character string tells which option is being set.

**PEN CONTROL BIT**
0 Pen up  
1 Pen down

**OP CODE**

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>PWRIT call. The N characters to be plotted are in Byte 1 through Byte N. The location of the center of the first character is determined by the most recent pen position and centering option call.</td>
</tr>
<tr>
<td>34</td>
<td>FRAME advance. N is 0.</td>
</tr>
<tr>
<td>35</td>
<td>OPTN Call</td>
</tr>
</tbody>
</table>

**Byte1** specifies which option is being set  
**Value found in**

<table>
<thead>
<tr>
<th>Option</th>
<th>Value Found In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case (0=upper, 1=lower)</td>
<td>Byte 2</td>
</tr>
<tr>
<td>Intensity (0 through 255)</td>
<td>Byte 2</td>
</tr>
<tr>
<td>Orientation of characters in degrees</td>
<td>Bytes 3 &amp; 4</td>
</tr>
<tr>
<td>Character width</td>
<td>Bytes 3 &amp; 4</td>
</tr>
<tr>
<td>Font</td>
<td>Byte 2</td>
</tr>
<tr>
<td>Dash pattern (16 bits)</td>
<td>Bytes 3 &amp; 4</td>
</tr>
<tr>
<td>Spot size (0 through 32767)</td>
<td>Bytes 3 &amp; 4</td>
</tr>
<tr>
<td>Centering option</td>
<td>Byte 2</td>
</tr>
<tr>
<td>(0=left, 1=center, 2=right)</td>
<td>Bytes 2, 3, &amp; 4</td>
</tr>
<tr>
<td>Color</td>
<td></td>
</tr>
</tbody>
</table>

† Exact meanings of some options will be established when plotters are available at NCAR with these capabilities.

**Range information**

<table>
<thead>
<tr>
<th>MINMX</th>
<th>MINMY</th>
<th>MAXMX</th>
<th>MAXMY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 1 &amp; 2</td>
<td>3 &amp; 4</td>
<td>5 &amp; 6</td>
<td>7 &amp; 8</td>
</tr>
</tbody>
</table>

**Points Mode**, N is 2. If the byte 1 is 0, turn points mode off. If byte 1 is 1, turn points mode one with a simple point as the marker. If byte 1 is 2, turn points mode one with the character in byte 2 as the marker. When in points mode, the current marker is to be put at the pen position.
resulting from the normal execution of all normal 2-byte and 4-byte pen-up move and pen-down draw instructions.

**PAPER EFFICIENCY**

Besides OP CODEs 33, 34, and 35, most NCAR system plot packages produce a multibyte instruction with OP CODE=39. The 8 bytes following contain the approximate range of all plotting for the preceding frame (MXMIN, MYMIN, MXMAX, MYMAX). This can be very useful for minimizing paper waste on drum plotters. On other plotters, this instruction can be ignored. The location of this instruction is described in the next chapter NCAR's Graphics Metafile Structure of this manual. NCAR-supported metacode translators and portable translator shells contain proper logic to use this information if the ISIZE=3 option is set in the main program of the translator (and assuming the two READ statements in routine GETRNG are appropriate for the host machine).
This section describes the structure of a standard NCAR graphics metafile. Such a file is a stream of data records containing metacode instructions as put out by any of the supported metacode plot packages and as expected on input by any supported metacode translator or other metacode manipulator. The metacode instruction set itself is described in other documents in this manual.

All records in the metafile have a fixed length of 11,520 bits. This is equivalent to 1440 8-bit bytes or 720 16-bit words, and is an integral multiple of the word length of any available computer.

The first 32 bits of each record are reserved for control information, and are divided into two 16-bit fields. The first field is the count of 8-bit metacode data bytes in the record. The count does not include the 32 bits of control information. The first 4 bits of the second 16-bit field are a data type identifier. A binary value of 0010 for this flag identifies the record as containing metacode. The next single bit is used to indicate frame boundaries. The bit is set to indicate that the record is the first data record of a new frame. The remaining 11 bits of the second 16-bit field are unused at the present time. Following the record control information are the metacode data.

Frames must begin on record boundaries. If the data do not fill the physical record, as will often be the case with the last data record of a frame, then the contents of the remaining space in the record are undefined. The last few metacode instructions of the last data record of a frame will typically be 16-bit no-op instructions (A020 in hex, or 120040 in octal). These will be included in the data count for the record; the number present will depend on the graphics image itself and the word size of the computer which generates the metafile. If the metafile includes a range instruction for each frame, it will immediately precede any no-op instructions at the end of a frame. The range instruction (which is 10 bytes long) will never cross record boundaries, hence, it will always be found in the last data record for a frame. There may (with a probability of less than 1%) be a few no-op instructions preceding the range instruction. The frame instruction itself will immediately precede whatever combination of range instruction and no-ops may be present.

A normal metafile; i.e., one resulting from a job that terminated normally, should have a single final record
with no data in it and zero in the byte count field. This allows multiple metafiles to be in a single data file ("file" in the FORTRAN sense), thus skirtng the portability problems raised by the lack of the multifile dataset concept in FORTRAN 66. In order that metafiles from abnormally terminated jobs may be successfully processed, an end-of-file mark should be considered synonymous with the special terminating record.

**OLD vs. NEW METAFILE STRUCTURE**

For users familiar with NCAR'S original metafile structure, and possibly maintaining software based on that structure, the following differences should be noted. The old structure was based on record pairs, with the first record of each pair being a 16-bit (at least) byte count record, and the second being the accompanying variable length data record. These two records are now merged into a single fixed length record, the first 16 bits of which is the data byte count. The notion of a "zero-byte-count record" to partition frames is replaced by defining one of the second 16 bits of each record as a frame boundary flag. A single "zero-byte-count record" now terminates the metafile, whereas two consecutive such records did so previously. Finally, if the metafile contains a range instruction for each frame, it will now be found immediately following the frame instruction that terminates a given frame, instead of occupying a record by itself at the very end of the frame.
CHAPTER 6: UNDERSTANDING AN NCAR SYSTEM PLOT PACKAGE

PURPOSE

This section is intended to help programmers understand the code in an NCAR System Plot Package (NSPP). The term "an NSPP" is used because there are many versions configured for various levels of output code compaction. The output code is called "metacode," a device-independent plotter instruction set which is fully described in How to Write a Metacode Translator to Add a Graphics Device to the System.

An NSPP contains the low-level routines used by NCAR graphics utilities (contouring, etc.) to draw lines and characters on various plotters. The most complex task performed by an NSPP is the drawing of axes with numerical labels at the tick marks or grid lines. Any tasks which are more complex are done by utilities.

FORMATION

An NSPP is produced for a given machine architecture by specifying character and word sizes which are used in preprocessing a master system plot package written in FRED. The output of the preprocessor is a particular system plot package in FORTRAN. Thus, at NCAR, when errors are to be fixed, the changes are made in the master system plot package and the various system plot packages are reformed by using the preprocessor.

STRUCTURE

It is the nature of system plot packages that they do not flow a great deal; looping and conditional structures are seldom used. Rather, routines are entered, their arguments are examined, and entries made in storage local to the plot package. Thus, understanding an NSPP is based on understanding the arguments to the various routines and understanding the various variables, tables, and buffers of the system plot package. The routine arguments are described in the user documentation, The System Plot Package.

NSPP routines store all information needed for subsequent processing in a named common block. Local variables of a given routine are not used to store information needed in later calls. The routines are serially reusable but not reentrant. (Local variables are used for DO-loop indices, etc.) The following table describes the variables that may be in the common block. Not all NSPPs contain all the variables. All integer variables in the common block start with M and no other variables do. Except for SMALL, all real variables in the common block start with X or Y.
<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMAJX, MMAJY, MMINX, MMINY, MXLAB, MYLAB, MFLG</td>
<td>GRIDAL arguments stored here so they will be in a known order for insertion in the instruction stream ONLY when ultracompact metacode is being produced.</td>
</tr>
<tr>
<td>MTYPE</td>
<td>Scaling type of the most recent SET call.</td>
</tr>
<tr>
<td>MX, MY</td>
<td>Plotter address of the pen location.</td>
</tr>
<tr>
<td>MXA, MYA, MXB, MYB</td>
<td>Plotter address corresponding to the first four arguments of the most recent SET call.</td>
</tr>
<tr>
<td>MTYPEX, MTYPEY</td>
<td>A decoding of MTYPE—0 = LINEAR, 1 = LOG</td>
</tr>
<tr>
<td>XXA, YYA, XXB, YYB, XXC, YYC, XXD, YYD</td>
<td>Exact copies of the first eight parameters of the most recent SET call.</td>
</tr>
<tr>
<td>XFACTR, YFACTR, XADD, YADD</td>
<td>Numbers computed from the most recent SET call arguments so that real valued coordinates can be translated to integers by [ MX = XFACTR \times XX + XADD ] or [ MX = XFACTR \times \text{ALOG10}(XX) + XADD ] and similarly for Y.</td>
</tr>
<tr>
<td>XX, YY</td>
<td>Most recent coordinate input to the PLOT package.</td>
</tr>
<tr>
<td>MFMTX, MFMTY, MUMX, MUMY, MSIZX, MSIZY, MXDEC, MYDEC, MXOR</td>
<td>Most recent LABMOD inputs except that MXDEC = 0 and MYDEC = 0 are decoded and MXDEC = 1 and MYDEC = 1 become 0.</td>
</tr>
<tr>
<td>MOP(I), MNAME(I)</td>
<td>Option names are given in MNAME and their current values in MOP</td>
</tr>
<tr>
<td>MXOLD, MYOLD, MXMAX, MYMAX, MXFAC, MYFAC</td>
<td>All used for increment instructions only. MXOLD and MYOLD are the plotter coordinates of the previous point, MXMAX and MYMAX are the greatest distance an increment can move, and MXFAC and MYFAC are the number of plotter units per increment unit (generally 1, but can be more if compaction is important and high resolution is not).</td>
</tr>
</tbody>
</table>
MODEF
= 0 Flash routines have not been used
= 1 Most recent FLASH call was to FLASH1
   (we are between FLASH1 and FLASH2 calls
   and the instructions should be put in the
   user's buffer)
= 2 FLASH1 call has been closed with a
   FLASH2 call
= 3 FLASH3 has been entered, but not exited,
   i.e., FLASH3 is dumping a user buffer.
= 4 Most recent FLASH activity is a completed
   FLASH3 call.

MTABLE(64) Translation table to change 6-bit characters
to 8-bit characters (only on 6 bit-per-character machines).

MF2ER = 0 no FLASH buffer overflow.
= N counts the number of times the buffer
   was reused so the required size can be
   estimated.

MSHFTX, MSHFTY The power of two of the ratio between the
resolution of the metacode address and the
resolution the user is working in. In the
default case, the user assumes the plotter
is 1024 by 1024 (1024 = 2^10). Metacode
addresses have 15 bits, so their capacity is
32,768. Thus, the default for MSHFTX AND MSHFTY
is 5, and user integer coordinates are left
shifted 5 to make plotter addresses.

MMGRX, MMGRY, MMNRX,
MMNRY Tick mark lengths (positive values point in).

MaskD, MaskOp, MPWrit For dd80 plot package only. MaskD contains the
various dd80 instruction formats. MaskOp contains
masks to set hardware options. MPWrit contains
masks for different numbers of trailing escape
characters for character strings (which must be a
multiple of 6 in length).
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCROUT</td>
<td>Number of metacode records that have been put out via PREOUT.</td>
</tr>
<tr>
<td>MFLCNT</td>
<td>Used to count the number of FLUSHB calls since last MBPRS initialization. It is used to avoid empty records which could otherwise be put out.</td>
</tr>
<tr>
<td>MFREND</td>
<td>FRAME sets to 1 to indicate last output call of a FRAME, and resets to zero before returning.</td>
</tr>
<tr>
<td>MFRLST</td>
<td>PREOUT manipulates, based on MFREND, so that it knows when a record is the first of a new frame.</td>
</tr>
<tr>
<td>MJXMIN, MJYMIN, MJXMAX, MJYMAX</td>
<td>Used to keep track of the range of the plotting address on the frame being created.</td>
</tr>
<tr>
<td>MNXSTO, MNYSTO, MXXSTO, MXYSTO</td>
<td>Used to hold MJXMIN,... after FLASH1 call, and restore them after FLASH2. MJXMIN,... are accumulated anew during FLASH saving, and stored in user FLASH buffer after FLASH2 call.</td>
</tr>
<tr>
<td>MPAIR1, MPAIR2</td>
<td>Two 16-bit pairs used to initialize each output record, so that PREOUT may format first 32 bits. They are actually put into MBPRS at proper times.</td>
</tr>
<tr>
<td>MPRINT</td>
<td>Unit number for printing error messages too extensive to be handled by ULIBER.</td>
</tr>
<tr>
<td>MSYBUF</td>
<td>Buffer to hold up to a few hundred metacode instructions.</td>
</tr>
<tr>
<td>MSBLEN</td>
<td>Word length of MSYBUF.</td>
</tr>
<tr>
<td>MNCPW</td>
<td>The number of characters per word on the host computer.</td>
</tr>
<tr>
<td>MINST</td>
<td>Holds instruction OP-CODE for the instruction being formed.</td>
</tr>
<tr>
<td>MBUFA</td>
<td>Contains the address of the buffer for the metacode instructions, either LOC(MSYBUF) or LOC(USR BUFFER) from a FLASH1 call.</td>
</tr>
<tr>
<td><strong>MBUFLU</strong></td>
<td>The number of words of the buffer pointed to by MBUFA that have been filled with metacode or dd80 instructions.</td>
</tr>
<tr>
<td><strong>MFWA,MLWA</strong></td>
<td>Contains the first word address and the last word address for the FLASH buffers.</td>
</tr>
<tr>
<td><strong>MIPAIR,MBPRS</strong></td>
<td>MBPRS is used to store byte pairs of metacode until they can be packed in an integral number of words and placed in the buffer pointed to by MBUFA. MIPAIR tells how much of MBPRS has been used.</td>
</tr>
<tr>
<td><strong>MBUFL</strong></td>
<td>The length of the buffer pointed to by MBUFA.</td>
</tr>
<tr>
<td><strong>MUNIT</strong></td>
<td>Unit number for writing metacode.</td>
</tr>
<tr>
<td><strong>SMALL</strong></td>
<td>Smallest positive number on the host computer. This is used when non-positive numbers are plotted with log scaling.</td>
</tr>
</tbody>
</table>
CHAPTER 7: PREPROCESSING TO CREATE A SYSTEM PLOT PACKAGE

INTRODUCTION
This chapter provides reference information for NCAR Graphics Project programmers, or for programmers with access to the NCAR CDC 7600 who wish to attempt construction of a custom graphics package.

The master system plot package is written in an extension of FORTRAN called FRED (see NCAR Software Support Library, Volume III, Chapter 13). The FRED preprocessor reads card images of its FORTRAN extension as input and creates a FORTRAN output file of card images.

at this time, FRED runs only on NCAR's Control Data 7600 using NCAR's system. It is not portable. To create a system plot package, the following deck structure is used:

*JOB,ssss,bbbbbbbb,user-name
*LIMIT,T=14S
*RUN,ED,BS=ULIB,BN=FRED
FETCH,S=(43510006),SN=PLOT.FRED,IL
(mods)
*EDIT,S=PSCR,D=PLIB,DN=name
*END

MODS
The mods are used to change the values of FRED-time variables. The nature of the mods determines the characteristics of the plot package produced. The FRED-time variables are described in Understanding the Master System Plot Package. The variables most commonly changed are listed in the following table:
<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPLOTR</td>
<td>Specifies metacode output or dd80 output.</td>
</tr>
<tr>
<td>LCOMP</td>
<td>Specifies level of metacode compaction.</td>
</tr>
<tr>
<td>LPORT</td>
<td>Specifies if non-pertable code for NCAR's system can be used or if portable code is required.</td>
</tr>
<tr>
<td>LNCPW</td>
<td>The number of characters that can be stored in a default length integer variable.</td>
</tr>
<tr>
<td>LPAIR</td>
<td>The number of 16-bit metacode packets that are stored before packing. Should be near 15 in magnitude and such that LPAIR*16 bits exactly fill an integral number of default-length integer words.</td>
</tr>
<tr>
<td>LOUTWD</td>
<td>The number of words exactly filled by LPAIR 16-bit packets.</td>
</tr>
<tr>
<td>LUMTCD</td>
<td>Unit number for metacode output.</td>
</tr>
</tbody>
</table>

Examples  The following table gives a few sample mod decks for the production of typical plot packages. The card numbers listed should be verified against a listing of PLOTMASTER. The mod decks are in a form directly usable with the 7600 Editor.
<table>
<thead>
<tr>
<th>Computer</th>
<th>Output</th>
<th>Mods</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCAR 7600</td>
<td>dd80</td>
<td>none</td>
</tr>
<tr>
<td>NCAR 7600</td>
<td>Metacode</td>
<td>CHANGE 50,199,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPLOTR=0&quot;=&quot;LPLOTR=1&quot;</td>
</tr>
<tr>
<td>IBM 370</td>
<td>Metacode</td>
<td>CHANGE 50,199,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPORT=0&quot;=&quot;LPORT=2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPLOTR=0&quot;=&quot;LPLOTR=1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHANGE 200,500,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LNCPW=10&quot;=&quot;LNCPW=4&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPAIR=15&quot;=&quot;LPAIR=16&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LOUTWD=4&quot;=&quot;LOUTWD=8&quot;</td>
</tr>
<tr>
<td>CRAY-1</td>
<td>Metacode</td>
<td>CHANGE 50,199,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPORT=0&quot;=&quot;LPORT=2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPLOTR=0&quot;=&quot;LPLOTR=1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHANGE 200,500,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LNCPW=10&quot;=&quot;LNCPW=8&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPAIR=15&quot;=&quot;LPAIR=16&quot;</td>
</tr>
<tr>
<td>PDP 11*</td>
<td>Metacode</td>
<td>CHANGE 50,199,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPORT=0&quot;=&quot;LPORT=2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPLOTR=0&quot;=&quot;LPLOTR=1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHANGE 200,500,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LNCPW=10&quot;=&quot;LNCPW=2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LPAIR=15&quot;=&quot;LPAIR=16&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LOUTWD=4&quot;=&quot;LOUTWD=16&quot;</td>
</tr>
</tbody>
</table>

Please note that embedded blanks in the quoted strings above are necessary; they are indicated by 5.

*using a compiler with default 16-bit integer variables.
CHAPTER 8: UNDERSTANDING THE MASTER SYSTEM PLOT PACKAGE

NOTE: This chapter provides reference information for NCAR Graphics Project programmers, or for programmers with access to the NCAR CDC 7600 who wish to attempt construction of a custom graphics package.

PREPROCESSING

The master system plot package is written in FRED and preprocessing it with FRED on ULIB results in the production of a system plot package with attributes determined by the state of various FRED-time variables. To understand the master system plot package, the FRED-time variables, plot package common block variables, and user argument lists must be understood. The plot package common block variables are described in Understanding an NCAR Plot Package. The user argument lists are described in the user documentation, The System Plot Package.

PARAMETERS

The FRED-time variables of the master system plot package fall into two classes: those which are never changed (such as L2(I) which is 2^I), and those which are changed to produce system plot packages with various characteristics. The following list contains the variables in the second class.

\[
\begin{align*}
\text{LPLOTR}^\dagger & \quad \text{LNCPW}^\dagger \\
\text{LCOMP}^\dagger & \quad \text{LOUTWD}^\dagger \\
\text{LPACK} & \quad \text{LPAIR}^\dagger \\
\text{LPORT}^\dagger & \quad \text{LSYBUFT} \\
\text{LOUTCD} & \quad \text{LUMTCD}^\dagger \\
\text{LBUFFS} & \quad \text{LFMTC} \\
\text{LLAB} & \\
\end{align*}
\]

\dagger Most commonly changed variables

The following excerpt from the master system plot package contains explanations of all FRED-time variables. All FRED-time variables start with L and no other variables do.
SETS FOR DETERMINING THE TYPE OF CODE

COMMENTS

ABOUT

FLAGS

LFORM FORMAT FOR METACODE OUTPUT
0 FORTRAN BINARY OUTPUT
1 FORMATTED CARD IMAGES FOR BENCHMARKS

LPLOTR TYPE OF PLOTTER CODE OUTPUT
1 METACODE - FOR ALL PLOTTERS
0 DD80 ONLY

LCOMP TYPE OF COMPACTION. NON-ZERO ONLY WHEN LPLOTR IS TOO
0 NONE
1 INCREMENT INSTRUCTIONS USED, REDUNDANCIES OUT
2 HIGH LEVEL SUBSTITUTIONS USED FOR BACKGROUNDS

LPACK TYPE OF ROUTINE PACKAGE BEING PRODUCED
0 A PLOT PACKAGE IS PRODUCED
1 MAKE A TRANSLATOR FOR THE DD80
2 MAKE A TRANSLATOR FOR THE CALCOMP
3 MAKE A TRANSLATOR FOR A PRINTER
4 MAKE A TRANSLATOR FOR A TEKTRONIX
5 MAKE A TRANSLATOR FOR DISTRIBUTION
6 MAKE A TRANSLATOR FOR EXAMINING METACODE

LPORT PORTABILITY FLAG
0 NON-PORTABLE CODE
1 PORTABLE CODE TO A COMPUTER
   WITH 6 BITS PER CHARACTER
2 PORTABLE CODE TO A COMPUTER WITH 8 BITS PER CHARACTER

LOUTCD OUTPUT CODE
0 ASCII
1 EBCDIC

.SET LFORM=0
.SET LPLOTR=0
.SET LCOMP=1
.SET LPORT=0
.SET LOUTCD=0
.SET LPACK=0
F COMMENTS
F ABOUT
F SETS AND
F MACROS
F
F SET VARIABLES FOR CONSTANTS IN THE CODE
F MACROS FOR CONSTANTS IN THE CODE WHICH ARE FUNCTIONS OF OTHER THINGS
F MACRO AND SET DEFINITIONS
F NAME CONVENTION - ALL MACRO AND SET VARIABLES WILL START WITH L AND
F NOTHING ELSE WILL
F
F LADJST  NUMBER TO PUT FLAG BITS IN OP CODES (11000000 BINARY)
F LBPPOS  MACRO TO BIAS A 2 BYTE NUMBER TO THE ALLOWED RANGE
F LBTDND  MACRO TO BOUND INTEGERS TO LEGAL BYTE RANGE
F LBUFFS  0 THRU LBUFFS ALLOWED AS POINTER IN FLASH CALLS
F LBUF16  GET16 BUFFER SIZE IN NON-DD80 MC TRANSLATORS (THUS
F MAX PERMISSIBLE NUMBER OF 16-BIT PACKETS PER RECORD)
F LCAS    OPTION NUMBER FOR CONTROLLING CHARACTER CASE
F LCENT   OPTION NUMBER FOR SETTING CHARACTER CENTERING
F LCODE   VALUE OF A FRED-TIME CONDITIONAL COMPILATION CODE,
F O-ACTIVATE, 1=DEACTIVATE, 2=ELIMINATE
F LCOLR   OPTION NUMBER FOR SETTING COLOR
F LDEC    STANDARD DECREMENT FOR POSITION OF NUMERICAL LABELS
F ON AXES WHEN GENERATED BY GRIDAL
F LENGTH  NUMBER OF CHARACTERS IN A FRED-TIME STRING
F LFMTC   MAXIMUM NUMBER OF CHARACTERS ALLOWED IN THE FORMATS
F PASSED TO THE PLOT PACKAGE THROUGH A LABMOD CALL
F LFMTW   NUMBER OF WORDS NEEDED FOR LFMTC CHARACTERS
F LFONT   OPTION NUMBER FOR CONTROLLING CHARACTER FONT
F LFRAME  FRAME OP CODE
F LFRST   FRSTPT OP CODE
F LGRID   SYSTEM OP CODE FOR GRIDAL CALL
F LHIGH   HIGH INTENSITY (80 PCT OF AVAILABLE)
F LIMPOS  IMPOSSIBLE NUMBER FOR INITIALIZING VARIABLES
F LINT    OPTION NUMBER FOR CONTROLLING INTENSITY
F LLAB    MAXIMUM NUMBER OF CHARACTERS IN GRIDAL LABEL (AND
F THUS THE LARGEST VALUE FOR NUMX AND NUMY IN LABMOD)
F LLABM   SYSTEM OP CODE FOR LABMOD CALL
F LLOW    LOW INSENSITIVITY (49.9 PCT OF AVAILABLE)
F LM(I)   MACRO WHICH YIELDS A RIGHT-JUSTIFIED MASK I BITS LONG
F LMAXOP  MAXIMUM NUMBER OF OPTIONS
F LMGR    DEFAULT MAJOR TICK MARK LENGTH
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMNR</td>
<td>Default minor tick mark length</td>
</tr>
<tr>
<td>LNBPC</td>
<td>Number of bits per character</td>
</tr>
<tr>
<td>LNBPW</td>
<td>Number of bits per word on target computer (derived from LPAIR and LOUTWD)</td>
</tr>
<tr>
<td>LNCPW</td>
<td>Number of characters per word on the computer</td>
</tr>
<tr>
<td>LMASK</td>
<td>Number of masks for DD80 instruction formation</td>
</tr>
<tr>
<td>LNOP</td>
<td>2 byte no-op instruction (pen up increment of 0)</td>
</tr>
<tr>
<td>LNPMP</td>
<td>Number of option masks for DD80 instructions</td>
</tr>
<tr>
<td>LOCI(I)</td>
<td>Inverse of LOC function, that is, causes a subroutine to look in address I for an argument.</td>
</tr>
<tr>
<td>LOPEND</td>
<td>Last system op code</td>
</tr>
<tr>
<td>LOPNUM</td>
<td>Number of system defined option numbers</td>
</tr>
<tr>
<td>LOPPR</td>
<td>Macro to shift op code and put in flag bits in pair</td>
</tr>
<tr>
<td>LOPST</td>
<td>First system op code</td>
</tr>
<tr>
<td>LOPSTA</td>
<td>First system op code with flag bits</td>
</tr>
<tr>
<td>LOR</td>
<td>Option number for controlling character orientation</td>
</tr>
<tr>
<td>LOUTWD</td>
<td>Number of words that LPAIR byte pairs fill</td>
</tr>
<tr>
<td>LPAIR</td>
<td>Number of byte pairs saved before packing</td>
</tr>
<tr>
<td>LPAIRF</td>
<td>Macro to form byte pair from two bytes at run time</td>
</tr>
<tr>
<td>LPAIR1</td>
<td>The 16-bit patterns used to initialize the first</td>
</tr>
<tr>
<td>LPAIR2</td>
<td>2 words of MBPRS every LRBITS of generated data, so that PREOUT may format first 32 bits before output.</td>
</tr>
<tr>
<td>LPAT</td>
<td>Option number for controlling dashed line patterns</td>
</tr>
<tr>
<td>LPOINT</td>
<td>Point op code for DD80 instructions</td>
</tr>
<tr>
<td>LPSYM</td>
<td>OP code for PSYM when DD is activated</td>
</tr>
<tr>
<td>LPTS</td>
<td>OP code to switch to point mode</td>
</tr>
<tr>
<td>LPWRIT</td>
<td>Character op code</td>
</tr>
<tr>
<td>LRANG</td>
<td>OP code for instruction containing range</td>
</tr>
<tr>
<td>LRBITS</td>
<td>Size, in bits, of fixed-length metacode output</td>
</tr>
<tr>
<td>LRELS</td>
<td>Records (11520 now).</td>
</tr>
<tr>
<td>LRELS</td>
<td>OP code to switch to relative addressing (deferred)</td>
</tr>
<tr>
<td>LSBUFF</td>
<td>Pointer to system buffer bounds</td>
</tr>
<tr>
<td>LSCR</td>
<td>Number of addressable positions on plotter</td>
</tr>
<tr>
<td>LSCRPCD</td>
<td>Power (of 2) of default screen size</td>
</tr>
<tr>
<td>LSETP</td>
<td>Option op code</td>
</tr>
<tr>
<td>LSIZE</td>
<td>Option number for controlling character size</td>
</tr>
<tr>
<td>LSSIZ</td>
<td>Option number for spot size</td>
</tr>
<tr>
<td>LSYBUF</td>
<td>System buffer length</td>
</tr>
<tr>
<td>LTICK</td>
<td>System op code for setting tick length</td>
</tr>
<tr>
<td>LUDDB0</td>
<td>DD80 unit number on NCAR system</td>
</tr>
</tbody>
</table>
COMMENTS ABOUT
SETS AND MACROS
(Con't)

F LUMTCD  UNIT NUMBER FOR OUTPUTTING METACODE
F LUNORM  MACRO TO PERFORM THE INVERSE OF LNORM
F LUSROP  NUMBER OF USER DEFINABLE OPTIONS
F LVECT   VECTOR OP CODE
F LWDS(NC) FUNCTION TO COMPUTE THE NUMBER OF WORDS NEEDED TO
          HOLD NC CHARACTERS
F L2(I)   ITH POWER OF 2
.MACRO LOCI(IADDR)=IDUMMY(IADDR-LOC(IDUMMY)+1)
.MACRO LWDS(ICHARS)="(ICHARS-1)/LNCPW+1"
.MACRO LBPOSS(I)=IAND(I+L2(15),"L2(16)-1")
.MACRO LBTBND(IBYT)=IAND(IBYT,"L2(8)-1")
.MACRO LCODE(ICCC)="ADE(ICCC)"
.MACRO LENGTH(ICHARS)="LOS(ICHARS)"
.MACRO LM(I)="L2(I)-1"
F LOPPR(IOP,INOC) DEFINED BELOW BECAUSE MACHINE NATURE NEEDED.
.MACRO LPAIRF(IL, IR)=IOR(ISHIFT(LBTBND(IL),8),LBTBND(IR))
.MACRO LUNORM(I1, I2)=ISHIFT(I1, I2)-L2(15-I2)

F SETS AND MACROS DEFINED

SET TYPE INTEGER L2(47)
SET L2(1)=2
REPEAT (I=2,47)
SET L2(I)=L2(I-1)*2
.
F

IF LPORT .EQ. 0
MA IAND=AND
MA IOR=OR
MA JLM2(K)=AND(K,LLM2)
SET LLM2=77770000000000000B
MA ISHIFT(K1,K2)=SHIFT(K1,K2)
ENDIF
F

SET TYPE COUNTER INCR
SET LIMPOS =-9999
SET LNCPW =10
SET LRBITS =11520
SET INCR=0
F PERSERVE THE ORDER OF LCAS THROUGH LPAT FOR QUICK DD80 RECOGNITION
SET LCAS =INCR
SET LINT =INCR
SET LOR =INCR
SET LSIZE =INCR
SET LFONT =INCR
SET LPAT =INCR
SET LSSIZ =INCR
SET LCENT =INCR
.SET LCOLR =INCR
.IF LNCPW.LE.2
.SET LDUMY =INCR
.ENDIF
.SET LOPNUM =INCR-1
.SET LUSROP =10
.SET LMAXOP =LOPNUM+LUSROP
F PLOTIT MUST BE CHANGED IF LVECT IS CHANGED.
.SET LVECT =1
.SET LFRST =1-LVECT
.SET LBUFFS =10
.SET LSBUFF =LBUFFS+2
.SET LOPST =L2(5)
.SET LOPEND =L2(6)-1
.SET LADJST =L2(7)+L
.F TOP
.IF LCODE(MC).EQ.0 OR LPACK.EQ.1
.SET INCR=L2(5)+L2(6)
.SET LPWRIT =INCR
.SET LFRAME =INCR
.SET LSETOP =INCR
.SET LGRID =INCR
.SET LTICK =INCR
.SET LLABM =INCR
.SET LRANG =INCR
.SET LPTS =INCR
.SET LREL =INCR
.OTHERWISE
.SET LPWRIT =3
.SET LFRAME =5
.SET LPSYM =4
.SET LPTS = 6
.SET LREL =9
.F DASHLN =LNMASK
.SET LNOPM =5
.ENDIF
.F
.SET LUDD80 ="6774B"
.SET LNMask =7
. IF Lcode(MC) .EQ. 0
 .SET LPAIR =15
 .SET LOUTWD =4
 .SET LNPW ="(16*LPAIR-1)/LOUTWD + 1"
 .SET LSYBUF ="(LRBITS-1)/LNPW + 1"
 .OTHERWISE
 .SET LPAIR =5
 .SET LOUTWD =3
 .SET LPOINT =LFRST+LVECT+1
 .SET LSYBUF =256
 .ENDIF
 .SET LUMTC =8

F

F

.SET LFMC =10
.SET LFMCT =LWDS(LFMTC)
.SET LLAB =20
.SET LLOW=127
.SET LHIGH=204

F

.SET LSCR =L2(15)-1
.SET LSCRD =10
.SET LDEC =LSCR/50
.SET LMGR =LSCR/85
.SET LMNR =LSCR/128
. IF LCODE(DD) .EQ. 0
 .MACRO PREOUT=Q8QWRB(MBUFA,MBUFLU,MUNIT)
 .ENDIF
 .IF LPORT .EQ. 0
 .MACRO TRANS =Q8QTRN
 .MACRO SPLTDA=Q8QSDA
 .MACRO PUT42 =Q8QP42
 .MACRO FLUSHB=Q8QFLB
 .MACRO WRITEB=Q8QWRB
 .MACRO SYSPLT=Q8QPPC
 .MACRO JUSTFY=Q8QJUS
 .ENDIF

8-8
COMMENTS ABOUT CODES

CONDITIONAL COMPILATION CODES

A ARGUMENT COMMENTS FOR USERS - ELIMINATE OR DEACTIVATE ONLY
AS ASCII
C FORTRAN COMMENTS
CA CALCOMP CODE CREATION
D COMMENTS FOR DD80 CODE CREATION
DD DD80 CODE CREATION
EB EBCDIC
EC EIGHT BITS PER CHARACTER ON COMPUTER
EE ERROR MESSAGES EXTENSIVE
ES ERROR MESSAGES IN SHORT MODE
EX EXAMINING METACODE (LPACK=5)
F FRED-TIME COMMENT CARDS - ELIMINATE ONLY
HO HOLLERITH INFORMATION FORMED FROM ASCII (USED IN TRANSLATOR ONLY)
I COMMENTS FOR II
II INCREMENT INSTRUCTION CREATION
J COMMENTS FOR SB
LV LOCAL VARIABLES USED WHenever POSSIBLE TO REDUCE SETUP TIME
M COMMENTS FOR METACODE CREATION
MC METACODE CREATION
N COMMENTS FOR NC
NC NON-COMPACTING STATEMENTS FOR WHEN UC IS TURNED OFF
NP NONPORTABLE CODE TO BE REPLACED BY PC CODE
OV OVERFLOW TO DISK WHEN FLASH BUFFER FILLS
PC PORTABILITY CODE
PR PRINTER USED AS PLOTTER (USED IN TRANSLATOR ONLY — LPACK .NE. 0)
RA MULTIBYTE RANGE INSTRUCTION CREATION
RF REDUNDANT FRSTPT ELIMINATION
RO REDUNDANT OPTION CODE ELIMINATION
RV REDUNDANT (ZERO LENGTH) VECTOR ELIMINATION
SB SIX BITS PER CHARACTER ON COMPUTER
SE CODE TO PICK SB OR EC
TR TRACE DEBUGGING CODE
U COMMENTS FOR UC
UC ULTRA-COMPACT OUTPUT CODE (FOR METACODE ONLY)
W WARNING COMMENTS FOR IMPLEMENTORS - ELIMINATE OR DEACTIVATE ONLY

THESE CODES ARE ACTIVATED, DEACTIVATED, OR ELIMINATED ACCORDING TO
FLAGS SET IN THE MACRO DEFINITIONS. TO TUNE THIS PACKAGE TO A PAR-
TICULAR USE, OVERRIDE THAT SETTING WHERE MARKED BELOW.
<table>
<thead>
<tr>
<th>GLOBALS</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTMOV</td>
<td>Moves NCHAR characters from ICHAR to the instruction stream. If configured for a 6-bit machine and metacode is being produced, the characters are translated to 8-bit characters. If dd80 code is being produced, flag characters are resolved.</td>
</tr>
<tr>
<td>CHRTRN</td>
<td>Translation tables for various character sets. used in block data.</td>
</tr>
<tr>
<td>CCOM</td>
<td>Comments explaining the variables in the common block.</td>
</tr>
<tr>
<td>COMMON</td>
<td>The common block.</td>
</tr>
<tr>
<td>DRAWAL</td>
<td>Used in GRIDAL to draw axis lines.</td>
</tr>
<tr>
<td>DRAWLT</td>
<td>Used in GRIDAL to draw lines and ticks.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Outputs error messages and causes termination.</td>
</tr>
<tr>
<td>FINDN</td>
<td>Finds a name in the MNAME table. Used in OPTN and GETOPT.</td>
</tr>
<tr>
<td>INITMB</td>
<td>Initializes MBPAIRS with MPAIR1 and MPAIR2; sets MPAIR to 2. Metacode packages only.</td>
</tr>
<tr>
<td>IMP</td>
<td>Marks places in code where implementors MUST change the code. Used in Block Data.</td>
</tr>
<tr>
<td>LOCSUB</td>
<td>Used to cause a local variable to be used to hold an argument to a subroutine.</td>
</tr>
<tr>
<td>PUT</td>
<td>Causes an instruction with OP CODE LCODE to be formed using MX and MY.</td>
</tr>
<tr>
<td>PUTPAIR</td>
<td>Puts a byte pair in the holding buffer and has the holding buffer dumped when it gets full.</td>
</tr>
<tr>
<td>TRN</td>
<td>Causes X and Y (which may be integer or floating) to be translated to plotter address units and put in MX and MY.</td>
</tr>
<tr>
<td>TRNPUT</td>
<td>TRN and PUT at once.</td>
</tr>
<tr>
<td>TRXY</td>
<td>Causes an X or Y value represented by Z (which equals 1HX or 1HY) to be translated from integer or floating point space to plotter address units and stored.</td>
</tr>
</tbody>
</table>
WARN Outputs warnings.
LIST OF PLOTS

Contained on the following pages are samples of pictures generated by the Graphics Utilities, using the demonstration drivers.

- AUTOGRAPH
- CONRAN
- CONRAQ
- CONRAS
- CONREC
- CONRECQCK
- CONRECSMTH
- CONRECSUPR
- DASHCHAR
- DASHLINE
- DASHSMTH
- DASHSUPR
- HAFTON
- ISOSRF
- ISOSRFHR
- PWRITX
- PWRITY
- PWRZI
- PWRZS
- PWRZT
- SCROLL
- SRFACE
- STRMLN
- SUPMAP
- THREED
- VELVCT
- WINDOW
SAMPLE PLOTS: AUTOGRAF

DEMONSTRATING EZXY ENTRY IN AUTOGRAPH

DEMONSTRATING EZY ENTRY OF AUTOGRAPH

DEMONSTRATING EZXY ENTRY OF AUTOGRAPH

DEMONSTRATING EZMY ENTRY OF AUTOGRAPH
SAMPLE PLOTS: CONRAQ, CONRAQ, CONCATAS
SAMPLE PLOTS: CONREC, CONRECOCK

DEMONSTRATION PLOT FOR CONREC ENTRY OF CONREC

DEMONSTRATION PLOT FOR EXCENTR ENTRY OF CONREC

DEMONSTRATION PLOT FOR CONREC ENTRY OF CONRECOCK

DEMONSTRATION PLOT FOR EXCENTR ENTRY OF CONRECOCK
SAMPLE PLOTS: CONRECSMTH, CONRECSUPR

DEMONSTRATION PLOT FOR CONREC ENTRY OF CONRECSMTH

DEMONSTRATION PLOT FOR ECONTRA ENTRY OF CONRECSMTH

DEMONSTRATION PLOT FOR CONREC ENTRY OF CONRECSUPR

DEMONSTRATION PLOT FOR ECONTRA ENTRY OF CONRECSUPR
SAMPLE PLOTS: DASCHMAR, DASHLINE, DASHSMITH, DASHSUPR
SAMPLE PLOTS: ISOSRF and ISOSEFBR
SAMPLE PLOTS: PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX

DEMONSTRATION PLOT FOR PWRITX
SAMPLE PLOTS: PURITY, PURZI, PURZS, PURZT
SAMPLE PLOTS: STIMUL and THREED

DEMONSTRATION PLOT FOR ROUTINE STIMUL

DEMONSTRATION PLOT FOR ROUTINE THREED
SAMPLE PLOTS: SUPMAP

SUPMAP DEMONSTRATION, STEREOGRAPHIC PROJECTION

SUPMAP DEMONSTRATION, ORTHOGRAPHIC PROJECTION

SUPMAP DEMONSTRATION, LAMBERT EQUAL AREA PROJECTION

SUPMAP DEMONSTRATION, LAMBERT CONFORMAL CONIC PROJECTION
SAMPLE PLOTS: SUPMAP (Con't)
SAMPLE PLOTS: SUPMAP (Con't)
SAMPLE PLOTS: VELVCT and WINDOW
The following article should be placed in the Implementor’s Guide portion of your copy of the Graphics Manual. It is an addition to the existing documentation:

NOTES ON CONVERTING THE SYSTEM PLOT PACKAGE FOR BYTE-ADDRESSABLE MACHINES

Discussed here are mods made to convert a word-addressable plot package running on a DEC 11/70, to a byte-addressable package running on the same machine. These mods should be instructive for conversion of the plot package to any byte-addressable machine. In what follows "nbw" will denote the number of bytes-per-word on the target machine (in the specific case under consideration, nbw=4).

If the plot package can be compiled with a FORTRAN compiler which guarantees that variables are loaded on word boundaries, then it would not be necessary to make the following mods—in such a case make sure the LOC function returns word addresses, and not byte addresses.

The most common situation where it is necessary to make more than trivial mods is on a strictly byte-addressable machine (VAX, DEC11/70, etc.) running FORTRAN IV PLUS where integer variables are four bytes long.

It should be emphasized that the mods below are simply "a way" to convert the package, and not "the way." No thought was given to doing the conversion in the best or most efficient way, but mainly to get something which works.

CHANGES TO THE SUPPORT ROUTINES

Make sure the LOC function returns byte addresses.

If necessary, change the implementation of WRITEB to reflect the fact that the first argument is a byte address.

If necessary, change the implementation of PACKUM to reflect that the final argument is a byte address.

CHANGES TO THE PORTABLE PLOT PACKAGE

Provide a subroutine SUBROUTINE I2B(IB,IW,N) which takes an array of N integers in IW and stores them sequentially as bytes in the INTEGER*1 array
IB; provide a subroutine SUBROUTINE B2I(IB,IW,N) which takes N bytes stored in the INTEGER*1 array IB and stores them sequentially in the word array IW.

An implementation of these two subroutines for the specific case in point:

SUBROUTINE I2B(IB,IW,N)

C
C THIS SUBROUTINE TAKES THE N WORDS IN ARRAY IW AND STORES
C THEM IN THE INTEGER*1 ARRAY IB ONE BYTE AT A TIME.
C
INTEGER*1 IB(1),IK(4)
DIMENSION IW(1),IJ(1)
EQUIVALENCE (IK(1),IJ(1))
DO 10 I=1,N
INDX = 4*(I-1)+1
IJ(1) = IW(I)
IB(INDX) = IK(1)
IB(INDX+1) = IK(2)
IB(INDX+2) = IK(3)
IB(INDX+3) = IK(4)
10 CONTINUE
RETURN
END

September 14, 1982  ADDENDA-2  IMPLEMENTOR'S GUIDE
SUBROUTINE B2I(IB, IW, N)

C
C THIS SUBROUTINE TAKES THE N BYTES IN ARRAY IB AND STORES
C THEM IN THE INTEGER ARRAY IW FOUR BYTES AT A TIME.
C
INTEGER*1 IB(1), IK(4)
DIMENSION IW(1), IJ(1)
EQUIVALENCE (IK(1), IJ(1))

C J=NO. OF WORDS TO BE FILLED; K=NO. OF BYTES IN FINAL WORD
J = (N+3)/4
K = MOD(N-1,4)+1
IF (K.EQ.4) JNDX = J
IF (K.NE.4) JNDX = J-1
DO 10 I=1, JNDX
   INDX = 4*(I-1)+1
   IK(1) = IB(INDX)
   IK(2) = IB(INDX+1)
   IK(3) = IB(INDX+2)
   IK(4) = IB(INDX+3)
   IW(I) = IJ(1)
10 CONTINUE
   IF (K.EQ.4) RETURN
   DO 40 L=1,4
      IK(L) = 0
40 CONTINUE
   DO 30 L=1, K
      IK(L) = IB(N-K+L)
30 CONTINUE
   IW(J) = IJ(1)
20 CONTINUE
RETURN
END
In what follows, we consider, in order, the changes which should be made to the appropriate subroutines of the portable plot package.

FLASH2

Change:  
   DIMENSION IDUMMY(1)  
To:  
   INTEGER*1 IDUMMY(1)  

Change:  
   MLWA(KPOINT+1) = MBUFA+MBUFLU-1  
   ISUB = MBUFA+MBUFLU-LOC(IDUMMY)  
   IDUMMY(ISUB+1) = MBUFLU  
   IDUMMY(ISUB+2) = MJXMIN  
   IDUMMY(ISUB+3) = MJYMIN  
   IDUMMY(ISUB+4) = MJYMAX  
   IDUMMY(ISUB+5) = MJYMAX  

To:  
   MLWA(KPOINT+1) = MBUFA+nbw*MBUFLU-1  
   ISUB = MBUFA+nbw*MBUFLU-LOC(IDUMMY)  
   INDX = ISUB+1  
   CALL I2B(IDUMMY(INDX),nbw*MBUFLU,1)  
   INDX = INDX+nbw  
   CALL I2B(IDUMMY(INDX),MJXMIN,1)  
   INDX = INDX+nbw  
   CALL I2B(IDUMMY(INDX),MJYMIN,1)  
   INDX = INDX+nbw  
   CALL I2B(IDUMMY(INDX),MJYMAX,1)  
   INDX = INDX+nbw  
   CALL I2B(IDUMMY(INDX),MJYMAX,1)
FLASH3

Change: DIMENSION IDUMMY(1)
To: INTEGER*1 IDUMMY(1)

Change: NUSR = IDUMMY(ISUB+1)
To: CALL B2I(IDUMMY(ISUB+1),NUSR,nbw)

Change: MBUFLU = MINO(NLEN,MSBLEN)
IF (MBUFLU .GT. 0) CALL PRED
NLEN = NLEN-MSBLEN
MBUFA = MBUFA+MSBLEN
IF (NLEN .GT. 0) GO TO 101

To: 101 NBYT = MINO(NLEN,nbw,MSBLEN)
ITMP = MOD(NLEN,nbw)
MBUFLU = NLEN/nbw+(ITMP+nbw-1)/nbw
MBUFLU = MINO(MBUFLU,MSBLEN)
IF (NBYT .GT. 0) CALL PRED
NLEN = NLEN-ITMP
MBUFA = MBUFA+ITMP
IF (NLEN .GT. 0) GO TO 101

Change: MJXMIN = MIN0(MJXMIN,IDUMMY(ISUB+2))
MJYMIN = MIN0(MJYMIN,IDUMMY(ISUB+3))
MJXMAX = MAX0(MJXMAX,IDUMMY(ISUB+4))
MJYMAX = MAX0(MJYMAX,IDUMMY(ISUB+5))

To: CALL B2I(IDUMMY(ISUB+nbw+1),MIMP,nbw)
MJXMIN = MIN0(MJXMIN,MIMP)
CALL B2I(IDUMMY(ISUB+2*nbw+1),MIMP,nbw)
MJYMIN = MIN0(MJYMIN,MIMP)
CALL B2I(IDUMMY(ISUB+3*nbw+1),MIMP,nbw)
MJXMAX = MAX0(MJXMAX,MIMP)
CALL B2I(IDUMMY(ISUB+4*nbw+1),MIMP,nbw)
MJYMAX = MAX0(MJYMAX,MIMP)
FLASH4

Change:
DIMENSION IFW(1),LWD(1)
To:
INTEGER*1 IFW(1),LWD(1)

Change:
JLDA = LOC (LWD)
To:
JLWDA = LOC (LWD) + nbw - 1

Change:
NEXTRA = 5
To:
NEXTRA = nbw * 5

FLUSHR

Change:
DIMENSION IDMMY(1)
To:
INTEGER*1 IDMMY(1)

Change:
CALL PACKUM (MBPRS,16,MBUFA+MBUFLU)
To:
CALL PACKUM (MBPRS,16,MBUFA+nbw*MBUFLU)

PREOUT

Change:
DIMENSION IDUMMY(1)
To:
INTEGER*1 IDUMMY(1)

Change:
MSYBUF(I) = IDMMY(ISUB)
ISUB = ISUB+1
To:
CALL B2I (IDUMMY(ISUB),MSYBUF(I),nbw)
ISUB = ISUB+nbw

Change:
IDUMMY(ISUB) = IOR(IDUMMY(ISUB),ISHIFT(NBYTES,IBFW-16))
To:
CALL B2I (IDUMMY(ISUB),ITMP,nbw)
ITMP = IOR(ITMP,ISHIFT(NBYTES,IBFW-16))
CALL I2B(IDUMMY(ISUB),ITMP,1)

Change:
ISUB = KBUFA-LOC(IDUMMY)+IFWD
To:
ISUB = KBUFA-LOC(IDUMMY)+nbw*(IFWD-1)+1

Change:
IDUMMY(ISUB) = IOR(IDUMMY(ISUB),ISETFB)
To:
CALL B2I (IDUMMY(ISUB),ITMP,nbw)
ITMP = IOR(ITMP,ISETFB)
CALL I2B(IDUMMY(ISUB),ITMP,1)

Change:
106 IDUMMY(ISUB) = IAND(IDUMMY(ISUB),ICLRFB)
To:
106 CALL B2I (IDUMMY(ISUB),ITMP,nbw)
ITMP = IAND(ITMP,ICLRFB)
CALL I2B(IDUMMY(ISUB),ITMP,1)
Change: ISUB = KBUFA+MBUFLU-LOC(IDUMY)  
To:  
ISUB = KBUFA+nbw*MBUFLU-LOC(IDUMY)-nbw+1

Change: ISUB = ISUB+I  
IDUMY(ISUB) = 0  
To:  
ISUB = ISUB+nbw  
ITMP = 0  
CALL I2B(IDUMMY(ISUB),ITMP,1)

Change: ISUB = KBUFA-LOC(IDUMY)  
To:  
ISUB = KBUFA-LOC(IDUMY)-nbw+1

Change: ISUB = ISUB+1  
IDUMY(ISUB) = 0  
To:  
ISUB = ISUB+nbw  
ITMP = 0  
CALL I2B(IDUMMY(ISUB),ITMP,1)
INDEX

Abnormally terminated job, 5-2
ACQUISITION, 2-1
Added,
  new plotter, 4-1.
Added flexibility, 1-2
Architecture,
  machine, 6-1
Argument lists, 8-1
Arguments,
  routine, 6-1
Axis drawing, 4-2
Base-B form, 3-4
Bit manipulation, 1-2
Block,
  named common, 6-1
BLOCK DATA routine, 2-1
Boundaries,
  record, 5-1
Character drawing, 1-1
Characters,
  hardware, 4-1
Code,
  matching, 2-1
  tailored, 2-1
Code compaction,
  output, 6-1
CODE DEFINITIONS,
  OP, 4-4
Codes,
  user-defined option, 4-1
CODES,
  OP, 4-5
Common block,
  named, 6-1
Compaction,
  output code, 6-1
Computer,
  target, 2-1
Conditional structures, 6-1
Configuration,
  target machine, 1-2
Constants,
  machine, 3-3
Constructs,
  metacode, 4-1
Control,
  pen, 4-1
Core requirements, 1-2
Cracker,
  format, 3-1
Current marker, 4-5
Data records, 5-1
DATA routine,
  BLOCK, 2-1
Data type identifier, 5-1
DECK STRUCTURE, 7-1
Deck structure, 7-1
DEFINITIONS,
  OP CODE, 4-4
Device-independent, 1-1
Device-independent
  instructions, 2-1, 4-1
Double precision, 3-4
Drawing,
  character, 1-1
  line, 1-1
  axis, 4-2
Drum plotters, 4-5
Dumps,
  hexadecimal, 2-1
  octal, 2-1
EFFICIENCY,
  PAPER, 4-5
ENCODE,
  Subroutine, 3-1
ENHANCEMENTS, 1-2
Error,
  fatal, 3-2
Examples, 7-2
  .FALSE., 3-3
Fatal error, 3-2
Flexibility,
  added, 1-2
Floating point numbers, 3-4
Fonts, 4-1
Form,
  software request, 2-1
  base-B, 3-4
Format,
  metafile, 3-1
Format cracker, 3-1
FORMATION, 1-1
FORMATION, 6-1
Four-byte positioning instruction, 4-1
Frame, 5-2
FRED, 1-1
FRED-time variables, 8-1
Function I1MACH, 3-3
Function IAND, 3-3
FUNCTION INTT (X), 3-3
Function IOR, 3-3
Function ISHIFT, 3-2
FUNCTION LOC (X), 3-2
Function R1MACH, 3-4
GETCHR,
Subroutine, 3-2
Graphics,
low-level, 1-1
Graphics interface,
portable, 1-1
GRAPHICS METAFILE STRUCTURE,
NCAR'S, 5-1
Hardware, 2-1
Hardware characters, 4-1
Hexadecimal dumps, 2-1
Host machine, 4-5
R1MACH,
Function, 3-3
IAND,
Function, 3-3
Identifier,
data type, 5-1
Implementation, 2-1
IMPLEMENTATIONS,
ROUTINE, 3-5
IMPLEMENTATIONS (Con't),
ROUTINE, 3-6
IMPLEMENTATIONS (Cont),
ROUTINE, 3-7
Increment instruction,
two-byte, 4-1
Increment instruction, 4-2
Instruction,
four-byte positioning, 4-1
two-byte increment, 4-1
increment, 4-2
range, 5-1
Instruction set, 1-1
Instructions,
device-independent, 2-1, 4-1
INSTRUCTIONS,
METACODE, 4-1
Instructions,
multibyte, 4-1
no-op, 5-1
INSTRUCTIONS (Con't),
METACODE, 4-3
Interface,
portable graphics, 1-1
Interfaced points, 4-1
INTRODUCTION, 5-1
INTRODUCTION,
INTRODUCTION TO METACODE, 4-1
INTT (X),
FUNCTION, 3-3
I/O, 1-2
IOR,
Function, 3-3
ISHIFT,
Function, 3-2
Job,
abnormally terminated, 5-2
Line drawing, 1-1
Lists,
argument, 8-1
LOC (X),
FUNCTION, 3-2
Loopy, 6-1
Low-level graphics, 1-1
Machine,
host, 4-5
Machine architecture, 6-1
Machine configuration,
target, 1-2
Machine constants, 3-3
Manipulation,
bit, 1-2
Manipulator,
metacode, 5-1
Marker,
current, 4-5
Master system plot package, 2-1
MASTER SYSTEM PLOT PACKAGE,
UNDERSTANDING THE, 8-1
Matching code, 2-1
Metacode, 4-1
METACODE,
INTRODUCTION TO, 4-1
Metacode constructs, 4-1
METACODE INSTRUCTIONS, 4-1
METACODE INSTRUCTIONS (Con't), 4-3
Metacode manipulator, 5-1
Metacode translation, 4-1
Metatile format, 3-1
METAFILE STRUCTURE, 5-1
METAFILE STRUCTURE,
NCAR'S GRAPHICS, 5-1
METAFILE STRUCTURE,
OLD vs. NEW, 5-2
MODS, 7-1
Multibyte instructions, 4-1
Named common block, 6-1
NCAR system-dependent, 3-4
NCAR'S GRAPHICS METAFILE STRUCTURE, 5-1
Needed routines, 3-1
NEW METAFILE STRUCTURE,
   OLD vs., 5-2
New plotter added, 4-1
No-op instructions, 5-1
Numbers,
   floating point, 3-4
Octal dumps, 2-1
Old structure, 5-2
OLD vs. NEW METAFILE STRUCTURE, 5-2
OP CODE DEFINITIONS, 4-4
OP CODES, 4-5
Option codes,
   user-defined, 4-1
Options,
   setting, 4-4
Output code compaction; 6-1
OVERVIEW, 1-1
Package,
   master system plot, 2-1
PACKUM,
   Subroutine, 3-2
PAPER EFFICIENCY, 4-5
Paper waste, 4-5
PARAMETERS, 8-1
Pen control, 4-1
PERROR,
   SUBROUTINE, 3-2
PERROR,
   Subroutine, 3-2
PHILOSOPHY,
   SYSTEM PLOT PACKAGE, 1-1
PICTURE,
   SAMPLE, 2-2
PICTURE,
   SAMPLE, 2-3
Picture size, 4-1
Plot package,
   master system, 2-1
PLOT PACKAGE,
   UNDERSTANDING THE MASTER SYSTEM, 8-1
PLOT PACKAGE PHILOSOPHY,
   SYSTEM, 1-1
PLOTS,
   SAMPLE, 9-1
Plotter added, new, 4-1
Plotters,
   drum, 4-5
Point numbers,
   floating, 3-4
Points,
   interfaced, 4-1
Portability problems, 5-2
Portable graphics interface, 1-1
Portable test program, 1-2
PORTABLE TRANSLATION, 4-1
Positioning instruction,
   four-byte, 4-1
Postprocessors, 4-2
Precision,
   double, 3-4
Precision,
   single, 3-4
PREPROCESSING, 8-1
Preprocessor, 1-1
Problems,
   portability, 5-2
   portable test, 1-2
   test, 2-1
PURPOSE, 6-1
R1MACH,
   Function, 3-4
Range instruction, 5-1
Record boundaries, 5-1
Records,
   data, 5-1
Request form,
   software, 2-1
Requirements,
   core, 1-2
Resolution, 4-1
Right-justified, 3-2
Routine,
   BLOCK DATA, 2-1
Routine arguments, 6-1
ROUTINE IMPLEMENTATIONS, 3-5
ROUTINE IMPLEMENTATIONS (Con't), 3-6
ROUTINE IMPLEMENTATIONS (Cont), 3-7
Routines,
   support, 2-1
Routines,
   needed, 3-1
SAMPLE PICTURE, 2-2
SAMPLE PICTURE, 2-3
SAMPLE PLOTS, 9-1
Scaling, 1-1
Scaling, 4-2
Set,
   instruction, 1-1
SETCHR,
   Subroutine, 3-2
Setting options, 4-4
Single precision, 3-4
Size,
   picture, 4-1
Sizes, various, 4-1
Software, transporting, 1-2
Software request form, 2-1
STRUCTURE, METAFILE, 5-1
STRUCTURE, NCAR'S GRAPHICS METAFILE, 5-1
Structure, old, 5-2
STRUCTURE, OLD vs. NEW METAFILE, 5-2
STRUCTURE, DECK, 7-1
Structure, deck, 7-1
Structures, conditional, 6-1
Subroutine ENCODE, 3-1
Subroutine GETCHR, 3-2
Subroutine PACKUM, 3-2
SUBROUTINE PERROR, 3-2
Subroutine PERROR, 3-2
Subroutine SETCHR, 3-2
Subroutine ULIBER, 3-1
Subroutine WRITEB, 3-1
SUPPORT, 2-1
Support routines, 2-1
System plot package, master, 2-1
SYSTEM PLOT PACKAGE, UNDERSTANDING THE MASTER, 8-1
SYSTEM PLOT PACKAGE PHILOSOPHY, 1-1
System-dependent, NCAR, 3-4
Tailored code, 2-1
Target computer, 2-1
target machine configuration, 1-2
T-digit, 3-4
Terminated job, abnormally, 5-2
Test program, portable, 1-2
Test program, 2-1
TESTING, 2-1
THE MASTER SYSTEM PLOT PACKAGE, UNDERSTANDING, 8-1
Translation, metacode, 4-1
TRANSLATION, PORTABLE, 4-1

Translators, 1-2
TRANSPORTATION, 1-2
Transporting software, 1-2
TRUE., 3-3
Two-byte increment instruction, 4-1
Type identifier, data, 5-1
ULIBER, Subroutine, 3-1
User-defined option codes, 4-1
Variables, FRED-time, 8-1
various sizes, 4-1
Waste, paper, 4-5
WRITEB, Subroutine, 3-1
(X), FUNCTION LOC, 3-2
(X), FUNCTION INTT, 3-3
Zero-filled, 3-2
FORMATTED DATA ASSIGNMENT

Formatted data assignment operations define entities by transferring data between input/output list items and internal records. Like other assignment statements, formatted data assignment statements only perform internal data transfers. Like formatted input/output statements, formatted data assignment statements specify an input/output list and invoke format control during their operations.

The two formatted data assignment statements are ENCODE and DECODE.

ENCODE AND DECODE STATEMENTS

The forms of the ENCODE and DECODE statements are

\[
\begin{align*}
\text{ENCODE} & \quad (n, f, d, e) \ [e] \\
\text{DECODE} & \quad (n, f, s, e) \ [d]
\end{align*}
\]

where

- \( n \) is the number of characters to be processed, specified by a nonzero integer expression not to exceed 152;
- \( f \) is a FORMAT identifier, except for an asterisk;
- \( d \) is the symbolic name of a destination variable, array element, or array where the \( n \) characters of \( e \) are packed (eight per word) by ENCODE;
sent is the symbolic name of the source variable, array element, or array where characters are unpacked and stored into dlist by DECODE; and

e and dlist
are lists specified the same as for formatted input/output statements. e is the list of items written to the destination entity; dlist is the list of items receiving the source entity.

The ENCODE statement

The ENCODE statement produces a sequence of \( n \) characters (packed eight per word) from values contained in the input list items specified in e under control of the format specification identified by \( f \). The character sequence is stored into a variable, array element or array identified by dent.

If \( n \) is not an integer multiple of eight, the last word in each record is padded with spaces to a word boundary. In effect, \( n \) is rounded up to be a multiple of eight.

Example:

e: array ZD(5):
    ZD(1) = 'THISbbbb'
    ZD(2) = 'MUSTbbbb'
    ZD(3) = 'HAVEbbbb'
    ZD(4) = 'FOURbbbb'
    ZD(5) = 'CHARbbbb'

f: FORMAT (5A4)

n: 20

dent: array ZE(3)

The sequence

ENCOD (20,1,ZE)ZD

produces

dent =
    ZE(1) = 'THISMUST'
    ZE(2) = 'HAVEFOUR'
    ZE(3) = 'CHARbbbb'

* SR-0009
sent E-8 J
where the asterisk and the X indicate the manner of editing,

\( h_i \) is any ASCII character listed in Appendix A as capable of internal representation, and

\( b \) is any nonzero, unsigned integer constant.

Examples:

*AN ASTERISK EDIT DESCRIPTOR*

-55X (moves current position 55 spaces to left)
The DECODE statement

The DECODE statement processes a sequence of \( n \) characters (packed eight per word) contained in the variable, array element, or array identified by \( s\text{ent} \) under control of the format specification identified by \( f \). The resulting values define the input list items specified in \( d\text{list} \).

If \( n \) is not an integer multiple of eight and the DECODE format calls for more than one DECODE record, the second and all subsequent DECODE records begin on a word boundary. In effect, \( n \) is rounded up to be a multiple of eight.

Example:

\begin{align*}
\text{sent:} & \quad \text{ZE:} \\
& \quad \text{ZE(1) = 'WHILETHI'} \\
& \quad \text{ZE(2) = 'ShHASbbP'} \\
& \quad \text{ZE(3) = 'IVEbbbbb'} \\
\hspace{1cm} n: & \quad 20 \\
\hspace{1cm} f: & \quad \text{FORMAT (5A5)} \\
\end{align*}

The sequence

\[ \text{DECODE (20,2,ZE)ZD} \]

produces

\[ \text{dlist = ZD(1) = 'WHILEbb'} \]
\[ \text{ZD(2) = 'THISbbbbb'} \]
\[ \text{ZD(3) = 'HASbbbbb'} \]
\[ \text{ZD(4) = 'FIVEbbbbb'} \]

EDIT DESCRIPTORS

The formats of obsolete edit descriptors are

\[ ^*h_1h_2\ldots h_n^* \text{ (asterisk)} \]

\[ [-b]X \]
RANF
(continued)

Method*
(continued)

Argument A is a dummy and is not used by the function. Use a floating-point argument to avoid extra pack and normalize instructions added by the compiler; where RANSET has not previously been called, the generative number is set to 9.

ENTRY POINT RANSET

FORTRAN Call

CALL RANSET(I)

The generative number (NUMBER) is reset to the integer specified by the argument I. Following a CALL RANSET(I), the repeated use of $X = \text{RANF}(1.0)$ generates a set of random numbers based on I. To repeat a set of random numbers, this generative number must be reset to the generative number of the original set. I should be an odd integer between 1 and $2^{30}$. An octal constant may also be used as the argument.

Ascent Calling
Sequence

RJ RANSET
EO *+2
CON I

ENTRY POINT RANGET

FORTRAN Call

CALL RANGET(I)

The generative number currently in use by RANF is returned in I. This is a 48-bit integer number with bits 59-49 set to 1. Use an 020 format to print. To repeat a set of random numbers, this generative number may be reentered with or without the mask by a CALL to RANSET.

Ascent Calling
Sequence

RJ RANGET
EO *+2
CON I
RANF

Purpose
To generate a uniformly distributed sequence of random numbers between 0 and 1 in floating-point format, by repeatedly using RANF as a function for the set. Call RANSET as a subroutine to change the generating number.

FORTRAN Function
X = RANF(A)

Entry Points
RANF, RANSET, RANGET

ENTRY POINT RANF

Ascent Calling Sequence
RJ RANF
BX6 XI
SA6 X

Normal Return
The results returned in XI are stored in X.

Error Message
None

Storage
26 locations

Accuracy
The \( \chi^2 \) test on 10,000 numbers was satisfactory.

Timing
6600: 13 \( \mu \)sec
7600: 3 \( \mu \)sec

Method*
Number_{i+1} = Number_i \times K \pmod{2^{48}} \) where Number and K are packed as floating integers.

\[ X = \text{Number}_{i+1}/2^{48} \]
\[ K = 2^{24} - 3 \]
BYTE
(continued)

TRY POINT GBYTE
(continued)

Call GBYTE(NPACK,ISAM,IBIT\-offset,NBITS in byte) to get NBITS after skipping IBIT\-offset bits in word NPACK.

Put the byte into ISAM right-adjusted, with the rest of cell ISAM set to zero. The maximum byte size is 60. The permissible range of IBIT\-offset is 0\-59.

Example

CALL GBYTE(NPA,ISA,6,6)

About 18 microseconds per byte. This is about three and a half times as fast as ENCODE/DECODE for handling 6\-bit characters, and about seven times as fast as ENCODE/DECODE when the latter uses an I4 format.
## GBYTE

**ENTRY POINTS** GBYTE, GBYTES, SBYTE, SBYTES

<table>
<thead>
<tr>
<th><strong>Language</strong></th>
<th>Ascent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage</strong></td>
<td>1248 locations</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>To provide the capability to format in terms of numbers of bits rather than numbers of characters.</td>
</tr>
</tbody>
</table>

**TRY POINT GBYTE**

<table>
<thead>
<tr>
<th><strong>Purpose</strong></th>
<th>To unpack bits or bytes from NPACK to ISAM.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORTRAN Reference</strong></td>
<td>CALL GBYTE(NPACK,ISAM,IBIT,NBITS)</td>
</tr>
</tbody>
</table>

**Ascent Calling Sequence**

- RJ GBYTE
- EQ **+5**
- CON NPACK
- CON ISAM
- CON IBIT
- CON NBITS

**Return Arguments** None

**Printed Message** None
In the example, the offset is 56 bits and byte size is 6 bits, so that the total number of bits (62) is greater than one word (60 bits). Since word boundaries are ignored in packing bytes, SBYTE continues packing into the next word.

About 18 microseconds per byte. This is about three and a half times as fast as ENCODE/DECODE for handling 6-bit characters, and about seven times as fast as ENCODE/DECODE when the latter uses an I4 format.
ENTRY POINT SBYTE

Purpose
To reverse the process that occurred in GBYTE.

FORTRAN Reference
CALL SBYTE(NPACK, ISAM, IBIT, NBITS)

Ascent Calling Sequence
RJ SBYTE
EQ *+5
CON NPACK
CON ISAM
CON IBIT
CON NBITS

Return Arguments
None

Error Message
None

Comments
The byte size in NPACK is cleared and the byte put in; data surrounding the packed byte are not affected.

Example

DIMENSION NA(2)

CALL SBYTE(NA, NB, 56, 6)

| 59 | 0 |
| NB | 00000000000000000077 |
| NA(1), NA(2) | 00000000000000000017 | 60000000000000000000 |
| 59 | 0 | 59 | 0 |

In binary the two words NA(1) and NA(2) look like this:

<table>
<thead>
<tr>
<th>NA(1)</th>
<th>NA(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>0 ... 00000111110000000000 ... 0</td>
</tr>
<tr>
<td>octal</td>
<td>1 7 6</td>
</tr>
</tbody>
</table>
Skip 3 bits as specified in the IBIT-offset, take a 6-bit byte (the two 7's), skip 9 bits (the next three 0's) as specified in NSKIP, and do this twice (ITER=2).

About 18 microseconds per byte. This is about three and a half times as fast as ENCODE/DECODE for handling 6-bit characters, and about seven times as fast as ENCODE/DECODE when the latter uses an I4 format.
ENTRY POINT GBYTES

Purpose
To get an ITER number of bytes from NPACK into the ISAM array.

FORTRAN Reference
CALL GBYTES(NPACK,ISAM,IBIT,NBITS,NSKIP,ITER)

Ascent Calling Sequence
RJ GBYTES
EQ *+7
CON NPACK
CON ISAM
CON IBIT
CON NBITS
CON NSKIP
CON ITER

Turn Arguments
None

Printed Message
None

Comments
Call GBYTES(NPACK,ISAM,IBIT-offset,NBITS,NSKIP,ITER). After the first byte, specified as in GBYTE, there is a skip of NSKIP bits. The next byte (of the same size and the same skip) then begins, and so on. The maximum byte size is 60 bits, but the skip can be longer than 60 bits. The permissible range of IBIT-offset is 0-59.

Example
DIMENSION ISB(2)
CALL GBYTES(NPB,ISB,3,6,9,2)

```
07700077000000000000
```

```
00000000000000000077 00000000000000000077
```

```
59 0
0 59 0
```
RANRD

Note: See BUFRD, p. 4.19, for complete description of buffered operations.

ENTRY POINTS BRANRD, RANCK, BRANST, BRANRL

language Ascent

storage 254₈ locations

purpose To provide a buffered I/O operation from the random file that resides on the drum (disk). The routine has true buffering capability on two channels.

Even-named records are sent to one of the two random files available and odd-named records to the other. Each record must have a unique name. An even name has bit 0 = 0; an odd name has bit 0 = 1. Most alphanumeric names will be odd-named files because the word will be blank-filled and bit 0 = 1. Integer names should have type integer variable names; if these are floating, the pack and normalize will force all integers to be even, since bit 0 = 0. Two channels transmit records simultaneously. Any record can be written, read, or rewritten at random to the file. Each record may have a different length and can be rewritten with varying lengths. The length of the record given on each first write must be the maximum length that will be used for that record. The maximum number of record names is 200. If more are required, use the ULIB version of BRANRD, where the number of file names may be changed. See the librarian in room 30. The records created by BRANRD are temporary and exist only during the execution of the job.
**ENTRY POINT SBYTES**

**Purpose**

To reverse the process that occurred in GBYTES.

**FORTRAN Reference**

CALL SBYTES(NPACK, ISAM, IBIT, NBITS, NSKIP, IITER)

**Ascent Calling Sequence**

- RJ SBYTES
- EQ *+7
- CON NPACK
- CON ISAM
- CON IBIT
- CON NBITS
- CON NSKIP
- CON ITER

**Return Arguments**

None

**Printed Message**

None

**Comments**

SBYTES uses the same arguments as GBYTES.

**Example**

```fortran
DIMENSION ISB(2)
CALL SBYTES(NPC, ISB, 45, 6, 3, 2)
```

<table>
<thead>
<tr>
<th>NPC</th>
<th>0000000000000000000077077</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISB(1), ISB(2)</td>
<td>00000000000000000000000077077</td>
</tr>
</tbody>
</table>

| 59 | 0 |

45 bits (IBIT-offset) are skipped in NPC. Then the right-most 6 bits (NBITS) in ISB(1) are packed into NPC, 3 bits are skipped in NPC (NSKIP), and the process is repeated (ITER=2) from ISB(2).

**Timing**

About 18 microseconds per byte. This is about three and a half times as fast as ENCODE/DECODE for handling 6-bit characters, and about seven times as fast as ENCODE/DECODE when the latter uses an I4 format.
BRANRD starts a buffered read from the random record NAME into ARRAY. The length of the record is LENGTH. Before the array is used, BRANCK should be called to assure completion of a read.

Timing

Depends on exact sequence of operations being performed and on amount of data being read.

ENTRY POINT BRANWT

Purpose

To start a write.

FORTRAN Reference

CALL BRANWT(NAME,ARRAY,LENGTH)

NAME

Any collection of numbers or characters identifying the record. If bit 0 = 1, the name is odd; if bit 0 = 0, the name is even. This can be any 60-bit configuration except all 0's.

ARRAY

First word address in memory of the array to be transmitted.

LENGTH

Number of words to be transmitted.

Ascent Calling Sequence

RJ   BRANRD
EQ   **+4
CON   NAME
CON   ARRAY
CON   LENGTH

Return Arguments

None
4.8
Auxiliary Library Subroutines

BRANRD
(continued)

ENTRY POINT BRANRD

Purpose
To start a read.

FORTRAN Reference
CALL BRANRD(NAME,ARRAY,LENGTH)

NAME
Any collection of numbers or characters identifying the record. If bit 0 = 1, the name is odd; if bit 0 = 0, the name is even. This can be any 60-bit configuration except all 0's.

ARRAY
First word address in memory of the array to be transmitted.

LENGTH
Number of words to be transmitted.

Ascent Calling Sequence
RJ       BRANRD
EQ       **+4
CON      NAME
CON      ARRAY
CON      LENGTH

Return Arguments
None

Printed Messages
ATTEMPT TO READ UNWRITTEN RECORD
ATTEMPT TO USE RECORD WITH LENGTH GT 1ST USE LENGTH
ATTEMPT ANOTHER OPERATION WITHOUT CHECKING LAST
CANNOT RESET FILE
CANNOT SET UP FILE
EOF ENCOUNTERED
NAME MUST BE NONZERO
RECORD LENGTH MUST BE GT 0
UNSUCCESSFUL READ OR WRITE
ENTRY POINT BRANCK

Purpose
To check for completion of the I/O operation and for transmission error.

FORTRAN Reference
CALL BRANCK(NAME)

NAME
NAME assigned to record.

Ascent Calling Sequence
RJ BRANCK
EQ *+2
CON NAME

Return Arguments
None

Printed Messages
EOF ENCOUNTERED
UNSUCCESSFUL READ OR WRITE

Comments
BRANCK will wait for completion of the I/O operation on the record called NAME. It will check the status of the operation. If the I/O was unsuccessful or the record was missing on the file, a message is printed and the program terminated. NAME is placed in X2 on termination. BRANCK can be called with a dummy name. If the record is not being read or written, or if the name is a dummy, the call is a NOP (no operation).

Note: BRANCK should be called to complete the operation on the last even-named record before the next even-named record is read or written.

Timing
Depends on exact sequence of operations and amount of computation done between last call to BRANRD/BRANWT and the call to BRANCK.
BRANRD
(continued)

ENTRY POINT BRANWT
(continued)

Printed Messages
ATTEMPT TO READ UNWRITTEN RECORD
ATTEMPT TO USE RECORD WITH LENGTH GT LST USE LENGTH
ATTEMPT ANOTHER OPERATION WITHOUT CHECKING LAST
CANNOT RESET FILE
CANNOT SET UP FILE
EOF ENCOUNTERED
NAME MUST BE NONZERO
RECORD LENGTH MUST BE GT 0
UNSUCCESSFUL READ OR WRITE

Comments
BRANWT starts a buffered write from ARRAY to a random record
called NAME. The length of the record written is LENGTH.
Before destroying the array, BRANCK should be called to
assure completion of the write.

Timing
Depends on exact sequence of operations being performed
and on amount of data being written.
4.72
Auxiliary Library Subroutines

IOPROC
(continued)

ENTRY POINT RDTAPE
(continued)

FORTRAN Reference
(continued)

NADDR
Address of the first word in the record, i.e., the first location to be used for storage.

NWDCNT
Number of words to be read or written from the record, or maximum possible record size.

Ascent Calling Sequence

| RJ | RDTAPE |
| EQ | **6 |
| CON | NUNIT |
| CON | MODE |
| CON | NTYPE |
| CON | NADDR |
| CON | NWDCNT |

Return Arguments
None

Printed Messages
ATTEMPT TO USE AN ILLEGAL MODE NUMBER
ATTEMPT TO USE ILLEGAL UNIT NUMBER

Comments
When reading an EOF with RDTAPE, the first word of the buffer read in for the EOF will contain a BCD 17.

Timing
Data dependent
Comments

This should not be used to backspace records written with the FORTRAN WRITE statement.

Timing

Data and device dependent

ENTRY POINT RDTAPE

Purpose

To read a record.

FORTRAN Reference

CALL RDTAPE(NUNIT,MODE,NTYPE,NADDR,NWDCNT)

NUNIT

A logical tape unit.

MODE†

0  Even parity (BCD mode), no character conversion.
1  Odd parity (binary mode), no character conversion.
2  Even parity (BCD mode), conversion of external BCD to display code (or vice versa if writing a record).

NTYPE

0  Used for all tape writes and to read tapes created on the NCAR system. The NWDS from IOWAIT ignores any partial words. (1 provides the same option.)

2  Used to read records from another computer. The NWDS from CALL IOWAIT includes a count for any partial word at the end of the record.

Note: Add 4 to any of the above options to ignore read parity error. The system will not try to correct this error. Thus a type 4 is the same as type 0 but with no reread on parity errors.

† Character conversion applies only to tapes. The drum never does character conversion on a read or write.
Auxiliary Library Subroutines

IOPROC
(continued)

ENTRY POINT WRTAPE
(continued)

Ascent Calling Sequence

<table>
<thead>
<tr>
<th>RJ</th>
<th>WRTAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>++6</td>
</tr>
<tr>
<td>CON</td>
<td>NUNIT</td>
</tr>
<tr>
<td>CON</td>
<td>MODE</td>
</tr>
<tr>
<td>CON</td>
<td>NTYPE</td>
</tr>
<tr>
<td>CON</td>
<td>NADDR</td>
</tr>
<tr>
<td>CON</td>
<td>NWDCNT</td>
</tr>
</tbody>
</table>

Return Arguments
None

Printed Messages
ATTEMPT TO USE AN ILLEGAL MODE NUMBER
ATTEMPT TO USE ILLEGAL UNIT NUMBER

Comments
None

Timing
Data dependent
ENTRY POINT WRTAPE

Purpose
To write a record.

FORTRAN Reference
CALL WRTAPE(NUNIT,MODE,NTYPE,NADDR,NWDCNT)

NUNIT
A logical tape unit.

MODE†
0 Even parity (BCD mode), no character conversion.
1 Odd parity (binary mode), no character conversion.
2 Even parity (BCD mode), conversion of external BCD to display code (or vice versa if writing a record).

NTYPE
0 Used for all tape writes and to read tapes created on the NCAR system. The NWDS from IOWAIT ignores any partial words. (1 provides the same option.)
2 Used to read records from another computer. The NWDS from CALL IOWAIT includes a count for any partial word at the end of the record.

Note: Add 4 to any of the above options to ignore read parity error. The system will not try to correct this error. Thus a type 4 is the same as type 0 but with no reread on parity errors.

NADDR
Address of the first word in the record, i.e., the first location to be used for storage.

NWDCNT
Number of words to be read or written from the record, or maximum possible record size.

† Character conversion applies only to tapes. The drum never does character conversion on a read or write.
JOBID

ENTRY POINT JOBID

Language Ascent

Storage 108 locations

Purpose To return to the user job identification data in display code.

FORTRAN Reference CALL JOBID(IWHERE)

Ascent Calling Sequence

RJ JOBID
JP *+2
CON IWHERE

Return Arguments IWHERE is a four-word array in which the following job identification is returned:

IWHERE(1) Sequence number
IWHERE(2) Name
IWHERE(3) Scientist number
IWHERE(4) Project number

All items are left-justified with blank fill.

Printed Message None

Comments The central monitor reads a table from the user's buffer area containing this information.

Timing About 10 microseconds
ENTRY POINT IOWAIT

Purpose
To wait for the completion of a read or write.

FORTRAN Reference
CALL IOWAIT(NUNIT,NSTATE,NWDS)

NUNIT
A logical tape unit

NSTATE
Status:
0  good read or write
1  EOF
2  parity error on read or unable to write
3  end of tape

NWDS
Number of 6600/7600 60-bit words read or written.

It is possible to call IOWAIT and get a good return before a unit has been used.

Ascent Calling Sequence
RJ       IOWAIT
EQ       **+4
CON      NUNIT
CON      NSTATE
CON      NWDS

Return Arguments
None

Printed Message
ATTEMPT TO USE ILLEGAL UNIT NUMBER

Comments
• After a read or write, IOWAIT must be called before reading or writing again from the same unit.

• In a call to RDTAPE or WRTAPE, the array size must be equal to or larger than the NWDCNT specified.

Timing
Data and device dependent
THE IFTEN PrePROCESSOR
David J. Kennison, Author

June 15, 1983
# Table of Contents

## INTRODUCTION

1

## FEATURES OF IFTRAN

<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String Replacement</td>
<td>1</td>
</tr>
<tr>
<td>Arithmetic Evaluation</td>
<td>1</td>
</tr>
<tr>
<td>Conditional Compilation</td>
<td>1</td>
</tr>
<tr>
<td>Code-Block Definition</td>
<td>2</td>
</tr>
<tr>
<td>FORTRAN Language Extensions</td>
<td>2</td>
</tr>
<tr>
<td>Free-form Input</td>
<td>2</td>
</tr>
<tr>
<td>Extended Comments</td>
<td>2</td>
</tr>
<tr>
<td>Alternate Input Units</td>
<td>2</td>
</tr>
<tr>
<td>Print Output Controls</td>
<td>2</td>
</tr>
<tr>
<td>FORTRAN Output Controls</td>
<td>2</td>
</tr>
<tr>
<td>Treatment of Unknown Commands</td>
<td>2</td>
</tr>
</tbody>
</table>

## HISTORY OF IFTRAN

2

## THE FORM OF THE IFTRAN INPUT FILE

3

## THE FORM OF IFTRAN COMMANDS

3

## THE FORM OF IFTRAN STATEMENTS

3

## THE IFTRAN COMMANDS

4

- `ACTIVATE alphabetic-character`
- `CALL save-block-name [,string-replacements]`
- `CONTINUATION [continuation-character]`
- `DEACTIVATE alphabetic-character`
- `DSOK`
- `EJECT`
- `ELIMINATE alphabetic-character`
- `ELSEIF expression`
- `ELSE`
- `END`
- `ENDIF`
- `FINISH`
- `IF expression`
- `INCLUDE unit-number [,card-length] [,R]`
- `TITLE any-title`
- `KWOK`
- `LIST`
- `NODSOK`
- `NOWKOK`
- `NOLIST`
- `NOREPLACE`
- `OPTION keyword[=value] [[,]keyword[=value]]`
- `ORIF expression`
- `OTHERWISE`

---

**IFTRAN**

Table of Contents - 1

June 1983
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE IFTRAN STATEMENTS</td>
<td>8</td>
</tr>
<tr>
<td>BLOCK ... END BLOCK</td>
<td>8</td>
</tr>
<tr>
<td>DO ... END DO</td>
<td>9</td>
</tr>
<tr>
<td>FOR ... END FOR</td>
<td>9</td>
</tr>
<tr>
<td>IF ... ORIF ... ELSE ... END IF</td>
<td>10</td>
</tr>
<tr>
<td>INVOKE</td>
<td>10</td>
</tr>
<tr>
<td>LOOP ... END LOOP</td>
<td>10</td>
</tr>
<tr>
<td>REPEAT ... UNTIL</td>
<td>11</td>
</tr>
<tr>
<td>WHILE ... END WHILE</td>
<td>11</td>
</tr>
<tr>
<td>STRING REPLACEMENT</td>
<td>11</td>
</tr>
<tr>
<td>THE EFFECT OF THE REPLACE-TOKENS FLAG</td>
<td>12</td>
</tr>
<tr>
<td>RESOLUTION OF AMBIGUITIES</td>
<td>13</td>
</tr>
<tr>
<td>THE EVALUATOR</td>
<td>13</td>
</tr>
<tr>
<td>OPTIONS and KEYWORDS</td>
<td>14</td>
</tr>
<tr>
<td>THE IFTRAN PRINT FILE</td>
<td>19</td>
</tr>
<tr>
<td>USING IFTRAN AT NCAR</td>
<td>21</td>
</tr>
<tr>
<td>USING IFTRAN ON THE CRAY-1</td>
<td>21</td>
</tr>
<tr>
<td>Examples</td>
<td>23</td>
</tr>
<tr>
<td>USING IFTRAN ON THE IBM 4341</td>
<td>24</td>
</tr>
<tr>
<td>USING IFTRAN ON THE PDP 11/70</td>
<td>26</td>
</tr>
<tr>
<td>IMPLEMENTING IFTRAN ELSEWHERE</td>
<td>28</td>
</tr>
</tbody>
</table>

IFTRAN Table of Contents - 2  
June 1983
The IFTRAN preprocessor is a FORTRAN program which reads an input file containing a program written in the IFTRAN language (FORTRAN plus extensions) and writes two output files, one containing an indented, easy-to-read listing of the input file, and the other containing a FORTRAN translation of the input file.

IFTRAN is easier to write and to read than basic FORTRAN. The code to solve a given problem can be expressed in a way more closely resembling the logical description of the solution. Some of the annoying bookkeeping details of FORTRAN can be avoided. Various features allow one to parameterize one's code and to express variant versions of the code in the same master deck, thus simplifying maintenance. The time required to check out an IFTRAN program is normally considerably less than that required by an equivalent FORTRAN program.

Because the IFTRAN preprocessor is highly portable, code written in the IFTRAN language is also highly portable.

In the following paragraphs are outlined the many useful capabilities of IFTRAN.

**Features of IFTRAN**

**String Replacement**

The ability to specify string replacements makes it possible to define "pre-compile-time variables" and to parameterize one's code. For example, the basic dimensions of a model can be declared once and for all by a few string replacements at the beginning of the input file. See the section **STRING REPLACEMENT**, the .REPLACE and .NOREPLACE commands, and the .OPTION command keywords CB, LR, RS, and RT.

**Arithmetic Evaluation**

Expressions involving "pre-compile-time variables", to be evaluated by IFTRAN, may be used. See the section **THE EVALUATOR** and the .OPTION command keywords ES and ET.

**Conditional Compilation**

Two types of conditional compilation are provided. First, input lines with a particular alphabetic character in column 1 may be "activated" (character replaced by a blank), "deactivated" (character replaced by a C), or "eliminated" (entire line ignored): the state of any alphabetic character may easily be changed. Second, a construct is provided to allow blocks of code to be included or excluded based on the value of relational expressions which are evaluated at pre-compile time. See the .ACTIVATE, .DEACTIVATE, .ELIMINATE, and .IF commands.
Blocks of code may be defined and saved for later insertion at selected points in the subsequent code. For example, a COMMON block can be defined in one place and then used as necessary. This has the advantage that a change in the definition need only be made in one place. See the .CALL, .SAVE, and .USE commands.

Various "structured programming" constructs are provided: DO/ENDO, FOR/ENDDR, IF/ENDIF, LOOP/ENDLOOP, REPEAT/UNTIL, WHILE/ENDWHILE, BLOCK/ENDBLOCK/INVOKE. See the section IFTRAN STATEMENTS, the .KWO and .NOKWO commands, and the .OPTION command keyword NW.

"Tab" and "continuation" characters may be defined, allowing the use of somewhat more free-form input than that required by the rigid FORTRAN conventions. See the .CONTINUATION and .TAB commands and the .OPTION command keywords CC and TC.

An extended-comment character may be defined, allowing comments to be placed at the end of FORTRAN statements. See the .XCOM command and the .OPTION command keyword XU.

Input may be included from alternate input units. See the .INCLUDE command.

Various controls allow one to modify the appearance of the print output file. See the .DSOK, .EJECT, .LIST, .NOSOK, .NOLIST, and .TITLE commands and the .OPTION command keywords CB, DS, IL, LR, FC, PF, PI, PN, and PR.

Various controls allow one to modify the appearance of the FORTRAN output file. See the .OPTION command keywords CB, IF, IL, LS, OC, OF, OI, ON, and OS.

Unknown commands may be treated as errors or as comments. See the .OPTION command keyword UC.

IFTRAN was originally written by E. F. Miller of General Research Corporation. In July, 1975, John Gary, of NCAR, received an improved version of it from Martin J. Cohen, of Technology Service Corporation, and gave a copy to Tom Wright, who implemented it at NCAR, made several improvements, and was largely responsible for its popularity here. Lofton Henderson assumed responsibility for the maintenance of IFTRAN sometime in 1977, corrected several errors, and compiled a write-up. Most of that write-up is still correct for the current IFTRAN; it contains many useful examples (some of which are reproduced here). In 1981, Dave Kennison completely re-wrote the code, fixing errors, making it cleaner, faster, and smaller and adding many new features. One feature, the ability to "trace" string replacements as they were done, was removed; it

IFTRAN

-2-  

June 1983
was principally of interest to the maintainer of IFTRAN. Work with IFTRAN is continuing and suggestions are solicited.

THE FORM OF THE IFTRAN INPUT FILE

The IFTRAN input file consists of 80-column card images containing IFTRAN commands, IFTRAN statements, FORTRAN statements, and FORTRAN comments. Columns 1 through 72 are used for this purpose. Non-blank characters in columns 73 through 80 may be used for sequencing information, but are otherwise ignored.

Anything not recognizable as an IFTRAN command, an IFTRAN statement, or a FORTRAN comment is simply passed through unchanged (except perhaps by string replacement) to the card output file.

THE FORM OF IFTRAN COMMANDS

IFTRAN commands have a period, an asterisk, or a "deactivated" alphabetic character in column 1 and an alphabetic character in column 2. The characters in columns 2 and following - up to, but not including, the first blank - form a "command keyword" identifying the command. Only as many characters of the command keyword as are needed to uniquely identify it need be given, but, for improved readability, extra characters may be used. At most five characters are checked for correctness; the sixth and following are ignored. (In fact, if the keyword is intrinsically shorter than five characters, characters past the intrinsic length are ignored.)

Parameters required by a command, if any, are placed after the command keyword, separated from it by at least one blank and from each other (usually) by commas. The exact form of the parameters depends on the command used. Extra blanks may be used as desired between, but not within, syntactic elements, to improve readability.

A command longer than 72 characters may be continued in columns 2 through 72 of following cards by putting a comma in column 1 of those cards.

Unrecognized commands yield an error message (which can be turned off by a ".OPTION UC=0") and are otherwise treated as comments.

THE FORM OF IFTRAN STATMENTS

IFTRAN statements are written just like FORTRAN statements, in columns 7 through 72. Continuation cards for statements which won't fit on a single card are formed in the same way as in FORTRAN. A single statement may include up to 19 continuation cards.

As in FORTRAN statements, blanks are ignored, with two exceptions: The various clauses of a FOR statement must be separated by at least one blank. Also, if the "key-
word form of IFTRAN statements has been enabled (by either of the commands .KWOK or "OPTION KW=1"), there must be at least one blank following any IFTRAN keyword.

Any IFTRAN statement may have a statement label in columns 1 through 5. This is somewhat contrary to the spirit of structured programming, of course, but there are times when a statement label simply cannot reasonably be avoided. In the resulting FORTRAN, the statement label will appear on a CONTINUE statement in the appropriate place.

THE IFTRAN COMMANDS

Descriptions of all the IFTRAN commands, in alphabetical order, follow. The characters [ and ] are used to enclose optional parameters.

.ACTIVATE alphabetic-character

"Activates" a particular alphabetic character, which means that following cards with that character in column 1 will have the character in column 1 changed to a blank. See also the commands .DEACTIVATE and .ELIMINATE. All alphabetic characters are initially deactivated.

.CALL save-block-name [,string-replacements]

Has precisely the same effect as a "USE save-block-name", preceded by a "REPLACE [string-replacements]" if string replacements are specified. The "string-replacements" are separated from the "save-block-name" and from each other by commas.

.CONTINUATION [continuation-character]

Specifies a character to be used as a continuation character in FORTRAN and IFTRAN statements. Any input line with this character in column 1 will be modified by moving characters from columns 1 through 67 to columns 6 through 72 and replacing the characters in columns 1 through 5 by blanks. Using this command without "continuation-character" turns the feature off (which is the default state). Do not attempt to specify an alphabetic, an asterisk, a period, a comma, or a zero as the continuation character. See also the command .TAB and the .OPTION command keywords CC and TC.

.DEACTIVATE alphabetic-character

"Deactivates" a particular alphabetic character, which means that following cards with that character in column 1 will have the character in column 1 changed to a C. See also the commands .ACTIVATE and .ELIMINATE. All alphabetic characters are initially deactivated.

.DSOK

Turns on the "double-space" flag, which causes a blank line to be inserted in the print output file between any two lines which are of different types. There are four types, corresponding to the four different kinds of input cards: IFTRAN commands like .USE or .OPTION, IFTRAN statements like WHILE or INVOKE, FORTRAN statements, and comments. This flag is initially on. See also the com-
.EJECT
Causes a page eject at the current position in the print output file. Consecutive .EJECT cards cause only a single eject.

.ELIMINATE alphabetic-character
"Eliminates" a particular alphabetic character, which means that following cards with that character in column 1 will be ignored. See also the commands .ACTIVATE and .DEACTIVATE. All alphabetic characters are initially deactivated. The abbreviation .EL implies .ELIMINATE rather than .ELSE.

.ELSEIF expression
Has the same effect as ".ORIF expression" (which see, below). Included for those (like myself) who have trouble remembering which word to use.

.ELSE
Marks the beginning of a block of code to be used if the expressions in all .IF and .ORIF commands of a particular .IF construct are false. See the command .IF, below. The abbreviation .EL implies .ELIMINATE rather than .ELSE.

.END
Marks the end of a block of cards to be SAVED. See the command .SAVE. The abbreviation .EN implies .END rather than .ENDIF.

.ENDIF
Marks the end of a .IF construct. See the command .IF. The abbreviation .EN implies .END rather than .ENDIF.

.FINISH
Marks the end of an IFTRAN input file.

.IF expression
Introduces a .IF construct, which has the form:

```plaintext
.IF  e0
   (code to use if e0 is true)
.ORIF el
   (code to use if e0 is false and el is true)
.ORIF e2
   (code to use if e0 and el are false and e2 is true)
...
.
.
.ELSE
   (code to use if all ek above are false)
.ENDIF
```

The .IF and .ENDIF commands are required; .ORIF and .ELSE commands are optional. (".ELSEIF" may be used as a synonym for ".ORIF"). Each ek is an expression, formed in the usual way using integers and logical constants (or,
more probably, "pre-compile-time variables" defined by string-replacement specifiers on preceding .REPLACE commands) as operands and the usual set of FORTRAN arithmetic and logical operators. An arithmetic expression is considered to be true if it has a non-zero value, false if it has a zero value. Logical constants are given integer values (1 means TRUE and 0 means FALSE) and the logical operators are set up in a manner consistent with this. If the expression contains operands, its first and last characters must be the "evaluation-starts" and "evaluation-terminates" characters. See the section entitled THE EVALUATOR, below, for a complete discussion of expression evaluation.

Each code block in a .IF construct may contain other .IF constructs; they may be nested up to 16 deep.

more probably, "pre-compile-time variables" defined by string-replacement specifiers on preceding .REPLACE commands) as operands and the usual set of FORTRAN arithmetic and logical operators. An arithmetic expression is considered to be true if it has a non-zero value, false if it has a zero value. Logical constants are given integer values (1 means TRUE and 0 means FALSE) and the logical operators are set up in a manner consistent with this. If the expression contains operands, its first and last characters must be the "evaluation-starts" and "evaluation-terminates" characters. See the section entitled THE EVALUATOR, below, for a complete discussion of expression evaluation.

Each code block in a .IF construct may contain other .IF constructs; they may be nested up to 16 deep.

more probably, "pre-compile-time variables" defined by string-replacement specifiers on preceding .REPLACE commands) as operands and the usual set of FORTRAN arithmetic and logical operators. An arithmetic expression is considered to be true if it has a non-zero value, false if it has a zero value. Logical constants are given integer values (1 means TRUE and 0 means FALSE) and the logical operators are set up in a manner consistent with this. If the expression contains operands, its first and last characters must be the "evaluation-starts" and "evaluation-terminates" characters. See the section entitled THE EVALUATOR, below, for a complete discussion of expression evaluation.

Each code block in a .IF construct may contain other .IF constructs; they may be nested up to 16 deep.

more probably, "pre-compile-time variables" defined by string-replacement specifiers on preceding .REPLACE commands) as operands and the usual set of FORTRAN arithmetic and logical operators. An arithmetic expression is considered to be true if it has a non-zero value, false if it has a zero value. Logical constants are given integer values (1 means TRUE and 0 means FALSE) and the logical operators are set up in a manner consistent with this. If the expression contains operands, its first and last characters must be the "evaluation-starts" and "evaluation-terminates" characters. See the section entitled THE EVALUATOR, below, for a complete discussion of expression evaluation.

Each code block in a .IF construct may contain other .IF constructs; they may be nested up to 16 deep.
and the statement label run together and that BLOCK DATA statements must be written with BLOCK and DATA run together. The use of this feature is not highly recommended. See also the command .NOKWOK and the .OPTION command keyword KW.

**.LIST**

Turns on a flag enabling output to the IFTRAN print file. This flag is initially on. See also the command .NOLIST and the .OPTION command keyword PR.

**.MODOK**

Turns off the "double-space" flag. See also the command .DSOK and the .OPTION command keyword DS.

**.NOKWOK**

Suspends the recognition of IFTRAN statements written in "keyword form" (which is the default, anyway). See also the command .KWOK and the .OPTION command keyword KW.

**.NOLIST**

Suspends the generation of lines in the print output file. See also the command .LIST and the .OPTION command keyword PR.

**.NOREPLACE**

Suspends the replacement of strings. See the command .REPLACE. All string replacements currently defined remain defined; a subsequent .REPLACE may be used to cause resumption of string replacement. See also the .OPTION command keyword RS.

**.OPTION**

```
[ keyword[=value] ]
``` 

Used to change the state of various internal flags controlling the behavior of IFTRAN. See the section OPTIONS, below, for a complete list of the keywords which may be given and their uses.

**.ORIF expression**

Marks the beginning of a block of code to be used if the expressions in all preceding .IF and .ORIF commands in the same .IF construct are false and the expression in this one is true. Note that there must be no blank between .OR and IF. See .IF, above.

**.OTHERWISE**

A synonym for .ELSE (q.v., above).

**.REPLACE [string-replacements]**

This command is used to activate string replacement and to define string replacements to be done. Each "string-replacement" has the form:

```
/ original-string / replacement-string /
```

where / represents any character which does not occur in either of the two strings. The "string-replacements" are separated from each other by commas. See the section STRING REPLACEMENT, the command .NOREPLACE, and the .OPTION command keyword RS.
.SAVE save-block-name
To save a block of code for later insertion at one or more points in the code, use the sequence:

```
.SAVE save-block-name
... block of code ...
.END
```

Subsequently, the commands ".USE save-block-name" (q.v., below) and ".CALL save-block-name" (above) may be used to retrieve the saved block and insert it in the code. This feature is most commonly used to set up COMMON blocks.

The block of code SAVED may not contain another .SAVE command, a .USE command, or a .INCLUDE command. This restriction will be relaxed in the future if demand warrants.

.TAB [tab-character]
Specifies a character to be used as a tab character in FORTRAN and IFTRAN statements. Any input line with this character in column 1 will be modified by moving characters 2 through 67 to columns 7 through 72 and replacing characters 1 through 6 by blanks. Using this command without "tab-character" turns the feature off (which is the default state). Do not attempt to specify an alphabetic, an asterisk, a period, or a comma as the tab character. See also the command .CONTINUATION and the .OPTION command keywords CC and TC.

.USE save-block-name
Causes the block of code specified by "save-block-name" to be retrieved and placed in the code after the .USE command. See .SAVE, above.

.XCOM [extended-comment-character]
If an "extended-comment-character" is specified, it may subsequently be used within any IFTRAN command, IFTRAN statement, or FORTRAN statement to say that the rest of the card image in which the character occurs is a comment, to be ignored. If the .XCOM statement appears without "extended-comment-character", the feature is turned off (which is the default state). See also the .OPTION command keyword XC.

IFTRAN Statements
The IFTRAN statements are used to implement extensions of the FORTRAN language which make it easier for the programmer to write his code and encourage structured-programming techniques. The constructs provided are described below. The characters [ and ] are used to indicate optional portions of the statements.
The BLOCK construct is used to define "internal subroutines" for use within a FORTRAN routine. See the INVOKE statement, below.

The DO construct simply provides a more convenient way of writing a FORTRAN DO, relieving the user from the necessity of coming up with a unique statement label.

The FOR construct provides a generalized looping capability. "il" is the initial value of the loop variable "i". "i2", if given, is the last value of the loop variable. "i3", if given, is the loop-variable increment; 1 is the default value. "we", if given, is a logical expression to be tested before each execution of the loop; if the condition is false, the loop will be terminated. "uc", if given, is a logical expression to be tested after each execution of the loop; if the condition is true, the loop will be terminated. "ni", if given, is an expression specifying the next value of "i" for which the loop is to be executed (like "i+1", for example); if this feature is used, no TO or BY clauses may be specified. STEP may be used as a synonym for BY. The various clauses in a FOR statement must be separated from each other by at least one blank.
The IF construct is much like the FORTRAN-77 block-IF, but has a slightly different syntax; the word THEN is omitted from the IF statement. The word ELSEIF may be used in place of ORIF. FORTRAN-77 block-IF's may be used in addition to IPTRAN block-IF's; no problems will result.

**INVOKE**

**INVOKE (block-name)**

The INVOKE statement is used to cause execution of the block of code specified by "block-name". The nature of the FORTRAN used to implement this feature is such that all invocations of a given block must appear prior to the occurrence of the BLOCK/END BLOCK defining it, in order that the "assigned GO TO" used to return from the block be complete. Also, because of the FORTRAN-77 rule banning extended ranges, a block should not be INVOKEd from inside a DO loop; this restriction does not apply to FORTRAN-66. "block-name" may be any desired string of characters; if the last three characters are ",NR", no code is generated for return from the block; this is useful for error-exit blocks.

**LOOP ... END LOOP**

**LOOP**

... code to be executed repetitively ...

**END LOOP**
The LOOP construct provides a way of setting up an infinite loop. Within the loop, the statements:

```
EXIT and EXIT IF (logical-expression)
```

may be used to cause a transfer of control to the first statement following the loop. Note that there is no way of specifying the level of the LOOP construct to be exited; an EXIT statement inside two nested LOOPS will cause only the innermost to be exited.

**REPEAT ... UNTIL.**

```
REPEAT
... code to be executed repetitively ...
UNTIL (logical-expression)
```

The REPEAT construct is used to loop through a section of code, testing a logical expression at the end of each iteration to see if looping is to continue. The code in the loop is always executed at least once.

**WHILE ... END WHILE.**

```
WHILE (logical-expression)
... code to be executed repetitively ...
END WHILE
```

The WHILE construct is used to loop through a section of code, testing a logical expression at the beginning of each iteration to see if looping is to continue. If logical-expression is initially false, the code in the loop is not executed even once.

**STRING REPLACEMENT.**

String replacements defined by .REPLACE commands are done (while string replacement is turned on) in all FORTRAN and IFTRAN statements, in the entire parameter string of .IF, .INCLUDE, .OPTION, .ORIF, .SAVE, and .USE commands, in the "save-block-name" on a .CALL command, and in the "replacement-string" portions of string replacement specifiers on subsequent .CALL and .REPLACE commands.

A complete FORTRAN or IFTRAN statement consists of the characters from columns 1 through 72 of its first card, plus those from columns 7 through 72 of continuation cards, if any. Statements split across two or more card
images are put back together before string replacement is
done. For example, the IFTRAN statements:

```
.REPL /361 X/362 Y/ , /A+B/B+C/
361 X=Y+Z+ ... ... ... ... A+
1B
```

(pretending that "A+" occurs in columns 71 and 72) would produce the output:

```
362 Y=Y+Z+ ... ... ... ... B+
+C
```

Note that continuation cards generated by IFTRAN have a plus sign in column 6. There is no longer any restriction on generating new continuation cards, as there was in older versions of IFTRAN.

Statements are examined from left to right for strings to be replaced. Once a replacement has been done, the search for other strings to be replaced resumes with the first character following the last replacement character. Note that there is no re-scan of replacement strings.

String replacement is considerably faster in this version of IFTRAN than it was in older versions. It is now quite practical to leave string replacement turned on throughout the bulk of the code, particularly if the "replace-tokens" flag can be turned on (see the keyword RT in the section OPTIONS, below).

**THE EFFECT OF THE REPLACE-TOKENS FLAG**

When this flag is on, IFTRAN checks for replacements in a given statement only where it finds an alphabetic character or a dollar sign immediately following a character which is neither alphabetic nor a dollar sign. For example, the input:

```
.OP  RT
.RE /N/100/
.DIMENSION A(N)
```

would yield a DIMENSION A(100) statement in the FORTRAN output file.
Note that only the third occurrence of N is replaced by the integer 100, because it is the only one which follows a character which is neither an alphabetic nor a dollar sign. Actually, in this case, IFTRAN has checked for string replacements beginning at the D, at the A, and at the third N.

Note that, when the "replace-tokens" flag is turned on, a string replacement defined by "/4096/TTTT/", attempting to replace the constant 4096 by the variable name TTTT, would not work, because 4 is neither alphabetic nor a dollar sign.

This feature does not quite provide what its name might be taken to imply; no full-fledged FORTRAN token scan is attempted. Using it imposes certain obvious restrictions on the way in which string replacements are set up, but has advantages, both in speed and in the avoidance of replacements being done where they are not wanted.

RESOLUTION OF AMBIGUITIES

If you define a string replacement to replace A by something and later define a new string replacement replacing A by something else, the second definition replaces the first.

If you define a string replacement to replace A and another to replace AB and the string AB occurs in the code, the replacement of AB will be done; longer replacements are preferred. An older version of IFTRAN was set up to prefer that replacement which had been defined first. This is no longer the case.

If you define a string replacement to replace AB and another to replace BC and the string ABC occurs in your code, AB is replaced. This is not seen as an ambiguity.

THE EVALUATOR

If string replacement is turned on, strings starting with the current "evaluation-starts" character and ending with the current "evaluation-terminates" character (see the keywords ES and ET in the section OPTIONS, below) will be replaced by an integer value. Such a string must be a legal expression. String-replacements (if any) occurring in the string to be evaluated will be performed before evaluation.

Operands may be integer or logical constants (or "precompile-time variables", which, after string replacement, become integer or logical constants). .TRUE. may be abbreviated .T. and is equivalent to the integer 1; .FALSE. may be abbreviated .F. and is equivalent to the integer 0.
The operators +, -, *, /, **, .LT., .LE., .GT., .GE.,
.EQ., .NE., .NOT., .AND., .OR., and .XOR. may be used.
The logical operators have been set up so as to treat
non-zero operands as being true and zero operands as be-
ing false and to generate the integer 1 for true or 0 for
false.

Evaluation is done as it would be in FORTRAN. Grouping
parentheses may be used freely. Some functions are pro-
vided. ABS(I) gives the absolute value of I.  
MAX(I1,I2,...) gives the largest and MIN(I1,I2,...) the
smallest of the arguments. MOD(I1,I2) and SIGN(I1,I2)
work as in FORTRAN.

As mentioned above, string replacement must be turned on
in order for evaluation to be done. In practice, this is
not a serious restriction, since strings to be evaluated
will usually contain "pre-compile-time variables" defined
by string replacements.

String replacement is now performed on the replacement-
string parts of string replacement specifiers on .REPLACE
commands. Consider the following input sequence:

```
.OPTI ES=[ ET=]
.REPL /$IDIM/100/
.REPL /$JDIM/200/
.REPL -$IDO2-[/$IDIM/2]-
.REPL /$I2JD/[$IDO2*$JDIM]/
```

```
DIMENSION A($IDIM,$JDIM),B($IDO2),C($I2JD)
```

(Note the use of minus signs, rather than slashes, in the
third .REPL, needed because of the slash used in the
replacement-string part.) These cards would be modified
by string replacement and evaluation as follows:

```
.REPL /$IDIM/100/
.REPL /$JDIM/200/
.REPL -$IDO2-50-
.REPL /$I2JD/10000/
```

```
DIMENSION A(100,200),B(50),C(10000)
```

OPTIONS The .OPTION command is used to change the setting of
flags controlling the behavior of IFTRAN and has the for-
mat:

IFTRAN -14- June 1983
Each parameter "kwk[=vlk]" specifies the assignment of a new value to some internal IFTRAN control. Each "kwk" is one of the keywords described below. Each "vlk" is either an integer, the letter T, meaning TRUE and equivalent to the integer 1, the letter F, meaning FALSE and equivalent to the integer 0, or a single character. If "=vlk" is omitted, the parameter specified by "kwk" is given the value 1, if it would naturally have an integer value, T if it would naturally have a logical value, or "null" if it would naturally have a character value.

There must be at least one blank between .OPTION and the first parameter. If the comma is omitted between one parameter and the next, there must be at least one blank in that position. Other blanks may be used as desired between, but not within, syntactic elements.

The Keyword CB
CB (default value 0) is the "Cull-Blanks" flag. If this flag is set non-zero, all blanks in columns 7 and following, up to the first non-blank, of FORTRAN and IFTRAN statements will be culled, the object being to remove indenting already in the IFTRAN source deck so that the IFTRAN listing and the FORTRAN output file will be properly indented. The effect will not be seen in the IFTRAN listing unless the parameter LR=1 is also used.

The Keyword CC
CC (default value "null") is the "Continuation Character". If this character is set non-null, it may be used in column 1 of input statements to indicate that the following 66 columns represent a continuation of the statement on the previous card. See also the command .CONTINUATION.

The Keyword DS
DS (default value 1) is the "Double-Space" flag. If it has the value 1, the listing produced by IFTRAN will have a blank line between any two lines which differ in type. There are four types, corresponding to the four different kinds of input cards: IFTRAN commands like .USE or .OPTION, IFTRAN statements like WHILE or INVOKE, FORTRAN statements, and comments. See also the commands .DSOK and .NDSOK.

The Keyword ES
ES (default value "null") is the "Evaluation-Starts" character. If it is given a non-null value, the character may be used in IFTRAN commands, IFTRAN statements, and FORTRAN statements to indicate that a string to be evaluated follows.
The Keyword ET ET (default value "null") is the "Evaluation-Terminates" character. If it is given a non-null value, it may be used to indicate the end of strings preceded by the ES character, to be evaluated. If the ES character is specified and the ET character is not, the ES character is used both to start and to terminate strings to be evaluated.

The Keyword IF IF (default value 0) is the "Indent-FORTRAN" flag. If it is given a non-zero value, the FORTRAN output produced by IFTRAN is indented by that many columns per nesting level. Use of this parameter is currently discouraged because the indenting is done in such a way as to sometimes introduce FORTRAN errors in statements containing character strings of the form "nH..." or "'...". Use at your own risk.

The Keyword IL IL (default value -4) is the "Indent-Listing" flag. If it is given a non-zero magnitude, the print output produced by IFTRAN is indented by that many columns per nesting level. Negating the value indicates that vertical lines, made up of dots, are to be used to show the range of IFTRAN constructs. See also the keyword CB, above.

The Keyword KW KW (default value 0) is the so-called "Keyword-form" flag. If it is given the value 1, parentheses may be omitted from IFTRAN constructs. There must be at least one blank between a keyword and a following expression ("WHILE I.EQ.3", for example). When this flag is turned on, FORTRAN DO statements must be written with no blank between DO and the statement label and BLOCK DATA statements must be written with no blank between BLOCK and DATA. See also the commands .KWOK and .NOKWOK.

The Keyword LI LI (default value -1) is the "Label Increment". See LS, below.

The Keyword LR LR (default value 0) is the "List-Replacements" flag. Normally, the print output from IFTRAN shows the source code in its original state, with no string replacements or blank-culling done. If this flag is set non-zero, the source code is listed after string replacements and blank-culling have been done. This flag must be set non-zero in order to see the effect of blank-culling on the print output. When LR is given the value 0, sequence numbers generated by PN=2 or PN=3 (q.v., below) for continuation cards will be taken from corresponding continuation cards in the original input deck; when it has the value 1, sequence numbers generated by PN=2 or PN=3 for continuation cards will be taken from the card on which the statement started.
The keyword LS  LS (default value 19999) is the "Label Starting value". Statement labels generated by IFTRAN for use in the FORTRAN output will be (within each routine) LS, LS+LI, LS+2*LI, ... etc. The user must be careful not to give the parameters LS and LI values such that the set of labels used by IFTRAN overlaps the set of labels used in the original code (if any).

The keyword OC  OC (default value 0) is the "Output-Comments" flag. This flag may be set non-zero to cause comments to be copied from the source code to the output FORTRAN code.

The keyword OF  OF (default value 1) is the "Output-FORTRAN" flag. This flag may be set to zero to suppress the generation of any FORTRAN statements in the FORTRAN output file. I believe there may be a bug in this; in any case, the feature is not particularly useful and is provided only for the sake of completeness. It could be used to extract all the comments from a piece of IFTRAN code.

The keyword OI  OI (default value 0) is the "Output-IFTRAN-commands" flag. This flag may be set non-zero to cause IFTRAN commands (like .USE, .OPTION, etc.) to be copied to the FORTRAN output file (with a C in column 1).

The keyword ON  ON (default value 0) is the "Output-sequence-Numbering" flag. This flag may be set non-zero to request that sequencing information be placed in columns 73-80 of the cards sent to the FORTRAN output file. The value 1 will cause IFTRAN to use the "nesting level" and a card number relative to the beginning of the routine; these will match similar numbers placed at the beginning of the print output lines. The value 2 causes the contents of columns 73-80 of the original input cards to be used. Continuation cards will have the same numbers as the first card of the statement.

The keyword OS  OS (default value 0) is the "Output-IFTRAN-Statements" flag. This flag may be set non-zero to cause IFTRAN statements (like IF, WHILE, BLOCK, etc.) to be copied to the FORTRAN output file (with a C in column one).

The keyword PC  PC (default value 1) is the "Print-Comments" flag. This flag may be set to zero to suppress comments in the print output.

The keyword PF  PF (default value 1) is the "Print-FORTRAN" flag. This flag may be set to zero to suppress FORTRAN statements in the print output. I believe there may be a bug in this; in any case, the feature is not particularly useful and is provided only for the sake of completeness. It could be used to extract all the comments from a piece of IFTRAN code.
The Keyword PI  PI (default value 1) is the "Print-IFTRAN-commands" flag. This flag may be set to zero to suppress IFTRAN commands (like .IF, .OPTION, etc.) in the print output.

The Keyword PN  PN (default value 0) is the "Print-sequence-Numbering" flag. This flag may be set non-zero to request that sequencing information be placed in columns 73-80 of the cards listed in the print file. The value 1 will cause IFTRAN to use the "nesting level" and a card number relative to the beginning of the routine; these will match similar numbers placed at the beginning of the print output lines. The value 2 causes the contents of columns 73-80 of the original input cards to be used. See also the discussion of the LR flag (above), the value of which affects the behavior of PN=2 and PN=3.

The Keyword PR  PR (default value 1) is the "Print-output" flag. Setting this flag to zero suppresses all print output except the abbreviated print that appears when an error is detected. See also the .LIST and .NOLIST commands.

The Keyword RS  RS (default value 0) is the "Replace Strings" flag. Setting this flag non-zero causes string replacement to commence, and setting it to zero causes string replacement to be suspended. This flag is set non-zero by any .REPLACE command and zero by any .NOREPLACE command.

The Keyword RT  RT (default value 0) is the "Replace Tokens" flag. Setting this flag non-zero speeds up string replacement by only looking for replacements starting at an alphabetic character or a dollar sign which follows a character which is neither alphabetic nor a dollar sign.

The Keyword TC  TC (default value "null") is the "Tab Character". If this character is set non-null, the character may be used anywhere in columns 1 through 6 of input cards to indicate that the following 66 characters are to be moved to columns 7 through 72. See also the .TAB command.

The Keyword UC  UC (default value 1) is the "Unknown-Commands" flag. When this flag is set non-zero, input cards with a period, a C, or an asterisk in column 1 and a non-blank character in column 2 which are not recognizable as IFTRAN commands are listed with an error message indicating this. Setting the flag to zero suppresses the error message. In any case, such cards are treated as comment cards.

The Keyword XC  XC (default value "null") is the "eXtended-Comment" character. When a non-null character is specified as the value of this parameter, that character may be used on any input card (except a comment card or a .XCOM card) to indicate that what follows it on that card is a comment,
to be ignored. The extended-comment character may also be set by the \texttt{.XCOM} command. The new IFTRAN will list extended comments on the print output as long as the LR flag (q.v., above) has the value 0. It will not transfer them to separate comment cards in either the print output or the FORTRAN output, as an older version of IFTRAN did.

Once a given character is defined to be the extended-comment character, it is treated as such on a subsequent \texttt{.OPTION} card; this can cause puzzling behavior. If the extended-comment character is already set to \texttt{!}, for example, the command \texttt{".OPTION XC=!”} is flagged as being in error. To IFTRAN, it looks like the illegal command \texttt{".OPTION XC= “}. If you don’t know whether the extended-comment character is set or not and you want to set it to a particular character, use the command \texttt{".OPTION XC"} to turn the feature off, followed by the command \texttt{".OPTION XC=desired-character"}. An alternative is to use the command \texttt{".XOM desired-character"}; since extended comments are not allowed on \texttt{.XOM} cards, this will work no matter what the current value of the XC parameter is.

Consider the following simple IFTRAN test deck:

\begin{verbatim}
PROGRAM TEST
.REPL /$1$/100/
    DIMENSION A($1$)
    DO (I=1,$1$)
        A(I)=RANF()
        IF (A(I).EQ.0.)
            PRINT 1001
            STOP
    END IF
    END DO
    PRINT 1002, A
    STOP
1001 FORMAT (" RANF MISBEHAVIOR")
1002 FORMAT (" .10F7.3")
END
\end{verbatim}

The IFTRAN print file produced for this deck is as follows (the first and last lines have been shortened by the removal of some internal blanks so that they will fit on single lines in this write-up):
Each page of the print file has a header line giving a page number and the first 72 characters of the first FORTRAN statement from the current routine. If other cards precede the first FORTRAN statement of a given routine, you will get 72 minus signs in the header line on the first page of the listing of that routine.

Each input line is listed with a sequence number and the current code nesting level preceding it. The code is indented to clarify the nesting of IFTRAN constructs. (In fact, FORTRAN-77 block-IF constructs will be indented, also.)

In the example, there were no errors. Error messages are given on two lines, the first consisting of the string "** ERROR **" and the second attempting to indicate the nature of the error. Error messages always apply to the following input statement.

The summary line indicates the number of cards read by IFTRAN (this includes cards from the primary and secondary input units and from the SAVE/USE scratch unit), the number of errors detected, and, if possible, the elapsed CPU time.
The IFTRAN preprocessor described in this document has been installed at NAR on the CRAY-1, the PDP 11/70, and the IBM 4341 computer systems. A highly portable version is available to requesters and has been installed on a wide variety of machines. The IFTRAN language is the same for all of these versions, so that IFTRAN programs which are correct on one machine may be moved to another by supplying the appropriate Job Control Language. (Note, however, that IFTRAN does no FORTRAN syntax checking but merely writes to the output file any card which it does not recognize as an IFTRAN command, an IFTRAN statement, or a comment card.)

IFTRAN exists on the CRAY-1 as an absolute binary module whose name is known to the system. To load and execute it requires only the card:

```
```

All parameters are optional. Each dk is the name of a CRAY-1 data set. The keywords, their meanings, and the defaults are as follows:

"I=d1" specifies the input data set "d1". IFTRAN reads its next file. The default is the system data set $IN. The data set is not rewound; it is left positioned following the first EOF, EOD or .FIN encountered.

"L=d2" specifies the list output data set "d2". The default is the system data set $OUT. "L=0" is allowed and suppresses the list output. The data set is neither rewound nor end-filed.

"F=d3" specifies the FORTRAN output data set "d3". The default is a data set called $FTN. "F=0" is allowed and suppresses the FORTRAN output. The data set is not initially re-wound; at termination, it is end-filed and rewound.

"SU=d4" specifies the SAVE/USE scratch data set "d4". The default is a data set called $SAVUSE. This data set is rewound upon encountering the first .SAVE command; its position at termination is input-dependent and, in general, difficult to predict.

"SUM=d5" specifies a data set "d5" to receive the message containing the summary statistics of the IFTRAN run and, if normal list output is suppressed, error messages. The default is the system data set.
$OOr. SUM=0 is allowed and suppresses the summary outputs.

NA, if used, prevents IFTRAN from executing "CALL ABORT" at termination if errors were detected; STOP is executed instead and system control passes to the next command in the JCL file. The keyword "NOABT", as described in an earlier version of this write-up, may also be used to obtain the same effect.

Any data set used by IFTRAN, whether user-specified or default, is dynamically opened when first referenced, if it has not previously been ASSIGNED, ACCESSed, or otherwise created.

Primary IFTRAN input is a collection of 80-column card images, terminated by an EOF, an EOD, or a .FIN command. Input may be switched temporarily to other files by using .INCLUDE commands in the primary input file. An EOF, an EOD, or a .FIN command in a secondary input data set (one specified by its alias on a .INCLUDE card) causes resumption of reading from the primary input data set.

IFTRAN terminates execution when an EOF, an EOD, or a .FIN command is found in the primary input data set. A STOP statement is executed if no errors were detected or if the NA parameter was used, a "CALL ABORT" otherwise.

The list output, FORTRAN output, and summary output data sets are written using the system routine WRITEC; blank fields in these data sets are compressed as described in the COS manual. This usually causes no problems; if, however, you copy one of these data sets to $OUT or dispose it to some other machine on the network, you may get a file with peculiar characters interspersed with the expected stuff. The way around this is to insert a card like:

\[
\text{ASSIGN,DN=dn,BFI=OFF.}
\]

before the IFTRAN card for each such data set. See the third example below.

One peculiarity of IFTRAN on the CRAY-1: Compiler-directive lines, which have the characters CDIR$ in columns 1 through 5, are treated as FORTRAN statements, rather than as comments. (CFT treats them in the same way.) If they were treated as comments, in order to get compiler directives passed to the compiler one would have to use ".OPTION UC=0,OC=1" to suspend the unrecognized-command error messages and to pass all comments on to the
FORTRAN output file, not just the desired ones.

Examples To process an IFTRAN deck from the input stream and compile the FORTRAN output if there are no errors:

```fortran
.
.
IFTRAN.
CFT,I=$FIN.
.
.
(eof)
... IFTRAN source ...
(eof)
.
.
```

To preprocess and compile some source code from a permanent data set, HOMER, with the IFTRAN listing suppressed:

```fortran
.
.
ACCESS, DN=HOMER.
IFTRAN, I=HOMER, L=0.
CFT, I=$FIN.
.
.
```

To preprocess a deck from the input stream, writing the FORTRAN to a data set called FORCODE and the listing to a data set called IFLIST, copying IFLIST to $OUT if IFTRAN aborts due to detected errors.
For a debugging run, to preprocess some source code from two user permanent data sets, MODEL and PROCESS, suppressing generation of the FORTRAN code and suppressing a possible job abort on the first IFTRAN execution:

```
... IFTRAN source ...
(eof)
```

**USING IFTRAN ON THE IBM 4341**

On the IBM 4341, the IFTRAN EXEC, which resides on NCAR-LIBS' 450 minidisk (normally the "J" disk), is invoked by the command IFTRAN. It sets up the appropriate FILEDEFs and calls the IFTRAN module, which reads a file having a user-specified file name and the file type IFTRAN and writes either or both of two files having the same name as the input file, one with the file type IFLIST, which contains the IFTRAN listing, and the other with the file type FORTRAN, which contains the FORTRAN output.

The IFTRAN module generates a non-zero return code when errors are detected and that return code is passed back by the IFTRAN EXEC for use by a user EXEC.

The form of the IFTRAN command is shown below. The characters [ and ] are used to enclose optional portions of the command:
IPTRAN [INPUT=] file-name [other parameters]

All parameters have the form "keyword=value", with no embedded blanks, and are separated from each other by at least one blank. The possible parameters are as follows:

[INPUT=] file-name
[TMODE=] mode-letter
[OUTPUT=] output-file-option
[FMODE=] fortran-file-mode
[IMODE=] listing-file-mode
[SMODE=] scratch-file-mode

If "keyword=" is omitted from any parameter, the assumed keyword will be that implied by the position of that parameter in the list. For example, in the command:

IPTRAN MYFILE A SMODE=D B

the values MYFILE, A, and B are considered to mean INPUT=MYFILE, TMODE=A, and FMODE=B, respectively, because of their positions.

For compatibility with an earlier IFTRAN EXEC, the value % is allowed for any parameter and has the same effect as omitting the parameter.

The parameters may be described as follows:

file-name is the name of the IPTRAN input file. This is the only required parameter. The type of the file to be processed must be IFTRAN, its record format must be F and its logical record length must be 80 characters.

mode-letter is the mode of the input file. If this parameter is omitted, the user's disks are searched for a file with the given name and the file type IFTRAN; the mode letter of the first disk containing such a file (if any) is used.

output-file-option is FORTRAN or IFLIST or BOTH, specifying which of the two IFTRAN output files are to be produced. If this parameter is omitted, both output files are produced.

fortran-file-mode is the mode letter of the FORTRAN output file, which has the same name as that of the
input file and the file type FORTRAN. If this parameter is omitted and the FORTRAN file is to be written, the mode of the input file is used. If you do not have write permission on the minidisk specified, an error message is typed and the IFTRAN EXEC terminates with a non-zero return code.

listing-file-mode is the mode letter of the IFTRAN print file, which has the same name as that of the input file and the file type IFLIST. If this parameter is omitted and the IFLIST file is to be written, the mode of the input file is used. If you do not have write permission on the minidisk specified, an error message is typed and the IFTRAN EXEC terminates with a non-zero return code.

scratch-file-mode is the mode letter of the IFTRAN scratch file, used to implement the SAVE/USE feature of IFTRAN. This file has the same name as that of the input file and the file type $SAVEUSE. If this parameter is omitted, the mode of the input file is used. If you do not have write permission on the minidisk specified, an error message is typed and the IFTRAN EXEC terminates with a non-zero return code.

The character & is used internally in various ways by the IBM 4341 version of IFTRAN and should not appear anywhere in an IFTRAN input file.

**USING IFTRAN ON THE PDP 11/70**

IFTRAN exists on the 11/70 as an absolute binary module. It operates as a filter, reading IFTRAN statements from the standard input unit and writing FORTRAN statements to the standard output unit. The IFTRAN print may be sent to a Unix file for later perusal. An optional flag (-l or -L) is provided to swap the roles of the two output files, so that the IFTRAN print goes to the standard output unit and the FORTRAN to a Unix file. Thus, the command

```
iftran < input-file
```

reads input-file and sends the FORTRAN output to the terminal. The command

```
iftran -l < input-file
```

reads input-file and sends the IFTRAN print to the terminal. The command

```
iftran < input-file > fortran-file
```

reads input from input-file and sends the FORTRAN output
to fortran-file. The command

    iftran -l < input-file > listing-file

reads input from input-file and sends the IFTRAN print to listing-file. Either of the commands

    iftran listing-file < input-file > fortran-file
    iftran -l fortran-file < input-file > listing-file

reads input from input-file, sends the FORTRAN output to fortran-file, and sends the IFTRAN print to listing-file.

If a Unix file whose name is given as an argument of the "iftran" command exists and is non-empty, the contents of the file will be overwritten as "iftran" executes, but the file will not be cleared initially. If the file originally contained more characters than were written to it by the "iftran" command, some of those characters will remain at the end of the file. This was considered preferable to clearing the file. It is all too easy to type

    iftran irreplaceable-file > there

when what you really meant to type was

    iftran < irreplaceable-file > there

It would be most unpleasant to have "irreplaceable-file" cleared by this mistake. (The author speaks from bitter personal experience!) As it stands, if you make this mistake, just use "control-D" or "shift-delete" to terminate the process and your irreplaceable file will be intact.

IFTRAN input must be in upper case (lower-case statements are passed on unchanged, however) and untabbed (except, possibly, for an initial tab character on a line - the default setting of the option "TC" is the tab character).

Error messages are sent to the "standard diagnostic unit" (as well as to the IFTRAN print file); they are preceded by a three-line header and followed by a one-line message indicating the number of cards processed and the number of errors found. By default, these lines come out on the user's terminal; the Unix command "fd2" may be used to re-direct them to a file. If either the FORTRAN output or the IFTRAN print is directed to the terminal along with the diagnostic output, the two may be intermingled in a confusing way.

June 1983
The IFTRAN command "INCLUDE" has a single parameter - the name of the Unix file to be included as a part of the input stream; no card length or rewind flag may be specified.

Fred Clare has written a Unix command "ift", creating a user interface for "iftran" which is easier to understand and more in keeping with the philosophy of "cc" and "INfort". Take a look at "/fred/bin/ift" (or just execute it with no arguments).

IFTRAN has been successfully implemented on a wide variety of machines. To implement IFTRAN elsewhere, you should first obtain two files from NCAR - the portable FORTRAN version of IFTRAN and the IFTRAN version of IFTRAN. The idea is to get the FORTRAN version running, however inefficiently, and then use it to process the IFTRAN version to get a better FORTRAN version.

The FORTRAN version is completely uncommented; the comments in the IFTRAN version will be helpful in implementing both versions.

The FORTRAN version conforms to the ANSI-77 standard, but is easily modifiable to an ANSI-66 form by replacing all occurrences of 'x' by 'lhx' (where x stands for any single character) and all occurrences of CHARACTER*1 by INTEGER. The string ',' may be somewhat tricky to change. If you do the replacements using an automated tool, be ready for problems with statements such as:

```
DATA ..., 'T', 'F', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '/
```

in which the replacement of ',' by 'lh', must not be done. If you do all of the other replacements first, this problem does not arise.

In the FORTRAN version, input is read from unit 1, print output goes to unit 2, card output goes to unit 3, unit 4 is used as a scratch unit, and unit 5 is the "summary unit", to which error messages are written when print output is suppressed and to which the line summarizing the elapsed time, the number of errors encountered, etc., is written at the end of the run.

You will probably need to modify the input-output routine IFTRIO and it would be to your advantage to modify the hash-coding routine IFHASH, even in the FORTRAN version, before using it to process the IFTRAN version.
Comments concerning problems encountered by implementors will be accepted and possibly included in future versions of this document as an aid to others.
# DIRECTIVES

## CONTENTS

1. **INTRODUCTION**
   - 1.1 General Definitions
     - 1.1.1 Operation
     - 1.1.2 Control
     - 1.1.3 Directive File
     - 1.1.4 Programmer Directive and User Directive Files
     - 1.1.5 Directive
     - 1.1.6 General, Variable, Function Controls
     - 1.1.7 Area, Area Vector
     - 1.1.8 Aggregate, Group, Element
     - 1.1.9 Numeric Storage Unit

2. **SYNTAX**
   - 2.1 Use of Blanks in Directives
   - 2.2 Datum Types
     - 2.2.1 String
     - 2.2.2 Name
     - 2.2.3 Integer
     - 2.2.4 Real
     - 2.2.5 Special
   - 2.3 Delimiters
     - 2.3.1 Logical Separator—Comma "",
     - 2.3.2 Group—Parentheses "(" and ")"
     - 2.3.3 Aggregate—Square brackets "[" and "]"
     - 2.3.4 Repetition Factor—Star "*"
     - 2.3.5 Assignment—Equals "=

---

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 General Definitions</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Operation</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Control</td>
<td>1</td>
</tr>
<tr>
<td>1.1.3 Directive File</td>
<td>2</td>
</tr>
<tr>
<td>1.1.4 Programmer Directive and User Directive Files</td>
<td>2</td>
</tr>
<tr>
<td>1.1.5 Directive</td>
<td>2</td>
</tr>
<tr>
<td>1.1.6 General, Variable, Function Controls</td>
<td>2</td>
</tr>
<tr>
<td>1.1.7 Area, Area Vector</td>
<td>3</td>
</tr>
<tr>
<td>1.1.8 Aggregate, Group, Element</td>
<td>3</td>
</tr>
<tr>
<td>1.1.9 Numeric Storage Unit</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Document Information</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Contents of This Document</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 Definition of Local Terms</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2.1 Token</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2.2 Lexical Scanner</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2.3 Datum</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2.4 Delimiter</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2.5 Keyword</td>
<td>6</td>
</tr>
<tr>
<td>1.2.2.6 Line</td>
<td>6</td>
</tr>
<tr>
<td>2. SYNTAX</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Use of Blanks in Directives</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Datum Types</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 String</td>
<td>7</td>
</tr>
<tr>
<td>2.2.2 Name</td>
<td>7</td>
</tr>
<tr>
<td>2.2.3 Integer</td>
<td>8</td>
</tr>
<tr>
<td>2.2.4 Real</td>
<td>8</td>
</tr>
<tr>
<td>2.2.5 Special</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Delimiters</td>
<td>9</td>
</tr>
</tbody>
</table>
| 2.3.1 Logical Separator—Comma "",
| 2.3.2 Group—Parentheses "(" and ")"
| 2.3.3 Aggregate—Square brackets "[" and "]"
| 2.3.4 Repetition Factor—Star "*"
<p>| 2.3.5 Assignment—Equals &quot;=&quot; | 11 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.2.3</td>
<td>Control Name Directives Without $FOR</td>
<td>30</td>
</tr>
<tr>
<td>4.1.2.4</td>
<td>Any LNK Directive</td>
<td>30</td>
</tr>
<tr>
<td>4.1.3.1</td>
<td>Any LET Directive With $FOR</td>
<td>30</td>
</tr>
<tr>
<td>4.1.3.2</td>
<td>Any Control Name Directive With $FOR</td>
<td>30</td>
</tr>
<tr>
<td>4.1.3.3</td>
<td>The VARVEC, VARNEW, and VARDRP Directives</td>
<td>30</td>
</tr>
<tr>
<td>4.2.1.1</td>
<td>The Control Name Directive NAMPD</td>
<td>31</td>
</tr>
<tr>
<td>4.2.1.2</td>
<td>ENDOP</td>
<td>31</td>
</tr>
<tr>
<td>4.2.2.1</td>
<td>Any LET Directive, With or Without $FOR</td>
<td>31</td>
</tr>
<tr>
<td>4.2.2.2</td>
<td>Any Control Name Directive, With or Without $FOR</td>
<td>31</td>
</tr>
<tr>
<td>4.2.2.3</td>
<td>The VECVAR, NEWVAR, DRPVAR, and NEWFUN Directives</td>
<td>32</td>
</tr>
<tr>
<td>4.2.3.1</td>
<td>Any DEF Directive</td>
<td>32</td>
</tr>
<tr>
<td>4.2.3.2</td>
<td>Any DIM Directive</td>
<td>32</td>
</tr>
<tr>
<td>4.2.3.3</td>
<td>Any LNK Directive</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>Problem Statement</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td>Tabular Statement of Problem</td>
<td>34</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Naming of Controls</td>
<td>35</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Order of Controls</td>
<td>36</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Typing of Controls</td>
<td>38</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Number of Elements in a Group</td>
<td>39</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Linking</td>
<td>40</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Initial Values</td>
<td>41</td>
</tr>
<tr>
<td>5.3</td>
<td>Building the Programmer Directive File</td>
<td>43</td>
</tr>
<tr>
<td>5.3.1</td>
<td>The DEF Directives</td>
<td>43</td>
</tr>
<tr>
<td>5.3.2</td>
<td>The DIM Directives</td>
<td>43</td>
</tr>
<tr>
<td>5.3.3</td>
<td>The LET Directives</td>
<td>43</td>
</tr>
<tr>
<td>5.3.4</td>
<td>The LNK Directive</td>
<td>43</td>
</tr>
<tr>
<td>5.3.5</td>
<td>The ORD Directive</td>
<td>44</td>
</tr>
<tr>
<td>5.3.6</td>
<td>The ENDOP Directive</td>
<td>44</td>
</tr>
</tbody>
</table>
5.4 Summary of Programmer Directive File................. 45

5. SAMPLE USER DIRECTIVE FILE............................... 47

6.1 Problem Statement........................................... 47

6.2 Tabular Statement of Problem.............................. 48

6.3 Building the User Directive File.......................... 49
  6.3.1 The NAMPD Control Name Directive.................. 50
  6.3.2 Listing Area Vectors................................. 50
  6.3.3 Setting Values....................................... 50
  6.3.4 The ENDOP Directive................................ 51

6.4 Summary of User Directive File............................ 52
1. INTRODUCTION

This document describes the syntax and semantics of directives, and the structure of these directives in a directive file (both of these are defined in detail below). It is intended to provide the complete definition of directives and directive files.

The remainder of this introduction is divided into two parts; the first defines general or global terms (i.e. those listed in the Terminology Glossary) which are key to the current discussion, while the second defines terminology and abbreviations used only in this section. If you are familiar with the concepts of the general terms, you may skip the first part without any loss of understanding. However, the second part should be read to understand terms and the format of this section.

1.1 General Definitions

1.1.1 Operation

The GENPRO program consists of numerous functions which operate on data in some fashion. Although there are literally thousands of different kinds of functions that could be used to operate on the data, these functions may be categorized and provided with a generic name referring to the types of processing this category performs. For example, a collection of filters, and interpolation, might be categorized together and called FILTRP, for FILter and inTerPolation, or a collection of functions which produce various kinds of graphics output might be called PLOTTER. The set of generic categories are called operations.

These operations are more than a conceptual gathering of functions; in general, they also represent a single module, or code designed to work independently of other operations, to perform all of these functions.

1.1.2 Control

Operations, by definition, are a collection of related functions. It is often the case that an individual user wants to use all of these functions on his data. Instead, he normally would want to choose, as it were, from a menu, the functions appropriate to the nature of his data. In addition, a given function may have various options on the way it processes the data, which must also be specified. In other words, the user must have control over the functioning of an operation, and control he has is specified through a set of options, called controls.
1.1.3 Directive File

At this point, the user may choose the options he desires through the use of controls, but setting these controls is a long and laborious process since there are so many, and they must be set up in a very exacting manner. What is needed is a means for a user to input these controls in a form that resembles a human language and allows for numerous abbreviations, to greatly simplify the task of telling each operation what to do. This input form is called a directive file.

1.1.4 Programmer Directive and User Directive Files

The controls of each operation are defined by two directive files; the Programmer Directive file (PD) is used by the creator of the operation to define which controls are necessary, how they are dimensioned, the type of values to be input into each control, etc., while the User Directive file (UD) is used by the person performing processing on an actual data set to define a set of values to be ascribed to each of the controls.

1.1.5 Directive

The combination of the PD and UD form a complete definition of all of the controls for an operation. Just as a puzzle is a whole, but actually consists of pieces which go together to complete the puzzle, each directive file actually consists of numerous parts which together form a directive file. These parts, called directives, are actually a set of commands, each of which helps define the options desired for a particular operation. Each directive resembles a simple subset of mathematical notation (human oriented language), which performs a specific task or function. These directives allow complete control over an operation, and are much easier to learn and use than changing the controls manually. Therefore, our attention for the rest of this document will be toward how these directives are formed, and how they work.

1.1.6 General, Variable, Function Controls

Back up a little, controls of an operation may actually be divided into categories which are helpful for us to consider. For example, there are some controls which instruct the entire operation; things such as the type of data, the location of this data, the amount of data, etc. are necessary regardless of the individual function to be applied. These are accordingly called General controls. Other controls refer to individual items of the list of dependent variables or parameters, such as the rate of that variable, where it is located, its title, etc. These controls are appropriately called Variable controls. Finally, still other controls refer to the nature of a function; the type of plot, what to label the axes, how many points to plot, etc., as an example for a PLOTTER operation. These
controls are called **Function controls**.

### 1.1.7 Area, Area Vector

Controls, therefore, fall into one of three categories, hereafter called Areas; the General Area, the Variable Area, and the Function Area. Each Area contains a list of items for which the controls of that Area apply; for example the Variable Area contains the list of dependent variables. The list for each Area is called the Area Vector. (In actuality, the General Area does not have a formal Area Vector, but, for the sake of consistency of discussion here, it is assumed that its Area Vector simply contains only one item.)

### 1.1.8 Aggregate, Group, Element

The concept of a control is user oriented in that it may be viewed as one option to be exercised on an operation. However, the concept must be expanded to describe the full flexibility of controls which are required to define options. It is possible that a control may consist of many related values. For example a control may contain a set of coefficients used in some calculation. Therefore, the first concept to be expanded is that a control may contain numerous related pieces of information. In addition, the number of coefficients given in the control may vary from job to job. Thus additionally, the actual number of values for a control is determined at run time, and therefore, may not be preset.

These concepts deal with the structure of a control, and therefore, different terminology will be used when referring specifically to the structure of a control. An aggregate represents a control in the special context of structure. It may consist of one or more groups. Each group is a unit of information. As an example, suppose that a set of times was needed to specify at what time to begin another process. Then each time would represent one group, the collection of all times, no matter how many there were, would represent the aggregate. Furthermore, a group may be expressed in terms of one or more elements. In the example above, assume further that each time was in terms of hours, minutes, and seconds. Then the group would be comprised of three elements; the first containing the hours, the second the minutes, and the third the seconds of the time specified. These terms form the structural aspect of a control.

### 1.1.9 Numeric Storage Unit

One additional term, which is tied heavily to computers, is the concept of a numeric storage unit (NSU). It is the unit in which an integer or real value can be stored on that particular machine. The length of an NSU, counted in the number of bits, varies considerably from machine to machine, and thus the...
general concept is introduced. In this discussion, it is only important to realize that even though an element is the lowest structural component of a control conceptually, it may actually need several NSU's to be expressed in a particular machine. This concept is not key to understanding how directives files are built, or work, and so a rough knowledge of its existence is enough here.

1.2 Document Information

1.2.1 Contents of This Document

This document describes directives and directive files in their entirety. To help break this task into smaller and more readily assimilated subtasks, each section of this document discusses one aspect of directives. The first is a discussion of the syntax of directives, or the structural components that together build legal directives. The next section covers semantics of directives, or the process of ascribing meaning to the assemblage of the structural components forming legal directives. The next section describes the structure of directive files, or the set of directives that form a complete file.

A simple parallel might prove helpful. In the English language, syntax is equivalent to grammar; it is not the concern of grammar to ensure sentences that make sense, but only to ensure that all legal components of a sentence appear, and appear in the correct order. Semantics is the concept of adding meaning or a sense to grammatically correct sentences. Finally structure would involve a design or plan for concatenating sentences together to form a coherent idea, (like this document.).

The final two sections provide examples of a Programmer Directive and a User Directive file, in that order. These files are for purposes of example only, and should not be construed as exhaustive examples of building a directive file, nor to represent any actual data. These two files are related to each other, however, in that they are the Programmer and User Directive files for a single hypothetical operation. In addition, an example of the Control Array generated by these two files may be found in the Control Array section of the DATA STRUCTURES document.

1.2.2 Definition of Local Terms

In discussing the syntax of directives, there are some terms that are helpful in describing the structural components.

1.2.2.1 Token

First, the term token will be used as a more exact term for "structural component" used earlier. It is
the smallest logical unit appearing in a directive. (The grammatical concept of "morpheme" is equivalent to the term "token" used here.)

1.2.2.2 Lexical Scanner

The human mind, when confronted with a sentence, is interpreting the sense, as well as the grammatical structure, virtually simultaneously with the input flow of information from the eyes. Computers, however, are terribly "nearsighted", have poor "memories", and a low "attention span". It is therefore critical to break directives down into their smallest logical units (tokens), so that the computer may analyze each token with respect to every other token in the directive. The computer process which breaks a directive down into tokens is called the lexical scanner.

1.2.2.3 Datum

In human language, the smallest logical unit is a word. Each word has associated with it a part of speech, or class of function performed in a sentence, such as noun, verb, adjective, etc. Similarly, a class of tokens, called datums, represent "words", or a token to which meaning can be ascribed. Each datum has associated with it a "part of speech", here called datum type, such as name, string, integer, special and real (These are discussed in detail in the Syntax section below). Each datum is part of a finite set called a vocabulary, similar to the vocabulary of any language, with a unique meaning ascribed to it.

Note that the plural of "datum" is either "data" or "datums", according to Webster. Since "data" carries an entirely different concept in computer terminology, it was abandoned in favor of the less used--but more specific--plural form, "datums".

1.2.2.4 Delimiter

Again, human language lends itself to a parallel for the next concept. Words form the most important parts of a sentence, for those are the ideas which will be stored away in our memories. However, the other portion of a sentence, "punctuation", is critical to clearly structure the words to take on the correct meaning. They form the mortar with which to assemble words to form a coherent sentence. Similarly, the second class of tokens is the set of delimiters, which act as the mortar to build datums into a legal directive. The set of delimiters is finite, each with a
particular meaning. This varies slightly from punctuation in that the meaning of a punctuation mark may vary widely dependent upon the context, whereas delimiters are ascribed unique meanings. The section on Syntax below discusses the full set of these delimiters, and their meaning and usage.

1.2.2.5 Keyword

The human mind has an ability to sort, organize, and store information. A trained mind can store large amounts of information and provide for itself a "handle" whereby that information can all be remembered simply by remembering the "handle". We find it much easier to remember what we have read when a title has been associated with a section of related information. Similarly, the lexical scanner is provided with its own form of "handles", called keywords, under which a set of information may be stored. As in documentation, where a title is just a word (or collection of words), the keyword is simply a special datum. Within a directive, it is identified in a specific manner, which is discussed in the Semantics section below.

1.2.2.6 Line

It has been tacitly assumed in the discussions above that a directive has many parallels to the concept of a sentence in English. This analogy is indeed a good one and provides a mnemonic to clear understanding of directives.

One last concept deals with a slightly larger parallel than the sentence. On a typewritten page, a sentence may physically occupy one or more lines. It would be ridiculous to require only one sentence on a line, for some sentences are so long that they would not fit on a line, while other sentences are so short that they would waste space.

This parallel carries over to directives very closely. Some directives are very long and require more than one line, while others may be short enough to allow for two or three on a line. Just as the length of a line on a typewritten page is fixed (determined by the width of the page), the length of a line for directives is fixed. Actually, the line width for directives is dependent on the medium on which the directive files are input. As an example, for computer cards, a line is defined to be one card, and its length to be the number of columns on that card, or 80.
2. SYNTAX

2.1 Use of Blanks in Directives

In general, blanks in a directive are not significant. Therefore, blanks may be used freely to enhance readability. The only exception is when blanks are included within a "string" datum type. (See 2.2.1 "String")

2.2 Datum Types

In the general case, it is necessary to input a variety of information to provide an operation with the required details to properly execute. On the computer this would include integer, real, and character information. All of the legal datum types are discussed in this section.

In the parallel case of English, the parts of speech for a given word may be determined by the context in which it appears, by its explicit definition, or by usage of prefixes or suffixes (the "er" suffix usually implies an adjective, the suffix "ology" usually implies a noun). In the case of datums, the type determination is most similar to the prefix/suffix case; it is determined completely from inspection of the datum, not through contextual or predefinable means.

2.2.1 String

Character strings are determined with the use of single or double quotes completely bracketing the string. The string may be of any length up to the length of a line, and may contain any set of alpha, numeric, or special characters, except the string delimiters. The string must be bracketed on both sides by the same delimiter. If the character being used as the string delimiter is also included as part of the string, it must be immediately preceded by one occurrence of that delimiter, or in other words, must appear twice.

EXAMPLE: Legal strings
The following datums,
"ISN'T THIS A NICE, LEGAL STRING?"
'ISN'T THIS A NICE, LEGAL STRING?'
would both be interpreted as,
ISN'T THIS A NICE, LEGAL STRING?

EXAMPLE: Illegal strings
"TERMINATED BY DIFFERENT DELIMITER"

2.2.2 Name

A name is a datum whose first character is alphabetic (A through Z), and all following characters are alphanumeric (alphabetic or 0 through 9). Names are similar to strings in that they are character representations, but dissimilar in that their length is fixed to the number of characters that can fit within a line.
into one numeric storage unit, determined by the computer on which this job is running.

EXAMPLE: Legal names
SUPERCALIFRAGILISTICEXPIALIDOCIOUS
and
SUP ERC ALI FRAG ILIS TICE XPI A LID OCIO US
are equivalent and the following table illustrates how this name would be represented on various machines.

<table>
<thead>
<tr>
<th>Machine</th>
<th>#chars/NSU</th>
<th>Internal representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>SUPE</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>SUPERCAL</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>SUPERCALIF</td>
</tr>
</tbody>
</table>

*The number of characters per numeric storage unit

EXAMPLE: Illegal names
1NAME (first character not alphabetic)
NAME? ("?" character not alphanumeric)

NOTE: For purposes of portability, all datums of datum type "name" will be unique in the first four characters.

2.2.3 Integer

Integer values are represented just as they would be in FORTRAN. The first character determines that it is numeric; a scan of the rest of the characters differentiates between integer and real values. An integer may be signed or unsigned; the plus (+) and minus (-) unary operators represent positive and negative values respectively, while an unsigned integer is always non-negative.

EXAMPLE: Legal integers
1
237
-0
-4 4 (legal, due to rule on use of blanks)

EXAMPLE: Illegal integers
+-4
4+
44G

2.2.4 Real

Real numbers are represented in the same manner as floating point numbers in FORTRAN. As in the case of integers, the first character must be numeric, or a sign. The difference between integer and real is the presence of a decimal point or "E" somewhere in the real number. A real number may be signed
or unsigned; the plus (+) and minus (-) unary operators represent positive and negative values respectively, while an unsigned real is always non-negative.

**EXAMPLE: Legal reals**

1.0  
36.  
-98.6  
-12E-4  
6.47E22

**EXAMPLE: Illegal reals**

1..0  
-1+1.0  
1.0.

### 2.2.5 Special

In addition to the types above which resemble FORTRAN types, there is a special value that may be designated in a directive. Currently there are four special cases; the meaning for each is reserved for the Semantics section below. However, the syntax for all four cases is the same: the first character is the percent sign (%), followed by a legal "name". The four cases are; %L, %T, %S, and %FOR.

**Delimiters**

This section explains the meaning and usage of delimiters that may be used in directives. Examples are given for each delimiter. These examples use correct syntax only--descriptions of correct semantics is reserved for later sections.

#### 2.3.1 Logical Separator--Comma ",,"

In English grammar, the comma is used to separate a sentence into logical ideas for readability. Similarly, commas are used in directives to separate datums from one another. Additionally, the comma may be used to separate logical constructions of datums (i.e. groups and aggregates).

**EXAMPLE: Separation of elements**

4.0, 37, CAVEAT, -444.33, "EMPTOR"  
(Since the discussion on the syntax of groups and aggregates appears below, further examples of comma are reserved until each new construct is introduced.)

#### 2.3.2 Group--Parentheses "(" and ")"

In a directive, group structure is represented with an open parentheses, followed by one or more datum (separated by commas) representing elements that comprise that group, followed
finally by a close parentheses. All datum within the parentheses must be of the same type, with the exception of the special type which may be mixed freely. The length of a group is fixed, and datum which exceed that length are ignored. If there is only one element in a group, parentheses around that group are optional.

**EXAMPLE:** Logical separator
Groups, and datum within a group, are separated by commas.
( 55., .07, 1.00 ), ( "MY KINGDOM FOR A HORSE", "NAY" )

**EXAMPLE:** Mixed type error
( 44., 3., RASPUTIN )

**EXAMPLE:** Defaulting
Assume that a group were defined to contain three elements. Assume further that only the third element is to be assigned a new value. Then
( , , "ZENOBIA" )
would cause the value of the first two elements to be unchanged, but the third would be given the value "ZENOBIA".

**EXAMPLE:** Overdimension
Assume that a group were defined to contain three elements. Then in,
( TOULOUSE, VINCENT, LEONARDO, SALVADOR )
SALVADOR would be ignored.

**EXAMPLE:** Underdimension
Assume that a group were defined to contain four elements. Then in,
( "SANCHEZ", "TONTO", "SAMWISE" )
the first three elements receive new values, while the fourth element of the group is left unchanged.

2.3.3 Aggregate--Square brackets "[" and "]"

Aggregate structure may be represented within a directive with the use of square brackets. Its syntax is an open square bracket followed by one or more groups, which are separated by commas, followed by a close square bracket. All rules regarding groups apply here. There are no restrictions to the number of groups in an aggregate, i.e. that number is variable and is set at run time. All groups in the aggregate have the same length and must have the same type. If an aggregate contains only one group, square brackets are optional.

**EXAMPLE:** Logical separator
Aggregates, and groups within an
aggregate, are separated by commas.

\[(A,B,C) , (D,E,F) , (G,H,I)\] , \[(1,2) , (3,4)\]

**EXAMPLE: Mixed type error**

\[(A,B,C) , (1,2,3) , (G,H,I)\] 

### 2.3.4 Repetition Factor--Star "*"

A series of datums which all contain the same value, may be abbreviated conveniently with the star delimiter, which causes a value to be repeated for a specified number of sequential datums. Its general form is a datum containing an unsigned integer which is the repeat factor, followed by a star delimiter, followed by a datum which contains the value to be repeated.

**EXAMPLE:**

\[(\text{EMPTY},\text{EMPTY},\text{EMPTY}) , 0.0 , 0.0\]

could be represented as

\[(3^*\text{EMPTY}) , 2^*0.0\]

### 2.3.5 Assignment--Equals "=

The introduction defined keyword as being a handle for a collection of tokens. The means for setting up keywords is via the assignment delimiter. A keyword, appearing as the first datum of a directive, is followed by the assignment delimiter (the equals sign ":="), which is followed by a collection of datums and appropriate delimiters (i.e., only the comma, parentheses, square bracket, and star delimiters). This collection of tokens is then assigned to the keyword defined.

Remember, keywords are also datums, and their type must always be "name". However, the types of datums in the collection being assigned to the keyword do not need to match in type (that is, unless they violate a higher priority rule, such as elements within a group). Only one keyword may appear in a directive, and only one assignment delimiter may appear as well.

**EXAMPLE: Legal assignments**

```
MARES = EAT, OATS, AND, DOES, EAT, OATS
TEMPERATURES = 98.6, -273.16, "ZERO", 210
X = Y
X = "Y+2"
```

**EXAMPLE: Illegal assignments**

```
"MARES" = EAT, OATS, AND, DOES, EAT, OATS
(keyword is of illegal type),
TEMPERATURES = (98.6, -273.16, "ZERO", 210)
(rules involving elements in a group violated),
X = Y + 2
(rules regarding separation of datums violated).
```
2.3.5.1 Continuation lines

An assignment may not all fit onto one line, and therefore the capability for continuation of a directive to the next line has been made available.

Again, in English, a sentence has not been completed until a period appears at the end of it. Therefore, if, while reading, you come to the end of the line and do not find a period (terminator) for the current sentence, you assume that the sentence is continued on the next line, and you read on.

In the case of directives, no explicit terminator is provided. In this case, then, it is assumed that a directive fits onto a line, unless specifically told otherwise. This is accomplished by indicating that the directive is still "open" by using a comma as the last character on that line (not counting blanks, of course).

EXAMPLE: Legal continuation lines

STORY = ONCE, UPON, A, TIME, THERE, WERE, THREE, BEARS, MAMA, BEAR, PAPA, BEAR, AND, BABY, BEAR

2.3.6 Multiple Assignments per Line--Dollar "$"

As seen in the discussion for continuation lines, usually the end of a line is also the terminator that represents the end of an assignment directive. There are two notable exceptions; continuation lines, which are intentionally left "open" with the use of a comma at the end, and multiple assignments per line, which basically will force the "closing" of an assignment before the end of line is encountered. The concept behind multiple assignments is that if there were some means of terminating an assignment other than the end of the line, then it would be possible to include more than one assignment on one line. The dollar delimiter is used to do just this; an assignment is terminated with the use of this delimiter, so that other assignments may be inserted on the same line of information. Since the dollar delimiter is used as an end-of-assignment mark, it can appear only after a complete, legal assignment.

EXAMPLE: Multiple assignments

STRING = "LEAGUE"
REAL = 1.0
INTEGER = 14
NAME = GENGHIS

could be represented as

STRING = "LEAGUE" $ REAL = 1.0 $ INTEGER = 14 $ NAME = GENGHIS
2.3.7 Comment--Slash "/"

All lines for a directive are usually fixed in length. However, the slash delimiter allows the length of the line in which it occurs to be shortened. Essentially the length of a line on which the slash delimiter occurs is the number of characters up to, but not including that delimiter. The net effect is to stop the lexical scanner from reading any more characters in an attempt to find any more tokens on that line.

The implication of the above discussion is that any characters following the slash delimiter are ignored. This provides a feature of adding comments (ignored information) into a directive file. It should be remembered, that all rules regarding line continuation and multiple assignments still apply, only they apply to shortened lines in this case.

EXAMPLE: Legal comment usage
TEMP = 98.6 /MEAN BODY TEMP OF ADULT HOMO-SAPIEN
/THIS ENTIRE LINE IS USED FOR COMMENTS, I.E. IGNORED.
/REMEMBER, EVERYTHING AFTER THE SLASH (/) IS
/COMPLETELY DISREGARDED, SO X=1,[(2,4),(8,1)],2
/IS NOT AN ASSIGNMENT DIRECTIVE AS ONE MIGHT
/THINK. AN EXAMPLE OF AN ASSIGNMENT
/CONTINUATION FOLLOWS;
TERMS = FRIGID, FROSTED, /COLD COMMENTS
SIZZLING, STEAMING, INFLAMED /HOT COMMENTS

EXAMPLE: Illegal comment usage
TEMP = /ILLEGAL BECAUSE CONTINUATION RULE NOT MET
3. **SEMANTICS**

The basic tools for interpreting input information from directive files have been discussed. It is the intent of this section to assemble the syntactic tools previously described, and apply them to the specific case of the GENPRO program. A vocabulary of keywords will be developed (with room for expansion), and definitions and rules prescribed for the nature, content, type, length, and structure of the collection of tokens associated with each keyword. It **IS** the intent of this section to therefore describe the semantics of individual directives; it **IS NOT** to describe how these directives are brought together to form a coherent directive file: that will be discussed in the following section.

3.1 **Defining Area Controls--The DEF Directive**

Before any information regarding a control may be associated and assigned to that control, the existence of that control must first be specified. The DEF directive is used to define the entire list of legal controls for a given Area.

The structure of this directive is;

**Keyword**  
Composed of the three characters 'DEF' followed by a three character representation of the Area for which these controls are being defined.

**Assignment Delimiter**  
The equals sign (=).

**List of Datums**  
The list of controls to be defined. Each datum of this list must be of type "name". The order of these controls in this directive are significant (see STRUCTURE of this document, and CONTROL ARRAY section 1.3.3 of the DATA STRUCTURES document).

**Rules**

* Each occurrence of the DEF directive must contain a unique subset of the total set of controls of the Area being defined, i.e. the intersection of controls of any two DEF directives of the same Area must be the null set.

* The DEF directive is required if any controls are to be specified for an Area.

* All datums of a DEF directive are of type "name".

* All controls are global in nature, and therefore have only one unique meaning. This implies that the same control may not be used in another Area of this operation, or in any Area of any other operation, unless it has the same definition. A list of
the controls currently defined are in the GLOSSARY document.

* Aggregates and groups are not allowed.

** EXAMPLE: Legal DEF directives**

DEFGEN = GC1, GC2, GC3, GC4, GC5
DEFVAR = VC1, VC2, VC3, VC4
DEFFUN = FC1, FC2, FC3, FC4, FC5, FC6

** EXAMPLE: Illegal DEF directives**

DEFGEN = "GC1", "GC2", "GC3"
(datum are of wrong type),
DEFVAR = 1.0, 3., .333
(datum are of wrong type),
DEFFUN = ( FC1, FC2, FC3 )
(group definition not allowed).

3.2 Dimensioning, Initializing and Typing Area Controls—The DIM Directive

Each control for an operation is represented by an aggregate (even if it consists of only one element in one group). An aggregate consists of a number of one or more groups, and that is not completely specified until the entire directive file has been scanned. However, the length of a group is fixed and must be initially defined. In addition, the type of the aggregate must be determined, and an initial value inserted into all elements of the aggregate. All these tasks are performed by the DIM directive.

The structure of this directive is:

**Keyword**

Composed of the three characters 'DIM' followed by a three character representation of the Area for which these controls are being dimensioned, typed, and initialized.

**Assignment Delimiter**

The equals sign (=).

**List of Datums**

A set of aggregates which contain initial values to be assigned to defined controls.

**Rules**

* The order of the aggregates defined in this directive must be in the same order as the controls they initialize, i.e. in the same order as the controls listed in the DEF directive.

* Since the number of elements in a group is dimensioned by this directive, all group lengths specified in this directive must be correct.

* Any type of datum, except "special" may be used to initialize elements.
The type of the entire aggregate is set by this directive. Therefore, all datum in following directives must agree in type with the initial value given here.

If more than one group is to be initialized for a given control, the aggregate delimiter (square brackets) is required to delimit this aggregate from the next.

**EXAMPLE:** Legal usage

Assume the following;

```
DEFGEN = GC1, GC2, GC3, GC4
```

Then the following is legal;

```
DIMGEN - (ONE,TWO,THREE), 'THIS IS A STRING',
[(2,3),(4,6)], [(0.), (0.)]
```

Convince yourself that the following is true;

<table>
<thead>
<tr>
<th>CONTROL NAME</th>
<th>TYPE</th>
<th>ELEMENTS/ GROUP</th>
<th>VALUE(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC1</td>
<td>name</td>
<td>3</td>
<td>(ONE,TWO,THREE)</td>
</tr>
<tr>
<td>GC2</td>
<td>string</td>
<td>1</td>
<td>'THIS IS A STRING'</td>
</tr>
<tr>
<td>GC3</td>
<td>integer</td>
<td>2</td>
<td>(3,3),(4,6)</td>
</tr>
<tr>
<td>GC4</td>
<td>real</td>
<td>1</td>
<td>(0.), (0.)</td>
</tr>
</tbody>
</table>

### 3.3 Setting Values for Area Controls

In addition to the initialization of values for all controls in an Area, there exist two directives which allow resetting of any or all values at any time in the directive file. Examples of the necessity of such resetting are; adding new groups to the aggregate, inserting values of datum type "special" into aggregates, or simply new case not provided by the initialization.

This section discusses both directives mentioned above, then move on to other topics related to the setting of values in aggregates.

#### 3.3.1 Setting All Controls—The LET Directive

This directive provides the means to reset values for all controls in an Area with only one command. The structure of this directive is;

Keyword Composed of the three characters 'LET' followed by a three character representation of the Area for which values of controls are to be reset.
Assignment Delimiter

The equals sign (=).

List of Datums

A set of aggregates which contain values which will replace the values used to initialize all controls.

Rules

* The order of aggregates in this directive must be the same as the order of controls for which these new values apply. The order of controls here refers to the order established by the last ORD directive (see below), or if there is none, by the order of the DEF directive(s) of that Area.

* Every datum in an aggregate must agree with the type preset by the DIM directive.

* The number of datums in a group for a given aggregate should agree with the dimension of a group preset by the DIM directive. If the number of datums is less than the defined number of elements in a group, then elements for which there is no specified datum will retain their original value. If the number of datums is more than the defined number of elements in a group, then the extra datums are ignored, which may not produce the desired result.

* Every item of the Area Vector receives exactly the same values for all controls.

EXAMPLE: Legal LET directives

Assume the following;

DEFGEN = GC1, GC2, GC3, GC4
DIMGEN = (ONE, TWO, THREE), "THIS IS A STRING", 
[(2,3),(4,6)], [(0.), (0.)]

Then the following is legal;

LETGEN = (ONE, TWO, THREE), ["STRING1", "STRING2"],
[(3,3),(4,6)], [3., 0.]

Convince yourself that the following is true;

<table>
<thead>
<tr>
<th>CONTROL NAME</th>
<th>TYPE</th>
<th>ELEMENTS/GROUP</th>
<th>VALUE(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC1</td>
<td>name</td>
<td>3</td>
<td>(ONE, TWO, THREE)</td>
</tr>
<tr>
<td>GC2</td>
<td>string</td>
<td>1</td>
<td>&quot;STRING1&quot;, &quot;STRING2&quot;</td>
</tr>
<tr>
<td>GC3</td>
<td>integer</td>
<td>2</td>
<td>(3,3),(4,6)</td>
</tr>
<tr>
<td>GC4</td>
<td>real</td>
<td>1</td>
<td>(3.), (0.)</td>
</tr>
</tbody>
</table>
3.3.2 Setting Single Controls--The Control Name Directive

It may be the case that only values for one of the controls specified in a long list of Area control need to be reset. It would be a long, laborious chore, wrought with potential errors, to reset the values for all controls, using the LET directive, just to reset the control that really needed new values. It would be indeed convenient if the control could be named as a keyword, followed by the values to be assigned to that control in this case. And this is exactly what the Control Name directive does.

Structurally, the control name, which has already been defined via the DEF directive, has a datum type "name". Since all keywords must be of datum type "name", it is actually a very simple extension to allow this directive.

Rules

* The keyword (control) must be defined (i.e. in a DEF directive for that Area).

* All datums specified must agree in type with the defined type (determined from the DIM directive).

* Only one control is being defined in this directive. Thus, values of only one aggregate are being reset. Therefore, the aggregate notation (square brackets) is not required to separate aggregates.

* Every item of the Area Vector receives exactly the same values for this control.

EXAMPLE: Legal Control Name directives
(This example is equivalent to the previous one.)
Assume the following:
DEFGEN = GC1, GC2, GC3, GC4
DIMGEN = (ONE, TWO, THREE), "THIS IS A STRING", [(2*3),(4,6)], [(0.), (0.)]

Then the following is legal;
GC2 = "STRING1","STRING2"
GC4 = 3.,0.

3.3.3 Value Abbreviations

In some of the examples above, the representation of groups that are to be left unchanged was awkward, since a slot for each element of the group had to be reserved with the use of the comma. In addition, if an entire group is to receive the same value, it also means extra work. This section describes legal abbreviations built into the code to interpret directives.
If all elements of a group have the same value, that value may be used without parentheses to define the entire group. The same applies for all groups in an aggregate.

EXAMPLE: Aggregate defaulting

Assume the following;

```
DEFVAR = VC1, VC2, VC3
DIMVAR = 1, [(1,2,3),(4,5,6),(7,8,9)], 4
```

then all the following LET directives are equivalent;

```
LETVAR = [,,(,),,,], 7
LETVAR = [,,(,),,,], 7
LETVAR = [,,], 7
LETVAR = [,], 7
LETVAR = =, 7
```

EXAMPLE: Re-setting values

Assume the following;

```
DEFVAR = VC1, VC2, VC3
DIMVAR = 1, [(1,2,3),(4,5,6),(7,8,9)], 4
```

assume also that the third group of the aggregate for the second control was to be reset. Then, the following are equivalent;

```
LETVAR = [,,(1,1,1)], 7
LETVAR = [,,(3*1)], 7
LETVAR = [,,1], 7
```

Note that (1,) defines the first element of the group, while 1 defines all three elements.

Value abbreviations may be used only for LET and Control Name directives.

3.3.4 Special Value Definitions

The four "special" datum types may be divided into two logical categories which are discussed in this section; %FOR is used for Area Vector definitions, and %L, %T, and %S are used for Value Sharing.

3.3.4.1 Area Vector Definitions--%FOR

It was stated, but not emphasized, in the discussion of the LET and Control Name directives, that the entire Area Vector is set to the values defined for an aggregate or aggregates. It would be convenient to be able to define a set of values, either for one control or a set of controls, for only a select subset of items of the Area Vector; this is made possible with
the use of the special value, %FOR. The general structure appears as follows;

1. Any legal LET or Control Name directive.

2. A comma delimiter to indicate that the directive is still "open".

3. The special value %FOR.

4. Another comma delimiter to again indicate "openness".

5. A list of datums of type "name" which represent the subset of items of the Area Vector for which this directive applies.

**EXAMPLE: Specifying Area Vector Items--%FOR**

Assume the following:

[1] The Variable Control Area Vector consists of:
PSF
VZI
TRK
LAT
LON
TMP

[2] DEF and DIM directives are;
DEFVAR = RATE
DIMVAR = 0

[3] PSF, TRK and TMP are to be at rate 20, the remainder at rate 1.
Then, LETVAR = 1
LETVAR = 20 ,%FOR, PSF,TRK,TMP
or,
RATE = 1
RATE = 20 ,%FOR, PSF,TRK,TMP
will accomplish the desired results.

3.3.4.2 Value Sharing--%L, %T, %S

It is often the case that numerous directive files share one or more controls. For example, snapshot operations may share the same snapshot intervals, while transformation operations may share the same output rates, while both snapshot and transformation operations may share the same names of items in the Area Vector. Value Sharing is an extension of directive structure which allows for such sharing of controls.
The basic principle is simple. It would be very desirable to define the value of a shared control only once, and all operations thereafter would simply use that value. If a special value was inserted as a value for a control which was to participate in this value sharing, the defined value would be inserted into its place as part of the code which inputs the directive files.

The method is to define an actual value for a control in the first directive file to participate in value sharing. On any \texttt{LET} or Control Name directive used for subsequent directive files, one of the special values \texttt{\%L, \%T, or \%S} (defined below), may optionally be inserted in place of any real value for that same control. When one of these special values appears, it triggers a search for a control of the same name in a directive file defined previous to the current directive file. It searches all "designated" directive files until that control name is matched. When it does, it substitutes the value defined in that previous directive file for the special value in the current directive file. If none is found, the value specified in the \texttt{DIM} directive of this directive file is substituted for the special value instead.

The term "designated" used in the previous paragraph, refers to the different meaning ascribed to each of the three special values used in Value Sharing. The meaning of each is as follows;

\textbf{\%T} Value sharing occurs only between transformation operations, i.e. only previous transformation directive files are searched for that control employing the \%T special value.

\textbf{\%S} Value sharing occurs only between snapshot operations, i.e. only previous snapshot directive files are searched for that control employing the \%S special value.

\textbf{\%L} Value sharing occurs between all operations, i.e. all previous directive files are searched until a match occurs for that control employing the \%L special value.

\textbf{Rules}

* Any number of controls in a directive file may employ Value Sharing.

* The special values \%L, \%T, and \%S do not have any type.
* The special values are defined to match in type with all other legal types.

* No special value may be used in a DIM directive, since typing cannot be properly executed with them.

* Repeat factors, value abbreviations, etc. may be used at will with all special values in exactly the same manner as any other legally typed value.

EXAMPLE: Sample use of value sharing

Directive file for
Operation 1: (TRANSFORMATION)
DEFFUN = FUN1, FUN2, FUN3, FUN4, FUN5
DIMFUN = 1, 2, 3, 4, 5

Directive file for
Operation 2: (SNAPSHOT)
DEFFUN = FUN1, FUN2, FUN3
DIMFUN = 2, 4, 6

Directive file for
Operation 3: (TRANSFORMATION)
DEFFUN = FUN1, FUN2, FUN3, FUN4
DIMFUN = 3, 6, 9, 12

Directive file for
Operation 4:
DEFFUN = FUN1, FUN2, FUN3, FUN4, FUN5
DIMFUN = 4, 8, 12, 16, 20
LETFUN = 5, 7, 9, 11, 13

Satisfy yourself that
FUN1 = 3,
FUN2 = 6
FUN3 = 6
FUN4 = 16
FUN5 = 5

3.4 Re-ordering Area Controls--The ORD Directive

The DEF directive not only defines the controls that are available in an Area, but also the order in which these controls are to be stored. If, however, a large set of controls was defined, and a large subset of these controls were to receive new values through a LET or Control Name directive, it could become very cumbersome.

It would be convenient for there to be a directive which re-orders the controls into any desired order. This re-ordering would not affect the actual order in which the controls are stored, but only
actively change the order of controls for use by any LET directive following this directive. The directive which performs this re-ordering is the ORD directive.

The structure of this directive is:

Keyword
Composed of the three characters 'ORD' followed by a three character representation of the Area for which these controls are being re-ordered.

Assignment Delimiter
The equals sign (=).

List of Datums
The list of controls involved in the re-ordering. Each datum of this list must be of type "name".

Rules
* Every control name in the list of datums must have already been defined in a DEF directive for this Area.
* A control name may appear only once in the ORD directive.
* The dimensions, types and initial values set in the DIM directive, and subsequent values set in any LET or Control Name directives, remain unchanged for all controls, whether specified in this ORD directive or not.
* Only LET directives following the ORD directive (until the next occurrence), are affected by this re-ordering. This implies, for example, that values in any control may still be set through the use of a Control Name directive, even if that control is not specified in the ORD directive.

EXAMPLE: Re-Ordering
If,
DEFGEN = GC1, GC2, GC3, GC4, GC5
DIMGEN = 5*0

then,
ORDGEN = GC5, GC3
LETFEN = 37, 52

would produce exactly the same result as
LETFEN = 37, 52

Even though it requires one extra directive, it can often be more readable and eliminate the need for counting commas as place holders.

Notice that
GC4 = 22
is legal, even if it occurs after the .Z&Ne
directive, i.e. Control Name directives are
not affected by ORD directives. But LET---
directives are affected by
Control Name directives.

3.5 Linking to Other Controls--The LNK Directive

It is often the case that while executing with an item in an Area Vec-
tor, that a value from another item of an Area Vector could be needed.
As an example, if X, Y, and Z were items of the Variable Area Vector,
and if the value of X was to be the sum of Y and Z, there would need
to be a link of some sort so that values for Y and Z could be gathered
to perform the calculation.

The LNK directive has been designed to form these links from one item
of an Area Vector to one or more items of some other (or the same)
Area Vector. Linking is actually a two step process. The first step,
through the LNK directive, is only to identify the names of controls
which are to be linked, and the Area Vector to which they are linking.
The second step occurs when actual values are inserted into these
"marked" controls. A search through the link Area Vector is invoked,
until a match of the name of an item of the Area Vector with the value
occurs. A pointer to that item is then inserted into that control.

The structure of this directive is;

Keyword

Assigned Delimiter

List of Datums

Rules

No control may be linked to more than one Area Vector; i.e. any
control may appear in only one LNK directive.

When a control is linked to the General Area Vector, values
inserted into that control should be names of General controls,
not names of items in the General Area Vector. The reason is
that the General Area Vector actually contains only one item.

It is legal to link only to an Area Vector of this operation.
Thus, if a $L, $T, or $S value appears, the link searching is
turned off, even though Value Sharing is still in effect.

On LET or Control Name directives the value of the control, must
be typed "name", and must be an item of the area vector (exep-
tion of General Area Vector mentioned above).
If the value of a datum of a link control is NOLINK, no search is performed and the value -1 is inserted in its place.

If a specified value (other than NOLINK) is not an item in the area vector, an integer value of zero is inserted in its place.

EXAMPLE: Linking

Assume the variable area vector contains the following items:
VARA
VARB
VARC
VARD
VARE
VARF

Then,
DEFFUN = FC1 , FC2
DIMFUN = 1 , A
VARLNK = FC2
LETFUN = 6 , VARC

will cause a link search in the Variable Control area vector for the variable named VARC. It is found to be the third item in the list, so the "integer" 3 is inserted into FC2.

Please carefully note that the area named in the LNK directive, is the name of the Area Vector in which to search, NOT the Area from which the link came.

EXAMPLE: Illegal type definition
DEFFUN = FC1
DIMFUN = 1
VARLNK = FC1

3.6 Area Vector Manipulation--VEC, NEW, and DROP Directives

Controls may be manipulated in a variety of ways using the directives discussed above. This section details manipulation of the Area Vector.

There are three basic Area Vector manipulating directives. The structure for all three is very similar:

Keyword

Composed of three characters representing the directive (VEC, NEW, or DROP), followed by a three character representation of the Area Vector involved.
Assignment Delimiter

List of Datums

The equals sign (=).

A list of datums representing items of the Area Vector. Each datum of this list must be of type "name".

The first directive, VEC, causes the entire Area Vector to be redefined, using its list of datums as the only items to be included in the Area Vector. The NEW directive allows the list of items associated with it to be appended to the existing Area Vector. Finally, the DRP directive allows any item in its list which is in the Area Vector, to be removed from the Area Vector.

Rules

* Only items previously specified in the Area Vector will be retained when using the VEC directive, i.e. if a new item is included in the list, it is skipped.

* Only items that are not already in the Area Vector will be appended to it when using the NEW directive.

* The only items of the list provided for the DRP directive that will be deleted, are those which match items already in the Area Vector, i.e. those datums in this directive which are not in the Area Vector are ignored.

* All items in the list provided with the NEW directive will be inserted at the end of the existing Area Vector.

* All items in the list provided with any of these directives must be of type "name".

* The order that these directives appear in the directive file is the order in which they will be processed.

EXAMPLE: Area vector manipulation

Assume the Variable Control Area Vector contains the following items:

ONE
TWO
THREE

Then, NEWVAR = FOUR, FIVE, SIX would produce:

ONE
TWO
THREE
FOUR
FIVE
SIX

Followed by,

DRPVAR = TWO,THREEE
would produce
ONE
THREE
FOUR
FIVE
SIX
Followed by,
VECVAR = THREE, ONE
would produce,
THREE
ONE

NOTE: Currently all three directives may be applied only to the Variable Area Vector, and only the NEW directive for the Functional Area Vector.

3.7 File and Job Terminators

There is one form that is legal in a directive file other than assignments and comments, that has not previously been mentioned. It is the Terminator directive, and as its name implies, it is used to terminate files, or as an end of file marker. The form of a terminator directive is simply the appropriate keyword. Below, are the three legal terminators currently available, and descriptions of their use.

3.7.1 End of Directive File--ENDOP

The last directive of every directive file must be the terminator ENDP. It serves as an end of file mark, and without it, directives from the next file, if any, would be read and interpreted in the wrong file, with disastrous results.

3.7.2 End of Set of Directive Files-- ENDCASE

Each case that is included in a single run, has a set of directive files associated with it; one Programmer Directive and one User Directive file for each operation in the case. Since GEN-PRO allows the execution of several cases in one single run, the ENDCASE terminator directive is required to separate cases one from another.

3.7.3 End of Job Terminator--ENDJOB

The ENDJOB terminator serves as a terminator for the entire job, i.e. it marks the end of all cases to be executed.

-BEGINPD / BEGINUD / BEGINHD need discussion.
4. STRUCTURE

4.1 The Programmer Directive File (PD)

The Programmer Directive file, or the PD file, is a directive file which is defined by the module programmer. Its basic purpose is to allow definition, ordering, and setting of controls in the manner for which the code was written. It removes from the user the problem of having to learn the inner workings of an operation's code to properly input a correct directive file, and therefore, additionally, ensures that the user will be less likely to cause catastrophic errors (such as not defining required controls).

Generally speaking, the PD file will define, order, dimension, initialize, and type all controls, set up correct linking, set up value sharing, and provide a Default Case as the initial condition.

4.1.1 Required Directives

The list of directives in this section must appear in every PD file.

4.1.1.1 Any DEF Directives

All controls must be defined by the DEF directive(s), since any information provided for a control not defined will be considered as an error and therefore ignored.

The following sub-sections describe the set of control names that must appear in the DEF directive for each Area, in the order that they must appear. Any additional control names for an Area are to be appended to the end of these lists.

4.1.1.1 General Area Required Controls

<table>
<thead>
<tr>
<th>NAMEOP</th>
<th>The name of this operation, typed &quot;string&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMPD</td>
<td>The name of the Programmer Directive file, typed &quot;string&quot;.</td>
</tr>
<tr>
<td>NAMLIB</td>
<td>The name(s) of the code file(s) of a library of routines, which are to be loaded for the proper execution of this operation, typed &quot;string&quot;.</td>
</tr>
<tr>
<td>ITYPOP</td>
<td>The type of this operation (2=transformation, 1=input, 0=snapshot), typed &quot;integer&quot;.</td>
</tr>
<tr>
<td>QURLAP</td>
<td>The number of overlap cycles required for this operation, typed &quot;integer&quot;.</td>
</tr>
</tbody>
</table>
4.1.1.1.2 Variable Area Required Controls

NAMVAR  The name of the dependent variable, typed "name".
INDGET  The index from which to get data for this dependent variable, typed "integer".
INRATE  The rate of this variable upon entry to this operation, typed "integer".

In addition, the following controls must be appended to the end of the above list for transformation operations.

INDPUT  The index to which the output data for this variable is put, typed "integer".
RATE    The rate of this variable upon exit from this operation, typed "integer".

4.1.1.3 Function Area Required Controls

NAMFUN  The name field of the functions, typed "name".

This control need appear only if a Function Control Area is to be defined for this operation.

4.1.2 Any DIM Directive

Exactly one DIM directive is required for each DEF directive. Furthermore, every control whose name is defined in the DEF directive, must be dimensioned, typed, and initialized in the corresponding DIM directive. Special values (%L, %T, %S, and %FOR) are altogether disallowed from this directive. The number of groups in an aggregate need not be determined at this time (and in general could not be), but the number of elements in a group must be given.

4.1.3 ENBGP

The last directive must be ENBGP.

4.1.2 Allowed Directives

This section lists directives that may optionally appear in the PD file dependent only upon need.
4.1.2.1 Any ORD Directive

The ORD directive may be used to re-order the list of control names specified in the DEF directive. It is HIGHLY RECOMMENDED that any control which the user is disallowed to specify (such as NAMVAR, NAMFUN, INRATE, INDFGET, INDPUT, etc.), should be removed from user influence by creating an ORD directive with these dangerous controls removed from the list.

4.1.2.2 Any LET Directive Without %FOR

Values set into the various controls via the DIM directive can be re-initialized with the use of the LET directive. The list of datums is ordered in that they must match with the order in which the control names were specified in the DEF or the ORD directive, whichever occurred.

4.1.2.3 Control Name Directives Without %FOR

Any legal control name, i.e. specified in a DEF directive, may be used as a keyword in a Control Name directive.

4.1.2.4 Any LNK Directive

It is important to remember that LNK directives only set up the links, they do not actually replace any names of area vector items with addresses. As such, the link controls, if any, should be set up only by the module programmer.

4.1.3 Disallowed Directives

The following is a list of directives that must not appear in a PD file under ANY circumstances.

4.1.3.1 Any LET Directive With %FOR

At the time the PD file is created, the list of items that will appear in an Area Vector is not known, and thus no modification of individual list items is allowed in this file.

4.1.3.2 Any Control Name Directive With %FOR

Disallowed for the same reasons as above.

4.1.3.3 The VARVEC, VARNEW, and VARHDR Directives

Since the items of the Area Vector have not as yet been specified, it would be impossible to modify it.
with any of these commands.

4.2 **The User Directives File (UD)**

The User Directive file, or the UD file, is a directive file which is defined by the user and is designed to change dynamically and frequently, as opposed to the PD file which will change only when modifications to the code arise. It is has no control over which controls are in use by a particular operation, or the length of individual groups, or the names and kinds of linking, or the types of aggregates, or the final order of controls. It does, however, have ultimate control over the values set into various controls to cause each operation to behave in a fashion desired by the user. Its forte then, is setting of values for controls, and deciding which items are to be included in an Area Vector.

4.2.1 **Required Directives**

4.2.1.1 **The Control Name Directive NAMPD**

NAMPD represents the name of the corresponding PD file of this operation. The PD file is required since it contains all the definitions (via the DEF directive), dimensions, initialization, types (via the DIM directive), links (via LNK directives), etc., that are tantamount to correctly interpreting the requests made here in the User Directive file. There may be several different PD files for a given operation which provide perhaps different Default Cases, etc., so the user should have control over which of these he desires to invoke.

4.2.1.2 **END**

The last directive of the UD file must be END.

4.2.2 **Allowed Directives**

4.2.2.1 **Any LET Directive, With or Without %FOR**

The user has the capability to reset values of all controls for either all items of the Area Vector (via the LET directive), or just a subset of items of the Area Vector (via the LET directive with the %FOR construct appended at the end).

4.2.2.2 **Any Control Name Directive, With or Without %FOR**

The user additionally has the capability to reset values for a single control for either all items of the Area Vector (via the Control Name directive), or just a subset of items of the Area Vector (via the Control Name directive with the %FOR construct).
4.2.2.3 The VECVAR, NEWVAR, DVAR, and NEWFUN Directives

All directives dealing with manipulation of items of the Area Vector are allowed in the UD file. Remember, that these directives are processed in the order that they appear in the UD file.

4.2.3 Disallowed Directives

Under all circumstances the following directives are strictly prohibited from the UD file.

4.2.3.1 Any DEF Directive

It is completely illegal for a user to rename or permanently re-order any control. To do so would reek havoc with that operation's code, since each control has a unique meaning and location, and to alter that arbitrarily would produce incorrect results.

4.2.3.2 Any DIM Directive

This directive is disallowed for much the same reason that the DEF was; it would cause a plethora of catastrophic errors to redimension groups, change the type of controls, etc.

4.2.3.3 Any LNK Directive

The LNK directive only sets up the links by naming the link control, and the affected area vector. This preliminary work cannot be undone or serious errors would result. However, through LET or Control Name directives, the items of the area vector may be inserted, and all substitution of addresses WILL occur.

4.2.3.4 Any MOD Directive
5. SAMPLE PROGRAMMER DIRECTIVE FILE

This section provides an example of a Programmer Directive file. This file is not intended to be exhaustive, even though it is long, nor should it be viewed as the best method to accomplish each of the necessary tasks. The structure of the directives allows for much flexibility, and therefore, each programmer building a Programmer Directive file may find unique ways to define that file.

This section is intended to reveal the thinking of a hypothetical programmer who is about to build a PD file. It is broken into three parts to organize this thinking. The first part is a wordy description of the problem facing the programmer due to the unique requirements of his operation. It provides an English description of the problem. The second section details the steps of how the programmer constructs a definition table. This table is a concise definition of the problem in table form. The important pieces of information required to build a complete PD file are all contained therein. The third part demonstrates how each directive in the file is used to specify portions of the table until the entire table has been specified through a set of directives. This then is the PD file.

A final part of this section only lists the definition table and the corresponding Programmer Directive file, and is intended for quick reference, without all the verbage.

**Problem Statement**

I am a programmer who has just completed writing the code for an operation. I would like to build my Programmer Directive file so I can begin testing this code as well as the PD itself. My first task is to gather all the information pertinent to my operation; that is, I collect all the names of the controls that my operation requires. The following is the results of my data gathering.

The name of my operation is SAMPLE. It requires two PLIB files named ROUTINE1 and ROUTINE2, respectively. This is a transformation operation that requires no cycles of overlap. I have need for a beginning and ending time of an event in the form of hours, minutes, and seconds for each occurrence.

I will have dependent variables, which will be used, manipulated in some fashion, and new values inserted for them. To do this, I will need three special pieces of information about each one of the dependent variables. Two of these are simply involved in the nature of each dependent variable, but the third piece of information depends on a value of another dependent variable.
Also, to correctly compute the new value for each dependent variable, I have need of an unknown number of functions. These functions vary from run to run, and therefore, cannot be built into the code. For each function, I need to get the beginning time, and depending on its value, I need to get zero, one, or more dependent variables to be included in the function.

Now that I have gathered the basic information, I need to add more details to it. I will name my PD file SAMPLE. The hours, minutes, and seconds of the beginning and ending time will all be real values. Without any further input, I would like to define only one beginning and ending time which include the entire process. Therefore, I will assign all zeros to the beginning time, and all 99.'s to the ending time.

I would like to produce the simplest Default Case, so that unless the user redefines certain values, he will run with the simplest case. Therefore, I will simply refer many of the values to the values assumed in the last operation. For example, I will define my input rates for every dependent variable to be the same as the output rate from the last transformation operation, and the output rate to be the same as the input rate.

The three pieces of information needed for the dependent variables are defined as follows. The first is simply a string of characters that are a title associated with that dependent variable. Initially, this is to be left blank. The second is a real number used as a scaling factor in some computation. If the value of this real number is -99.99, the computation is not performed, and this is the desired initial value. The third, as mentioned above will contain the name of another dependent variable which will be used in the calculation. If no dependent variable is to be used, the code will recognize the special name NONE as a flag to indicate this.

Both pieces of information relating to the computation of the functions are names, the first is the name of the control which symbolizes the beginning time, while the second contains the name or names of dependent variables to be included in this computation. In either case, the value NONE is used to indicate that no information is required. This will be the initial case for both.

5.2 Tabular Statement of Problem

The next step is to set up a table which defines all the required information, so that a Programmer Directive can be built from it. The major headings for which information must be provided are: the name of each control, the order the control is to appear to the user, the datum type of each value of the control, the number of elements in a group, what, if any, linking is to be established, and finally, what initial values should be assigned to each control to produce a Default Case. Each of these headings will be discussed in separate sections below.
1 Naming of Controls

Each of the three Areas has a set of controls that are always required. These controls are global in nature, as the DRIVER must find these controls and work with them as well as the individual operation. If these controls are missing, then, the operation will not work correctly, even though it may appear they are not necessary. A more complete description of these controls and their meaning may be found in the STRUCTURE section of this document or in the CONTROLS section of the GLOSSARY document.

1. For the General Area--NAMEOP, NAMPD, NAMLIB, ITYPOP, OVRLAP

2. For the Variable Area--NAMVAR, INDGET, INRATE, INDPUT, RATE

3. For the Function Area--NAMFUN

In addition, from the Problem Statement above, it appears that two controls for the General Area, three controls for the Variable Area and two for the Function Area need to be named. Names of controls may not be arbitrarily named since no two controls may have the same name. Therefore, after careful thought, the controls were named;

BEGTIM For the general control which contains the beginning time.

ENDTIM For the general control which contains the ending time.

TITLE For the variable control which contains the title of the dependent variable.

COMPUTE For the variable control which contains the value used in the computation of the dependent variable.

SOURCE For the variable control which contains the name of another dependent variable used in the computation of this dependent variable.

TEST For the function control which is a link to the beginning time.

VARS For the function control which contains a link to the list of dependent variables.

With the naming of unique controls, the first task is complete; these values may be filled into the first heading of the definition table.
**OPERATION "SAMPLE" DEFINITION TABLE**

1. General Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMPD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITYPOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVRLAP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEGTIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDTIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Variable Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMVAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDGET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INRATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDPUR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TITLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Function Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMFUN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**5.2.2 Order of Controls**

In the Programmer Directive, there are two orders which are important for controls. The first order, which is the order that controls are mentioned in the DEF directive, determines the order in which they will be stored in the Control Array. These are critical because an operation (and the DRIVER) must know where each control is stored. Therefore, the code of each operation imposes strict requirements on the order in which controls are listed in the DEF directive. Actually, in the definition table listed for the Naming of Controls section above, the controls are already ordered in the way they must be in the Control Array. Note that each of the global controls must appear first, and in the order provided.
The second ordering which is important is the order of the controls made available to the User Directive file. Essentially, this means the order of controls specified in the last ORD directive. Basically, it is desirable to "hide" certain controls from the user which are used for the internal running of the program and would only produce incorrect results if altered by the user. In general, the user does not even want to see such controls, since the only thing he could possibly do to them is screw them up. Therefore, the ORD directive will remove these controls from the user's list of controls.

In the definition table below, the first order is simply the order of the controls as they appear in the list. The second ordering is listed explicitly under the ORDER heading. Controls to be removed from the user's control are marked with a dash, while those made available are marked in sequential order.

OPERATION "SAMPLE" DEFINITION TABLE

1. General Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMPD</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMLIB</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITPPOP</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVRLAP</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEGTIM</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDTIM</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Variable Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMVAR</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDET</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INRATE</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPMT</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RATE</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TITLE</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTE</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Function Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMFUN</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARS</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2.3 Typing of Controls

A control structurally is an aggregate, which is made up of one or more groups, which is made up of one or more elements. Each element in the aggregate must have the same type, i.e. the types discussed under Datum Types of the SYNTAX section of this document (except for the "special" type). The programmer must next assign a type to each of his controls. The global controls are fixed in type for all operations. Although linking has not yet been discussed, notice that all controls which must get information from other sources of the Control Array (link controls), must be typed "name". Therefore the definition table would look like,

OPERATION "SAMPLE" DEFINITION TABLE

1. General Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td>1</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMPD</td>
<td>-</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMLIB</td>
<td>2</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITYPOP</td>
<td>-</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVRLAP</td>
<td>-</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEGTIM</td>
<td>3</td>
<td>real</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDTIM</td>
<td>4</td>
<td>real</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Revision 1
12/13/79
2. Variable Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMVAR</td>
<td>-</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDGET</td>
<td>-</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INRATE</td>
<td>-</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDPUT</td>
<td>-</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RATE</td>
<td>1</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TITLE</td>
<td>4</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTE</td>
<td>3</td>
<td>real</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td>2</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Function Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMFUN</td>
<td>-</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>1</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARS</td>
<td>2</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2.4 Number of Elements in a Group

The next item of information to consider is how many elements should be put into one group. Remember, the user has control over the number of groups in an aggregate, but the programmer has exclusive control over the number of elements in a group.

In most cases, this number is obvious. Most all of the controls have only one element per group, and others, like the beginning and ending times, have three closely related values which should be grouped together. However, other cases could take more thought to take advantage of all the flexibility available to the programmer through the Programmer Directive.

For example, examine the VARS control in the Function Area. It has been stated that it will be used to contain values which will link it to the Variable Area Vector. There may be zero (marked by NOLINK), one, or more of these values. You could define the group to be the largest number of such links to be anticipated, or you could set the group length to only one, and allow for an unspecified number of groups to be read for each function. The latter is better.

With the heading #ELM representing the number of elements in a group, the definition table now looks like this;
### OPERATION "SAMPLE" DEFINITION TABLE

#### 1. General Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td>1</td>
<td>string</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMPD</td>
<td>2</td>
<td>string</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMLIB</td>
<td>3</td>
<td>string</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITYPOP</td>
<td>4</td>
<td>integer</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVRLAP</td>
<td>5</td>
<td>integer</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEGTIM</td>
<td>6</td>
<td>real</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDTIM</td>
<td>7</td>
<td>real</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 2. Variable Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMVAR</td>
<td>1</td>
<td>name</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDET</td>
<td>2</td>
<td>integer</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INRATE</td>
<td>3</td>
<td>integer</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDPUT</td>
<td>4</td>
<td>integer</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RATE</td>
<td>5</td>
<td>integer</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TITLE</td>
<td>6</td>
<td>string</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTE</td>
<td>7</td>
<td>real</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td>8</td>
<td>name</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 3. Function Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMFUN</td>
<td>1</td>
<td>name</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>2</td>
<td>name</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARS</td>
<td>3</td>
<td>name</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2.5 **Linking**

The next thing which the programmer must determine for each control, is the possibility of linking to another section. Actually, a control so affected is used exclusively for the purpose of pointing to the correct item of the correct Area Vector, i.e. this control contains no information other than names of items of an Area Vector. Therefore, the programmer decides how many link controls and to what Area Vector they are to point, and then marks them off.

In the definition table, the heading LNK is used to mark the link controls and the Area Vector to which they point. A dash for a control indicates that it is not a link control. It was
mentioned earlier, but repetition doesn't hurt, that all link controls must be typed "name".

OPERATION "SAMPLE" DEFINITION TABLE

1. General Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td>1</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NAMPD</td>
<td>-</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NAMLIB</td>
<td>2</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ITYPOP</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>OVRLAP</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>BEGTIM</td>
<td>3</td>
<td>real</td>
<td>3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ENDTIM</td>
<td>4</td>
<td>real</td>
<td>3</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

2. Variable Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMVAR</td>
<td>-</td>
<td>name</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>INDGET</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>INRATE</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>INPVAL</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>RATE</td>
<td>1</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>TITLE</td>
<td>4</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>COMPUTE</td>
<td>3</td>
<td>real</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td>2</td>
<td>name</td>
<td>1 VAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Function Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMFUN</td>
<td>-</td>
<td>name</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>1</td>
<td>name</td>
<td>1</td>
<td>GEN</td>
<td></td>
</tr>
<tr>
<td>VARS</td>
<td>2</td>
<td>name</td>
<td>1</td>
<td>VAR</td>
<td></td>
</tr>
</tbody>
</table>

5.2.6 Initial Values

The final thing that a programmer must insert into the definition table before building his actual Programmer Directive, is the actual value of each control that he wants to make available to the user. In general, the programmer should be thinking in terms of the case which will be most useful to the largest group of users. This arrangement of values is called the Default Case.
It is the purpose of the Default Case to provide all the necessary information in the Programmer Directive file to properly execute your operation without any required additional information from the User Directive. This is a goal, and may not be achievable for all operations. The programmer should spend most of his time in carefully defining the initial values. This includes the best usage of values from previous operations through the Value Sharing values, $L$, $T$, and $S$. Good choices here can provide very convenient defaulting.

The final entry in the definition table for each control then, is listed under the VALUES heading. To avoid any confusion, group parentheses are added.

**OPERATION "SAMPLE" DEFINITION TABLE**

1. **General Area**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td>1</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;SAMPLE&quot;)</td>
</tr>
<tr>
<td>NAMPD</td>
<td></td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;SAMPLE&quot;)</td>
</tr>
<tr>
<td>NAMLIB</td>
<td>2</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;ROUTINE1&quot;),(&quot;ROUTINE2&quot;)</td>
</tr>
<tr>
<td>ITYPOP</td>
<td></td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(2)</td>
</tr>
<tr>
<td>OVRLAP</td>
<td></td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(0)</td>
</tr>
<tr>
<td>BEGTIM</td>
<td>3</td>
<td>real</td>
<td>3</td>
<td>-</td>
<td>(0.,0.,0.)</td>
</tr>
<tr>
<td>ENDTIM</td>
<td>4</td>
<td>real</td>
<td>3</td>
<td>-</td>
<td>(99.,99.,99.)</td>
</tr>
</tbody>
</table>

2. **Variable Area**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMVAR</td>
<td></td>
<td>name</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>INDGET</td>
<td></td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>INRATE</td>
<td></td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>INDPUT</td>
<td></td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>RATE</td>
<td>1</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>TITLE</td>
<td>4</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;RESERVE TWENTY CHARS&quot;)</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>3</td>
<td>real</td>
<td>1</td>
<td>-</td>
<td>(-99.99)</td>
</tr>
<tr>
<td>SOURCE</td>
<td>2</td>
<td>name</td>
<td>1</td>
<td>VAR</td>
<td>(NONE)</td>
</tr>
</tbody>
</table>

3. **Function Area**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMFUN</td>
<td></td>
<td>name</td>
<td>1</td>
<td>-</td>
<td>(NONAME)</td>
</tr>
<tr>
<td>TEST</td>
<td>1</td>
<td>name</td>
<td>1</td>
<td>GEN</td>
<td>(BEGTIM)</td>
</tr>
<tr>
<td>VARS</td>
<td>2</td>
<td>name</td>
<td>1</td>
<td>VAR</td>
<td>(NONE)</td>
</tr>
</tbody>
</table>
Building the Programmer Directive File

Once the definition table has been completed, it is a simple task to transfer the information contained therein to directives which together form a directive file. This part will provide a step by step description of which part of the definition table is being implemented with each directive.

5.3.1 The DEF Directives

The following DEF directives will define the names of the controls and the order in which they will appear in the Control Array. They define the information provided in the NAME column of the definition table.

DEFGEN = NAMEOP, NAMPD, NAMLIB, ITYPOP, OVRLAP, BEGTIM, ENDTIM
DEFVAR = NAMVAR, INDGET, INRATE, INDPUT, RATE, TITLE, COMPUTE, SOURCE
DEFFUN = NAMFUN, TEST, VARS

5.3.2 The DIM Directives

The DIM directives provide the information in the TYPE, #ELM, and some of the VALUES columns of the definition table. Only some of the values can be initialized in this command, because none of the "special" type values may be used in this directive. Therefore, bogus values matching the type of each of the following controls here, and the "special" values inserted later; NAMVAR, INDGET, INRATE, INDPUT, and RATE.

DIMGEN = "SAMPLE", [("SAMPLE")], ["ROUTINE1","ROUTINE2"], 2, 0, [(0.0,0.0)], (3.99)
DIMVAR = NONE, 4*0, "RESERVE TWENTY CHARS", -99.99, [(NONE)]
DIMFUN = [(NONAME)], (BEGTIM), NONE

5.3.3 The LET Directives

LET directives are of course legal in the Programmer Directive file. But since the DIM directives do provide initial values, the LET directive is most useful in defining values that could not be provided by the DIM directives, namely "special" values. The following directive therefore provides the controls NAMVAR, INDGET, INRATE, INDPUT, and RATE with the Value Sharing markers they require;

LETVAR = 5*IT

5.3.4 The LNK Directive

The LNK directive is the directive which marks the link controls, associating them with an Area Vector. All controls, no matter what Area they are in, may be linked to the same Area Vector in the same LNK directive. The following will therefore
provide the non-dashed information specified in the LNK heading of the definition table;

\[
\text{VARLNK} = \text{VARS}, \text{SOURCE} \\
\text{GENLNK} = \text{TEST}
\]

5.3.5 The ORD Directive

Finally, the order of controls available to the user specified in the ORDER column of the definition table can be provided with the following ORD directives;

\[
\text{ORDFUN} = \text{TEST}, \text{VARS} \\
\text{ORDGEN} = \text{NAMEOP}, \text{NAMLIB}, \text{BEGTIM}, \text{ENDTIM} \\
\text{ORDVAR} = \text{RATE}, \text{SOURCE}, \text{COMPUTE}, \text{TITLE}
\]

5.3.6 The ENDOP Directive

The last directive of the Programmer Directive file must be ENDOP. It performs no other function then marking the end of the directive file.

\[
\text{ENDOP}
\]
Summary of Programmer Directive File

The following definition table translates into the following Programmer Directive file:

**OPERATION "SAMPLE" DEFINITION TABLE**

1. General Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td>1</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;SAMPLE&quot;)</td>
</tr>
<tr>
<td>NAMPD</td>
<td>-</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;SAMPLE&quot;)</td>
</tr>
<tr>
<td>NAMLIB</td>
<td>2</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;ROUTINE1&quot;), (&quot;ROUTINE2&quot;)</td>
</tr>
<tr>
<td>ITYPOP</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(2)</td>
</tr>
<tr>
<td>OVRLAP</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(0)</td>
</tr>
<tr>
<td>BEGTM</td>
<td>3</td>
<td>real</td>
<td>3</td>
<td>-</td>
<td>(0.0, 0.0, 0.0)</td>
</tr>
</tbody>
</table>

2. Variable Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMVAR</td>
<td>-</td>
<td>name</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>INDGET</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>INRATE</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>INDPUT</td>
<td>-</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>RATE</td>
<td>1</td>
<td>integer</td>
<td>1</td>
<td>-</td>
<td>(%T)</td>
</tr>
<tr>
<td>TITLE</td>
<td>4</td>
<td>string</td>
<td>1</td>
<td>-</td>
<td>(&quot;RESERVE TWENTY CHARS&quot;)</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>3</td>
<td>real</td>
<td>1</td>
<td>-</td>
<td>(-99.99)</td>
</tr>
<tr>
<td>SOURCE</td>
<td>2</td>
<td>name</td>
<td>1</td>
<td>WAR</td>
<td>(NONE)</td>
</tr>
</tbody>
</table>

3. Function Area

<table>
<thead>
<tr>
<th>NAME</th>
<th>ORDER</th>
<th>TYPE</th>
<th>#ELM</th>
<th>LNK</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMFUN</td>
<td>-</td>
<td>name</td>
<td>1</td>
<td>-</td>
<td>(NONAME)</td>
</tr>
<tr>
<td>TEST</td>
<td>1</td>
<td>name</td>
<td>1</td>
<td>GEN</td>
<td>(BEGTM)</td>
</tr>
<tr>
<td>WARS</td>
<td>2</td>
<td>name</td>
<td>1</td>
<td>VAR</td>
<td>(NONE)</td>
</tr>
</tbody>
</table>
OPERATION "SAMPLE" PROGRAMMER DIRECTIVE FILE

`BEGINP`

`/` VARIABLE AREA

`/`

`DEFVAR = NAMVAR, INDGET, INRATE, INDPUT, RATE, TITLE, COMPUTE,` ...

`DIMVAR = NONE, 4*0, "RESERVE TWENTY CHAR", -99.99, [(NONE)]`

`LETVAR = 5*4`

`ORDVAR = RATE, SOURCE, COMPUTE, TITLE`

`/`

`FUNCTION AREA`

`/`

`ORDFUN = TEST, VARS`

`DIMFUN = [(NONAME)], (BEGTIM), NONE`

`DEFFUN = NAMFUN TEST, VARS`

`/`

`GENERAL AREA`

`/`

`DEFGEN = NAMEOP, NAMPD, NAMLIB, ITYPOP, OVRLAP, BEGTIM, ENDTIM`

`ORDGEN = NAMEOP, NAMLIB, BEGTIM, ENDTIM`

`DIMGEN = "SAMPLE", ["SAMPLE"], ["ROUTINE1", "ROUTINE2"], 2,` ...

`/`

`AREA VECTOR INFORMATION`

`/`

`VARLNK = VARS, SOURCE`

`GENLNK = TEST`

`/`

`ENDP`

Revision 1
12/12/70
6. SAMPLE USER DIRECTIVE FILE

This section provides an example of a User Directive file. It intends to provide a step by step procedure of what information to gather for, and how to build, a UD file. Three parts of this section are designed to detail the procedure. The first part is an English description which discusses the requirements of the user for this operation. The second builds a values table, which, like the definition table used for the Programmer Directive, is the place where all information destined for a directive file is gathered and organized. The third part demonstrates how each directive in the file is used to specify portion of the tale until the entire table has been specified through a set of directives. This then is the UD file.

A final part of this section only lists the values table and the corresponding User Directive file, and is intended for quick reference, without all the verbage.

There is a Programmer Directive file which relates to the same hypothetical operation discussed in the previous section. Additionally, the Control Array for this operation, combining both directive files, is given in the Control Array section of the DATA STRUCTURES document.

6.1 Problem Statement

A user who has a dataset which I would like to process using the UNPRO program. One of the operations that I will be using is operation SAMPLE. My first task is to gather all the information pertinent to this operation; that is, I become familiar with all the controls available to me for this operation. Therefore, I read the documentation on each of the controls available for this operation in the CONTROLS section of the GLOSSARY document.

Having thus become familiar with the purpose and usage of each of the controls, I begin to define these controls with my needs in mind. The following paragraphs discuss my needs for the controls of operation SAMPLE.

The beginning time of my dataset is 12 hours, 14 minutes, and 35 seconds. The ending time of my dataset is 16 hours, 27 minutes, and 0 seconds.

I have run other operations previous to this one, and when this operation is entered, I have the following Variable Area Vector; VAR1, VAR2, VAR7, VAR3, VAR6. For this operation, I would like to delete VAR7 and VAR6, and add a new item to the Area Vector, VAR4. All items of the Area Vector have a rate of 20 except for VAR3, which has a rate of 50.

The values which I need for the scalar computation (the COMPUTE con-1) are 1.4 for VAR1, 1.8 for VAR2, 2.74 for VAR3, and the none for VAR4 (flagged by -99.99). The dependent variables used in each
computation (the SOURCE control) are; none for VAR1, VAR1 for VAR2 and VAR3, and VAR1 and VAR2 for VAR4.

I will have two functions for this operation, named FUN1 and FUN2. Neither one of them has any dependent variables to use for links (the VARS control), and both will use the beginning time as the time on which to test (the BEGTIM control).

6.2 Tabular Statement of Problem

The next step is to build a values table from the information derived from the above discussion. This table differs from the definition table to provide the specific information necessary for building a UD file; where the PD had a table of controls versus the various headings referring to each control, the UD will need a table which consists of values for all controls for each item of the Area Vector. Information such as the type, linkages, number of elements in a group, etc., can not be changed by the UD, and are therefore not relevant headings to be inserted into this table.

The process for building a values table is as follows. The first thing to do is to break up all controls into their respective Area. Then, focusing on one Area at a time, list all the controls in one direction, and all the items of the Area Vector in the other. The following is an example of the Variable Area for SAMPLE;

<table>
<thead>
<tr>
<th>NAMVAR</th>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR3</th>
<th>VAR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDCET</td>
<td>INRATE</td>
<td>INDPUT</td>
<td>RATE</td>
<td>TITLE</td>
</tr>
</tbody>
</table>

Once this has been done, begin filling in values that you would like to have for every control, every item of the Area Vector, from the description given in the discussion of the previous part above. Without further discussion, the values table for operation SAMPLE for this run, will look like;
1. General Area

Vector>>
/Controls General

---
NAMEOP | "SAMPLE"
NAMELIB | "ROUTINE1", "ROUTINE2"
BEGTIM | (12., 14., 35.)
ENDTIM | (16., 27., 00.)

2. Variable Area

Vector>>> | VAR1 | VAR2 | VAR3 | VAR4

---
RATE | 20 | 20 | 50 | 20
SOURCE | NONE | VAR1 | VAR1 | VAR1, VAR2
COMPUTE | 1.4 | 1.8 | 2.74 | -99.99
TITLE | "VAR1 T" | "VAR2 T" | "VAR3 T" | "VAR4 T"

3. Function Area

Vector>>> | FUN1 | FUN2

---
TEST | BEGTIM | BEGTIM
VARS | VAR1 | VAR1

6.3 Building the User Directive File

Once the values table has been built, it is now possible to easily build the User Directive file. Alongside the values table should be the definition table. The VALUES heading of that table informs the user what values will be originally inserted into each item of the Area Vector. If that value is the same as in the values table, no further definition would be required. Remember that the order of controls available to the user may be different from the order specified in the DEF directive of the PD file, and that new order is given under the heading of ORDER in the definition table.

This part of the document will discuss each directive as it is built and what portion of the values table is being implemented.
6.3.1 The NAMPD Control Name Directive

The NAMPD Control Name directive must appear somewhere in the UD file. It is used to locate the correct PD file. Without it, the PD file will not be read, which implies that all the controls will not be defined, dimensioned, typed, or initialized, a rather serious (in fact, fatal) error. In this case, a simple glance at the PD file will show that NAMPD is "SAMPLE", and therefore;

NAMPD = "SAMPLE"

6.3.2 Listing Area Vectors

One task that the Programmer Directive cannot do is prescribe the list of items in the Area Vectors. At the time the User Directive is created, however, the items of the Area Vector are known and may thus be defined.

The General Area Vector is always of length one, and in actuality, never needs to be defined.

The Variable Area Vector may be defined in the User Directive and manipulated at will with the DRP and NEW directives. If it is not defined with the use of the VEC directive in this UD, the Variable Area Vector from the last transformation operation will be used.

If there are any items in the Function Area Vector, they must be defined with the use of a VEC directive in the UD of the operation in which they occur. In addition, no vector manipulation is allowed (DRP or NEW).

The following will define the items of the Variable and Function Area Vectors;

DRPVAR = VAR7, VAR6
NEWVAR = VAR4
VECFUN = FUN1, FUN2

6.3.3 Setting Values

The only remaining thing that needs to be done is to set new values into the controls. This may be done in an assortment of ways; using all Control Name directives to set each control individually, using LET directives and completely defining every control, using ORD directives followed by LET directives to set only the appropriate controls, etc. The following directives use several of the possible techniques to reset values.
BEGTIM = (12.,14.,35.)
ORDGEN = ENDTIM
LETGEN = (16.,27.,00.)

RATE = 20
RATE = 50, %FOR, VAR3
ORDVAR = COMPUTE, TITLE
LETVAR = 1.4 , "VAR1 T", %FOR, VAR1
LETVAR = 1.8 , "VAR2 T", %FOR, VAR2
LETVAR = 2.74 , "VAR3 T", %FOR, VAR3
LETVAR = , "VAR4 T", %FOR, VAR4
SOURCE = VAR1
SOURCE = NONE, %FOR, VAR1
SOURCE = ,VAR2,%FOR, VAR4

6.3.4 The ENDOP Directive

The last directive of the UD must be the ENDOP directive.

ENDOP
6.4 Summary of User Directive File

The following values table, along with the definition table, translates into the following User Directive file:

OPERATION "SAMPLE" VALUES TABLE

1. General Area

<table>
<thead>
<tr>
<th>Vector&gt;&gt;&gt;</th>
<th>/Controls</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMEOP</td>
<td>&quot;SAMPLE&quot;</td>
<td></td>
</tr>
<tr>
<td>NAMLIB</td>
<td>&quot;ROUTINE1&quot;,&quot;ROUTINE2&quot;</td>
<td></td>
</tr>
<tr>
<td>BEGTIM</td>
<td>(12.,14.,35.)</td>
<td></td>
</tr>
<tr>
<td>ENDTIM</td>
<td>(16.,27.,00.)</td>
<td></td>
</tr>
</tbody>
</table>

2. Variable Area

<table>
<thead>
<tr>
<th>Vector&gt;&gt;&gt;</th>
<th>/Controls</th>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR3</th>
<th>VAR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RATE</td>
<td>20</td>
<td>20</td>
<td>50</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td>NONE</td>
<td>VAR1</td>
<td>VAR1</td>
<td>VAR1,VAR2</td>
<td></td>
</tr>
<tr>
<td>COMPUTE</td>
<td>1.4</td>
<td>1.8</td>
<td>2.74</td>
<td>-99.99</td>
<td></td>
</tr>
<tr>
<td>TITLE</td>
<td>&quot;VAR1 T&quot;</td>
<td>&quot;VAR2 T&quot;</td>
<td>&quot;VAR3 T&quot;</td>
<td>&quot;VAR4 T&quot;</td>
<td></td>
</tr>
</tbody>
</table>

3. Function Area

<table>
<thead>
<tr>
<th>Vector&gt;&gt;&gt;</th>
<th>/Controls</th>
<th>FUN1</th>
<th>FUN2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>BEGTIM</td>
<td>BEGTIM</td>
<td></td>
</tr>
<tr>
<td>VARS</td>
<td>VAR1</td>
<td>VAR1</td>
<td></td>
</tr>
</tbody>
</table>

Revision 1
12/13/79
RATION "SAMPLE" USER DIRECTIVE FILE

NAMPD = "SAMPLE"
/
AREA VECTOR MANIPULATION
/
DRPVAR = VAR7, VAR6
NEWVAR = VAR4
VECFUN = FUN1, "FUN2"
/
SETTING GENERAL AREA CONTROLS
/
BEGTIM = (12., 14., 35.)
ORDGEN = ENDTIM
LETGEN = (16., 27., 00.)
/
SETTING VARIABLE AREA CONTROLS
/
RATE = 20
RATE = 50, %FOR, VAR3
ORDVAR = COMPUTE, TITLE
LETVAR = 1.4, "VAR1 T", %FOR, VAR1
LETVAR = 1.8, "VAR2 T", %FOR, VAR2
LETVAR = 2.74, "VAR3 T", %FOR, VAR3
' ' ' ' VAR = "VAR4 T", %FOR, VAR4
RCE = VAR1
SOURCE = NONE, %FOR, VAR1
SOURCE = ,VAR2,%FOR, VAR4
/
ENDOP
Because of some problems which have arisen concerning the use of a GENPRO II tape header as a Header Directive File (HD) (an HD is analogous to a UD for an input operation) and because of some ambiguities in the documentation concerning the use of the VEC___, NEW___, and DRP___ directives, I would like to propose a replacement set of directives to manipulate the area vectors.

**********

1) VEC___ -- Define/re-define the Area Vector.

The VEC___ directive defines or re-defines the specified area vector for the current operation. For example:

VECVAR = %T, A, B, C

defines the Variable Area Vector for the current operation to consist solely of all of the variables of the previous T-op in the order specified in that T-op, followed by the variables A, B, C in the order A, B, C. If there is no previous T-op, then the Variable Area Vector of the current operation consists solely of the variables A, B, C.

2) APP___ -- Append to the Current Contents of the Area Vector.

The APP___ directive appends additional variables/functions to the variables/functions already defined in the area vector. For example:

APPFUN = A, B, C, %S

appends the functions A, B, C, in the order A, B, C to the functions already existing (if any) in the Function Area Vector of the current operation, followed by all of the functions of the previous S-op, in the order in which they appear in the Function Area Vector of the previous S-op. If there is no previous S-op, only the functions A, B, C are appended.

3) REM___ -- Remove from the Current Contents of the Area Vector.

The REM___ directive removes variables/functions from those already defined in the area vector. For example:

REMVAR = A, B, C, %L

removes the variables A, B, C, from the Variable Area Vector of the current operation, if they already exist. If any of the variables specified do not exist, then no action is taken for those variables and no (fatal) error occurs. Likewise, all of the variables in the Variable Area Vector of the current operation with names the same as any of the variables in the area vector of the last operation are also removed.
already defined in the area vector. For example:

```
CHAVAR = X, Z, %T
```

changes the current names of the first and third variable in the Variable Area Vector of the current operation to X and Z. The name of the second variable is unchanged. The fourth and following variable names are changed to the names corresponding to the first and following variable names in the Variable Area Vector of the previous T-op, if any. If more variables exist in the CHAVAR list than exist in the current area vector, the variables in excess cause no related action and no (fatal) errors. If there are fewer variables in the CHAVAR list than exist in the current area vector, the excess variables in the current area vector are not affected and there are no (fatal) errors.

5) SUB--- -- Substitute for Names in the Current Area Vector.

The SUB--- directive substitutes a new variable/function name for a specified name already existing in the area vector. For example:

```
SUBFUN = Z,C , X,A , Y,B
```

substitutes the function name Z for the function name C, the name X for A and Y for B. The substitutions do not have to match the name order that exists in the area vector. If a substitution is specified for a name that does not currently exist in the area vector, no substitution is performed, and no (fatal) error results. The special names (%T, %S, %L) should not be used.

The following syntax is also valid and may be used to improve readability.

```
SUBFUN = (Z,C) , (X,A) , Y,B
```

NOTES:

1) The use of the special names %T, %S, %L in the examples is meant to show that their use is applied consistently to all of the Area Vector Manipulation Directives. The use of the special names is not necessarily useful for certain of the directives. Note that the search is carried back only one operation of the appropriate operation type. The effect of using the special names is the same as if the names represented by the special names appeared in place of the special name in the directive.

2) The Area Vector Manipulation Directives are meant to be applied consistently to both the Variable and Function Areas. There are no special cases. The use of certain directives in the Function Area is not necessarily useful. These directives may not be applied to the General Area Vector; i.e., there is no such directive as VECGEN.

3) The characters "(", "), ("", ")", may appear (properly paired and nested) in any of the Area Vector Directives. They do not connote any notion of "groups" or "aggregates" and are completely ignored by the TRANSLATOR.

4) Non-unique names are specifically allowed for variable/function names. This allows a VECVAR in an HD (Header Directive File) written on a system that defines 10 characters/NSU to be read on a machine that defines only 4 characters/NSU with any non-unique names resulting from the truncation of names greater than 4 characters to 4 characters changed to unique names using CHAVAR or
VECVAR = VARIABLE1, VARIABLE2, VARIABLE3
(read as VECVAR = VARI, VARI, VARI on the 4 character/NSU machine)

CHAVAR = VAR1, VAR2, VAR3

or

SUBVAR = (VAR1, VARIABLE1), (VAR2, VARIABLE2), (VAR3, VARIABLE3)
(read as SUBVAR = (VAR1, VARI), (VAR2, VARI), (VAR3, VARI) ) (see NOTE 5)

5) The TRANSLATOR processes the datums in the directives in the order they appear. The search through the existing area vector always starts from the beginning. For example:

VECVAR = A, B, C

SUBVAR = (C, A), (X, B), (Y, C)

yields the area vector Y, X, C.

In the example below, the braces "{", and "}", enclose truncated characters and are not part of the syntax of the directive:

VECVAR = VARI{ABLE1}, VARI{ABLE2}, VARI{ABLE3}

SUBVAR = (VAR3, VARI{ABLE3}), (VAR2, VARI{ABLE2}), (VAR1, VARI{ABLE1})

yields the area vector VAR3, VAR2, VAR1. (Note that as the result of non-uniqueness of name and the order in which the TRANSLATOR processed the non-unique names and their substitutions, VARIABLE1 has the new name VAR3, etc.)

*****************************************************************************
## CODE DEVELOPMENT CONSIDERATIONS

- Accessing GENPRO Work Array
- Reentrancy
- Data Flow Implications for Code Development
- Stages of Execution
- Requesting Save and Scratch Space
- Requesting Overlap
- Accessing Control Values
- Passing Data
- Example Code Segment

## COMMONLY USED SUBROUTINES

- Miscellaneous
- Dependent Variable Data Routines
- Control Array Routines
- Driver Information Routines
- Independent Variable Routines
- Error Reporting Routines

## LESS FREQUENTLY USED SUBROUTINES

- Control Array Routines
- Dependent Variable Routines
- Driver Information Routines
INTRODUCTION

This tutorial describes how one can develop GENPRO code modules. It assumes proficiency in the use of the GENPRO system and familiarity with common GENPRO terminology. In particular, the reader is assumed to be proficient in the use of PD's and UD's.

In order for a code to be a candidate for a GENPRO module it must be able to process its data sequentially. The data manager of the DRIVER controls the flow of data between Operations but it is always inherently sequential.

The basic items which an Operation implementor must take into consideration in the development of the code for that Operation are:

- Providing correct linkage to access GENPRO work array
- Execution stage
- Requesting and using SAVE and SCRATCH space
- Accessing (and perhaps altering) control arrays which form the run-time translation of the PD and UD for the Operation
- Requesting overlap to define minimum data windows for those Operations that need them
- Accessing data and possibly transforming input data to create output data to be passed to the next Operation

Each of these points will be addressed in this tutorial. Library subroutines are discussed according to functionality, details of the calling sequences are at the end of this document.

CODE DEVELOPMENT CONSIDERATIONS

Accessing GENPRO Work Array

The library routines described in this document form the interface to the GENPRO system. They provide the code developer with a logical means to access and set information maintained by the DRIVER. Much of this information is stored in common blocks hidden from the implementor. Other information, such as the translated control array, the Dependent Variable data, etc., is stored in the GENPRO work array. Many of the routines provided allow the user to access and use this work array in a logical manner. The user must include the GENPRO common block
which defines this work array. The code to do this is as follows (note the equivalencing for mnemonic purposes):

```plaintext
COMMON SPACE($ISPACE), OVRFLO, LENSP,
1 LENBUF($IOP), LENC($IOP), LENDAT($IOP), LENSAV($IOP), LENSFR,
2 LOCBUF($IOP), LOCCA($IOP), LOCDAT($IOP), LOCSAV($IOP), LOCSCR
DIMENSION CON(1), SAV(1), BUF(1), DATA(1), SCR(1)
DIMENSION ICON(1), ISAV(1), IBUF(1), IDATA(1), ISC(1)
EQUIVALENCE (SPACE(1), CON(1), SAV(1), BUF(1), DATA(1), SCR(1))
EQUIVALENCE (SPACE(1), ICON(1), ISAV(1), IBUF(1), IDATA(1), ISC(1))
```

In this code, $ISPACE and $IOP must be replaced with values, of course. These figures are implementation dependent and can be obtained from the source code of any Operation running on your machine.

**Reentrancy**

A standard GENPRO Operation code is reentrant, that is, multiple activations of the module implementing the Operation should be possible during the execution of a job. This corresponds to an Operation occurring more than once on the job stack. In order to be reentrant, any information or data generated during a call to an Operation which must be retained for use by subsequent calls to that Operation must be saved on exit from the Operation, and restored upon the next call to that Operation. This way, more than one activation of the module can be in effect in the operation stack. Such information is most conveniently placed in common blocks which can then be copied to SAVE space within the GENPRO work array (see discussion on requesting SAVE space below). On entry into the subroutine, the common blocks can then be restored from the work array. Of course, one is not restricted to the use of common blocks, any arrays or single variables may be copied into SAVE space for storage, and then restored later. Use of common blocks merely prevents the programmer from having to compute indices into storage of where the values have been kept.

**Data Flow**

Data partitioning and flow is controlled by the DRIVER for optimum efficiency. Each time an Operation is called, a certain amount of Dependent Variable data is available to that Operation for processing. For Snapshot Operations processing includes some type of display of the data, with no transformations done, and hence no "output data" is produced. For Transformation Operations, on the other hand, output data is produced and stored in a different location than the input data for use as input data to later Operations. In addition to "current data" which is being operated on, it is possible to define data windows which include amounts of "past" and "future" data which may then be accessible during a call to an Operation (see the discussion below.
On OVERLAP). For the Operation implementor, it is important to understand that data is passed to an Operation sequentially, and in varying amounts. There are some simple utilities (DIO and DPF as described below) which permit an Operation to access the needed data areas for both input and output data, for current, past and future data values.

**Stages of Execution**

There are five stages of execution, four of which are of interest to Operations. Stage 1 is the translation stage and should never be encountered by a called Operation. Stage 2 allows an Operation to access the Control Array, perform checking and editing of the array, and perhaps some initialization and informational displays as well. No Dependent Variable data is processed during this step. Execution Stage 3 is essentially like execution stage 2, still with no real data being processed. Execution stage 4 is the first call to an Operation with real data. Execution stage 5 is comprised of the remaining calls with real data. These stages are checked by an Operation and the appropriate code executed depending on the current stage.

Besides the execution stage, it is important to know the Independent variable interval being processed. Typically, stage 2 control editing is done only for the first Independent Variable Interval, since multiple intervals use the same control array.

The template below indicates a common code structure for a module. The calls would be done at the beginning of the module, after a call to SETUP (see discussion below on SETUP). Depending on the I.V. interval and the execution stage, different sections of the code would be executed.
CALL GETSTG(NSTAGE)  I NSTAGE = EXECUTION STAGE
CALL GIVNT(IVINT)  I IVINT = I.V. INTERVAL FOR THIS CALL

IF (NSTAGE.EQ.2.AND.IVINT.EQ.1) THEN
{ code for control array editing, etc. }  
ELSEIF (NSTAGE.EQ.3) THEN
{ stage 3 processing, not often used }  
ELSEIF (NSTAGE.EQ.4) THEN
{ process first data }  
ELSEIF (NSTAGE.EQ.5) THEN
{ normal processing of current data}  
ELSE
{ error in returning execution stage }  
END IF

Requesting Save

and Scratch Space

The DRIVER maintains save space for each operation that requests it. Save space is a contiguous segment of the work array and is therefore defined by a starting index and a length. An Operation can request the amount of the SAVE space, and the DRIVER will decide where to reserve the space in the work array. An Operation can make a call to the routine GETSAV to find where the save space is and can call SETSAV to request the space. Each Operation on the Operation Stack has separate save space.

Scratch space, on the other hand, is shared by all Operations, and enough is reserved to satisfy the largest request. An Operation can request scratch space with a call to SETSCR and obtain the index and length of the scratch space with a call to GETSCR.

A common use for save space is to save information that is needed for subsequent calls to an Operation, but which might be overwritten by other instances of the same Operation. That is, the code for an Operation must be usable by more than one instance of an Operation, and the data retained between calls must then be saved and restored.

In the following example, the common block /OPDATA/ contains variables which must be saved between calls to an Operation. Since there are two arrays of 10 elements and three single variables, a total of 23 NSU's of save space are requested. Notice that the block is first saved during stage 4 which is the first call to the Operation (with real data), and that it is restored on entry to the subroutine on subsequent calls. The MOVESP subroutine (a part of the GENSUB library) performs the move.
COMMON /OPDATA/ A(10),B(10),C,D,E
REAL SAVBLK(23)
EQUIVALENCE (A(1),SAVBLK(1))

CALL GETSTG(NSTAGE)
IF (NSTAGE.EQ.2) THEN

{ stage two code }

CALL SETSAV(NSAV)
ELSEIF (NSTAGE.GE.4) THEN
CALL GETSAV(LSAV,NSAV)
CALL MOVESP(SAV,LSAV,1,SAVBLK,1,1,NSAV)

{ stage 4 and 5 code }

CALL MOVESP(SAVBLK,1,1,SAV,LSAV,1,NSAV)
END IF

If the equivalencing technique above had not been used, the code would have been:
COMMON /OPDATA/ A(10), B(10), C, D, E

CALL GETSTG(NSTAGE)
IF (NSTAGE.EQ.2) THEN
   NSTAGE = EXECUTION STAGE
   { stage two code }
   CALL SETSAV(NSAV)
   ELSEIF (NSTAGE.GE.4) THEN
      CALL GETSAV(LSAV, NSAV)
      CALL MOVESP(SAVBLK, 1, 1, SAV, LSAV, 1, NSAV)
      CALL MOVESP(SAV, LSAV, 1, A, 1, 1, 10)
      LSAV = LSAV+10
      CALL MOVESP(SAV, LSAV, 1, B, 1, 1, 10)
      LSAV = LSAV+10
      CALL MOVESP(SAV, LSAV, 1, C, 1, 1, 1)
      LSAV = LSAV+1
      CALL MOVESP(SAV, LSAV, 1, D, 1, 1, 1)
      LSAV = LSAV+1
      CALL MOVESP(SAV, LSAV, 1, E, 1, 1, 1)
      LSAV = LSAV+1
      { stage 4 and 5 code }
      CALL MOVESP(A, 1, 1, SAV, LSAV, 1, 10)
      LSAV = LSAV+10
      CALL MOVESP(B, 1, 1, SAV, LSAV, 1, 10)
      LSAV = LSAV+10
      CALL MOVESP(C, 1, 1, SAV, LSAV, 1, 1)
      LSAV = LSAV+1
      CALL MOVESP(D, 1, 1, SAV, LSAV, 1, 1)
      LSAV = LSAV+1
      CALL MOVESP(E, 1, 1, SAV, LSAV, 1, 1)
      LSAV = LSAV+1
      END IF

Requesting Overlap
Some Operations may need a certain minimum data window in order to perform the needed data transformations. In such cases, an Operation may compute (or allow the user to specify via directives) the needed window and request an appropriate amount of overlap. The GETOVR and SETOVR routines return/set such overlap.

Accessing Control Values
Processing at stage 2 is essentially a matter of accessing, testing, and perhaps editing the control array created from the PD and UD for the Operation in question. A full set of access routines are provided to allow the programmer to view the data structure in a fairly logical manner so that accessing the array or altering values can be done via subroutine calls.
In order to use these routines, an Operation must first make a call to a routine called SETUP which will calculate and store certain needed addresses (hidden from the programmer) and which makes an association between an array of directive names supplied by the programmer and their relative positions within the control array. This mechanism permits the programmer to work with controls in an order-independent manner and facilitates the addition and deletion of controls without changing existing code.

In the example below, notice that SETUP takes as input an array of names (of controls) and returns an array twice as large that represents a sequence of pairs of integers indicating the area followed by the ordinal position of the control. These are passed to the access routines. Note that other than supplying the arguments automatically for each call, the user need not be aware of the positions of the controls at all. Since name types must be unique in the first four characters, the SETUP utility matches ONLY the first four characters of a name and will use the first named control it finds with a given four-character prefix (subsequent ones are ignored). The names listed would appear in a PD supplied by the user and given in the PD/UD file as input. Finally, the /ORD/ common block has been equivalenceed to a single array name for use as an argument to SETUP, but has been built out of two element arrays for mnemonic purposes. This helps to self-document the code.

```
PARAMETER (NCON = 5, IDIM = NCON*2) ! 5 CONTROLS IN PD
INTEGER NAMES(NCON) ! NAMES OF CONTROLS
INTEGER ORDER(IDIM) ! EQUIVALENCED ARRAY
EQUIVALENCE (ORDER(1),INAME(1))
COMMON /ORD/ INAME(2), INUMC(2), IITYP(2), IOVER(2), INAMV(2)
DATA NAMES/ 4HNAME, 4HNUMC, 4HITYP, 4HOVER, 4HNAMV /
CALL SETUP(NAMES,ORDER,NCON) ! PERFORM SET UP TO CONTROL ARRAY
CALL GETSTG(NSTAGE) ! NSTAGE = EXECUTION STAGE
ITEM = 1
CALL GETVAL(INAMV(1),INAMV(2),ITEM,NAME)
```

As an example of an access routine, the call to GETVAL above returns the (single NSU) value of the control called NAMV for the first item, and stores it in NAME. This is typical of all of the data access routines, the first two arguments are always the area, followed by the control number. Note again that the programmer does not have to be concerned with these as such, as long as he/she is using SETUP.
Presented immediately below is a list of the most common control array access routines and a brief description of what they do. Calling sequences and more, less frequently used routines are provided at the end of this document.

- **SETUP**: makes association between control names (unordered) and their relative positions with the PD for the calling Operation
- **LENAGR**: returns the length (storage reserved) of an aggregate - this is equivalent to the length of the longest aggregate value set to the control in question
- **LENAGT**: returns the length of a specific aggregate value terminated by a special value
- **LENGRP**: returns the length of a control group
- **LENELM**: returns the length of a control element
- **NUMGRP**: returns the maximum number of groups in an aggregate
- **NUMELM**: returns the number of elements in a group
- **NITEMS** returns the number of items in an area (number of items in a VECVAR or VE CFUN list for the variable and function areas and always 1 for the General Area)
- **GETELM**: returns the location and size of an element
- **GETGRP**: returns the location and size of a group
- **GETAGG**: returns the location and size of an aggregate
- **SETVAL**: sets value of a single-NSU control
- **GETVAL**: gets value of a single-NSU control
- **SETELV**: sets value of a single-NSU element
- **GETELV**: gets value of a single-NSU element
- **SETGRV**: sets value of a single-NSU group
- **GETGRV**: gets value of a single-NSU group
- **UNLINK**: unlinks an array of control numbers, returning the names of the linked controls
Notes: For controls, groups, or elements that consist of a single-NSU value it is much more convenient to make calls to the routines such as GETVAL and SETVAL. For controls with multiple-NSU values, one makes calls to routines such as GETAGG and GETGRP to obtain first word addresses and lengths. Values are then accessed and altered by taking the returned indices and operating directly on the CON (equivalently ICON) array itself.

Passing Data

In-place transformation and snapshot Operations need to obtain data from Dependent Variable data buffers. Transformation Operations need to also and in addition must be concerned with passing data on to the next Operation. That is, Transformation Operations get data from the previous Transformation Operation and put the transformed data into the input buffer of the next Transformation Operation. In addition to the input data for an Operation, the "current data" to be operated on, cycles of overlap may need to be accessible to an Operation. We refer to "current" cycles of data as that data which is to be transformed by the current call to the Operation, "past" data to be data which occurs "before" the current data (in relation to the Independent Variable) and that therefore has already been processed but is still accessible, and "future" data as that data which will be transformed by subsequent calls to the Operation.

This tutorial assumes that the reader is familiar with the GENPRO basic concepts of Independent/Dependent Variables, cycles of data, data rates, etc.

In order to know how many Dependent Variables are to be processed by an Operation, a call to NITEMS with the Area specified as 2 (for Variable Area) is made. The data routines use the ordinal position of a Dependent Variable in the VECVAR list as an identifying key, referred to in this document as simply "the variable number".

Presented immediately below is a list of the most common data access routines and a brief description of what they do. Calling sequences and more, less frequently used routines are provided at the end of this document.

- **IVVALU**: returns the value of the Independent Variable associated with a particular Dependent Variable data point

- **DPF**: returns locations and amounts of past and future data

- **DIO**: returns locations and amounts of the data currently available to the Operation
• MOVDAT: provides an easy call to move data through an Operation untouched. This is useful for those Transformation Operations which alter only a subset of the available variables.

• CLRDAT: provides an easy way to pass on a Dependent Variable which has all of its values replaced by a single special one, notably GAPVAL to indicate perhaps that an error was found in processing the Variable so only GAPVAL's are being passed on.

Example Code Segment

As an illustration of some basic calls, let us suppose we have an Operation PD which has a Control called BIAS defined for Dependent Variables which is a value to be added to each value of the Dependent Variable. In the code segment below, IBIAS is a two element array that has been set by SETUP to indicate the Area (in this case Area 2) and ordinal position of BIAS in the PD. The following code accesses the value of BIAS for each Dependent Variable and adds it to the corresponding data values and stores the result in the input buffer for the next Operation.

PARAMETER (IGEN=1, IVAR=2, IFUN=3) ! DIFFERENT AREAS
...
CALL NITEMS(IVAR,NUMVAR) ! NUMVAR = NUMBER DEPENDENT VAR
DO 10 JV=1,NUMVAR ! FOR EACH VARIABLE DO...
CALL DIO(JVAR,IFROM,ITO,NIN,NOUT) ! GET LOC AND AMOUNT OF CURRENT
CALL GETVAL(IBIAS(1),IBIAS(2),JVAR,BIAS) ! GET VALUE OF BIAS
DO 20 II=1,NOUT ! FOR EACH DATA POINT AD
DATA(ITO+II-1) = DATA(IFROM+II-1) + BIAS
20 CONTINUE
10 CONTINUE
COMMONLY USED SUBROUTINES

Miscellaneous  INTEGER FUNCTION ISERCH(IARRAY,LENSCH,NAMSCH)

Brief Description:

This function searches a list to match an element and returns the ordinal position of the element in the list or a 0 if it was not found.

INPUT ARGUMENTS:

- IARRAY: Array (list) to be searched
- LENSCH: Maximum number of elements to be searched in list
- NAMSCH: Element to be matched

OUTPUT VALUE:

- ISERCH: Ordinal position of the matched value found or 0 if it wasn't found

LOGICAL FUNCTION MATCH(NAME1,NAME2,N)

Brief Description:

This function returns a value of .TRUE. if the first N characters in NAME1 and NAME2 match, otherwise it returns .FALSE.

INPUT ARGUMENTS:

- NAME1: Array containing characters of first string to match
- NAME2: Array containing characters of second string to match
- N: Number of characters to match

OUTPUT VALUE:

- MATCH: .TRUE. if NAME1 and NAME2 match in the first N characters, otherwise .FALSE.
GETIO(IUNIT, N)

Brief Description:

This subroutine returns the FORTRAN logical unit number assigned to any of the units assignable in the DRIVER: KPROC, KERR, KCHECK, KFICHE, KFILM, KPRINT, KDUMP.

INPUT ARGUMENTS:

* IUNIT: Single NSU argument containing at least the first four characters of the name of the output unit; only the first four characters of the provided name are matched.

OUTPUT ARGUMENTS:

* N: FORTRAN logical unit number

CLHEAD(IUNIT, NAME)

Brief Description:

This subroutine prints out a header for informational displays generated by GENPRO Operations.

INPUT ARGUMENTS:

* IUNIT: FORTRAN logical unit to send header to
* NAME: Name of the Operation requesting the header

CLTAIL(IUNIT, NAME)

Brief Description:

This subroutine prints out a trailer for informational displays generated by GENPRO Operations.

INPUT ARGUMENTS:

* IUNIT: FORTRAN logical unit to send header to
* NAME: Name of the Operation requesting the header
UNLINK(IAREA, ICNTRL, IN, N, IOUT)

Brief Description:

After translation, linked controls have values which represent the ordinal position of the Function, Variable, or General control to which the link was made. Sometimes it is useful to reverse this process and obtain the names that were linked.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- IN: Array of linked controls
- N: Dimension of IN

OUTPUT ARGUMENTS:

- IOUT: Array of original unlinked controls

SETUP(NAMES, IORDER, N)

Brief Description:

This subroutine returns the area and control number of the control names passed to it.

INPUT ARGUMENTS:

- NAMES: Array of names of controls which are to be matched against controls found in the PD for the operation calling SETUP
- N: Number of names

OUTPUT ARGUMENTS:

- IORDER: Array of length 2*N returning the area and then control number of each control passed in.
Dependent Variable DPF(NVAR,IFIRST,ILAST,NPAST,NFUTUR)

Brief Description:

This subroutine returns information about past and future data points available on the current call to the Operation.

INPUT ARGUMENTS:

• NVAR: Number of the Dependent Variable (position in the VECVAR list)

OUTPUT ARGUMENTS:

• IFIRST: Address in the GENPRO work array of the first available data point of overlap
• ILAST: Address in the GENPRO work array of the last available data point of overlap
• NPAST: Number of past data points (of overlap) available on this call
• NFUTUR: Number of future data points (of overlap) available on this call

DIO(NVAR,LIN,LOUT,NIN,NOUT)

Brief Description:

This subroutine returns information about the data points passed on the current call to the Operation

INPUT ARGUMENTS:

• NVAR: Number of the Dependent Variable (position in the VECVAR list)

OUTPUT ARGUMENTS:

• LIN: Address in the GENPRO work array of the first current data point
• LOUT: Address in the GENPRO work array of the last current data point
• NIN: Number of current data points available on this call
• NOUT: Number of future data points available on this call
**MOVDAT(N)**

**Brief Description:**

This subroutine moves the data for a Dependent Variable from an Operations input buffer to its output buffer. This is useful for Dependent Variables which must be passed through an Operation but are not necessarily operated on.

**INPUT ARGUMENTS:**

- `N`: The number of the Dependent Variable (position in the VECVAR list).

**CLRDAT(N, VAL)**

**Brief Description:**

This subroutine fills the output buffer of a Dependent Variable with the value VAL. This can be useful for passing on GAPVAL's or some other special value for a Dependent Variable for use in subsequent operations.

**INPUT ARGUMENTS:**

- `N`: The number of the Dependent Variable (position in the VECVAR list).
- `VAL`: The value to set to the Dependent Variable data.
LENAGT(IAREA,ICNTRL,ITEM,ITERM,N)

Brief Description:

This subroutine returns the length of an aggregate which is terminated by a particular value. Contrast with LENAGR which returns the largest aggregate length assigned to a control.

INPUT ARGUMENTS:
- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- ITERM: Value which terminates the aggregate

OUTPUT ARGUMENTS:
- N: The length of the aggregate

LENAGR(IAREA,ICNTRL,N)

Brief Description:

This subroutine returns the aggregate length reserved for a control. This represents the number of NSU's required by the longest value set to that control. Contrast this with LENAGT which returns the length set to a specific instance of a control (which is always less than or equal to what LENAGR returns).

INPUT ARGUMENTS:
- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:
- N: The maximum aggregate length.
LENGRP(IAREA, ICNTRL, N)

Brief Description:
This subroutine returns the length of a group for a control.

INPUT ARGUMENTS:
- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:
- N: The group length.

LENELM(IAREA, ICNTRL, N)

Brief Description:
This subroutine returns the element length of a control.

INPUT ARGUMENTS:
- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:
- N: Length of a element.
NUMGRP(IAREA, ICNTRL, N)

Brief Description:

This subroutine returns the maximum number of groups in an aggregate for a control (compare with LENAGR which returns the total size).

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:

- N: Number of groups in the largest aggregate assigned to the control.

NUMELM(IAREA, ICNTRL, N)

Brief Description:

This subroutine returns the number of elements in a group for a control.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:

- N: Number of elements in a group.
NITEMS(IAREA, N)

Brief Description:

This subroutine returns the number of items (vectors) in an area. The General area has only one (always), number of items for the Variable or Function areas is the number of names occurring in a VECVAR or VECFUN directive.

INPUT ARGUMENTS:

- IAREA: Control area

OUTPUT ARGUMENTS:

- N: Number of items in the Area.

GETELM(IAREA, ICNTRL, ITEM, NGROUP, NELEM, IELEM, ISIZE)

Brief Description:

This subroutine returns the location and size of an element (of a group) of a control.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number
- NELEM: Element number

OUTPUT ARGUMENTS:

- IELEM: Location (index) of first NSU of the element in the GENPRO work array.
- ISIZE: Size of the element in NSU's
GETGRP(IAREA, ICNTRL, ITEM, NGROUP, IGROUP, ISIZE)

Brief Description:

This subroutine returns the location and size of a group of a control.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number

OUTPUT ARGUMENTS:

- IGROUP: Location (index) of first NSU of the group in the GENPRO work array.
- ISIZE: Size of the group in NSU's

GETAGG(IAREA, ICNTRL, ITEM, TERM, IAGG, ISIZE)

Brief Description:

This subroutine returns the location and size of an aggregate of a control.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number

OUTPUT ARGUMENTS:

- IAGG: Location (index) of first NSU of the aggregate in the GENPRO work array.
- ISIZE: Size of the aggregate in NSU's
SETVAL(IAREA, ICNTRL, ITEM, VAL)

Brief Description:

This subroutine sets the value of a control in the control array. It works only for single-NSU valued controls.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- VAL: Value to be set

GETVAL(IAREA, ICNTRL, ITEM, VAL)

Brief Description:

This subroutine returns the value of a control in the control array. It works only for single-NSU valued controls.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number

OUTPUT ARGUMENTS:

- VAL: Value to be set
GETELV(IAREA, ICNTRL, ITEM, NGROUP, NELEM, IVALU)

Brief Description:

This subroutine returns the value of an element of a group of a control. It works only for single-NSU controls.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number
- NELEM: Element number

OUTPUT ARGUMENTS:

- IVALU: The value of the element

SETELV(IAREA, ICNTRL, ITEM, NGROUP, NELEM, IVALU)

Brief Description:

This subroutine sets the value of an element of a group of a control. It works only for single-NSU elements.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number
- NELEM: Element number
- IVALU: The value of the element
SETGRV(IAREA, ICNTRL, ITEM, NGROUP, IVALU)

Brief Description:
This subroutine sets the value of a group of a control. It works only for single-NSU groups.

INPUT ARGUMENTS:
- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number
- IVALU: The value of the element

GETGRV(IAREA, ICNTRL, ITEM, NGROUP, IVALU)

Brief Description:
This subroutine gets the value of a group of a control. It works only for single-NSU groups.

INPUT ARGUMENTS:
- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number

OUTPUT ARGUMENTS:
- IVALU: The value of the element
**NMCALL(N)**

**Brief Description:**

Returns the number of calls already made to current Operation. First call it will return 1, second call 2, etc.

**OUTPUT ARGUMENTS:**

- **N**: number of calls

---

**GETOPN(IOP)**

**Brief Description:**

Returns the number of the current Operation

**OUTPUT ARGUMENTS:**

- **IOP**: number of the current Operation

---

**GETSTG(ISTAGE)**

**Brief Description:**

Returns the execution stage: 1) translation stage (should never be encountered by a calling Operation), 2) control array editing and initialization, 3) initialization prior to accepting incoming data, 4) first call to the Operation, 5) subsequent calls to the Operation

**OUTPUT ARGUMENTS:**

- **ISTAGE**: execution stage
**SETOVR(NOVR)**

**Brief Description:**

Sets the number of cycles of Overlap maintained by the DRIVER for the calling Operation

**OUTPUT ARGUMENTS:**

* NOVR: number cycles Overlap

---

**GETOVR(NOVR)**

**Brief Description:**

Returns the number of cycles of Overlap set by the DRIVER for the calling Operation

**OUTPUT ARGUMENTS:**

* NOVR: number cycles Overlap

---

**GETGAP(GAP)**

**Brief Description:**

Returns the value of GAPVAL as given in the DRIVER

**OUTPUT ARGUMENTS:**

* GAP: the value of GAPVAL
GETSAV(LSAV,NSAV)

Brief Description:

Returns the location and length of the save space maintained by an Operation

OUTPUT ARGUMENTS:

- LSAV: location (index) of the save space
- NSAV: length of the save space

SETSAV(NSAV)

Brief Description:

Requests save space to be allocated in the DRIVER

INPUT ARGUMENTS:

- NSAV Number of NSU's of save space requested

GETSCR(LSCR,NSCR)

Brief Description:

Returns the location and length of the scratch space (used by all Operations)

OUTPUT ARGUMENTS:

- LSCR: location (index) of the scratch space
- NSCR: length of the scratch space
SETSCR(NSCR)

Brief Description:
Requests scratch space to be allocated in the DRIVER

INPUT ARGUMENTS:

* NSCR Number of NSU's of scratch space requested

JOBINF(INFO,N)

Brief Description:
Returns an array of job information including date and
time of project being run, data and time of execution,
and the name of the host machine running GENPRO.

OUTPUT ARGUMENTS:

* INFO: an array containing information in the fol-
lowing order: execution date, execution time, pro-
ject date, project time, machine id, number of NSU's
in job id block, and finally the information in the
job id block itself.
IVVALU(NVAR,NPT,VAL)

Variable Routines

Brief Description:

This subroutine returns the value of the Independent Variable corresponding to a particular data point of a Dependent Variable

INPUT ARGUMENTS:

- NVAR: Number of the Dependent Variable (position in the VECVAR list)
- NPT: Number of the point, from 1 to number of data values of the Dependent Variable passed in this call to the Operation

OUTPUT ARGUMENTS:

- VAL: Independent Variable value associated with the Dependent Variable value

GIVNTS(N)

Brief Description:

Returns the number of different Independent Variable intervals

OUTPUT ARGUMENTS:

- N: number of Independent Variable intervals

GIVNT(N)

Brief Description:

Returns the current Independent Variable interval being processed

OUTPUT ARGUMENTS:

- N: The current Independent Variable interval being processed
GIVLAB(N, ILAB)

Brief Description:

Returns the dimension and label of the Independent Variable as set in the Driver

OUTPUT ARGUMENTS:

- N: dimension of the Independent Variable
- ILAB: array containing the labels of the Independent Variable

GIVSCL(N, SCL)

Brief Description:

Returns the dimension and scaling factors of the Independent Variable as set in the Driver

OUTPUT ARGUMENTS:

- N: dimension of the Independent Variable
- SCL: array containing the scaling factors of the Independent Variable

GIVIBE(BEG, END)

Brief Description:

Returns the beginning and ending points of the Independent Variable interval (BEGIV and ENDIV as set in the DRIVER).

OUTPUT ARGUMENTS:

- BEG: Beginning of the Independent Variable interval
- END: End of the Independent Variable interval
GIVDEL(DEL)

Brief Description:

Returns the change per cycle in the Independent Variable (DELIV as set in the DRIVER).

OUTPUT ARGUMENTS:

- DEL: change per cycle in the Independent Variable
Error Reporting Routines

**INIMSG(INUM,ISUB,ATLOC,ISEV,LENV,IHEAD)**

**Brief Description:**

Initiate the common message reporter; this must be done prior to any call to the message reporter (see SETMSG and PRTMSG).

**INPUT ARGUMENTS:**

- **INUM**: message number
- **ISUB**: two element array containing the name of the calling subroutine
- **ATLOC**: floating point number associated with a section of the calling subroutine's code
- **ISEV**: severity of the message: 'C' catastrophic, 'W' warning, 'F' fatal,
- **LENV**: length of a value; typically 1 except if strings requiring more than one NSU are used, in which case this is set to the number of NSU'S required for longest value
- **IHEAD**: type of header/trailer: 0 - no header or trailer; 1 - header but no trailer; 2 - both header and trailer; 3 trailer but no header

**SEIMSG(IVAL,ITYP)**

**Brief Description:**

Sets a value in the error message initialized by previous call to INIMSG. Each call replaces the next $ encountered in the message template in the error messages file.

**INPUT ARGUMENTS:**

- **IVAL**: value
- **ITYP**: type of value: 'I' for integer, 'R' for real, 'N' for name, and 'S' for string
PRTMSG

Brief Description:

Prints current message initialize by previous call to IN-IMSG and with $'s encountered in the message template replaced by value through calls to SETMSG.
LESS FREQUENTLY USED SUBROUTINES

Control Array Routines

CTYPE(IAREA,ICNTRL,ICTYPE)

Brief Description:

This subroutine returns the type of a control in a single NSU variable as it is stored internally in the GENPRO work array.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:

- ICTYPE: Data type of a control, returned in a single NSU variable as a left justified 'R' for real, 'I' for integer 'N' for name, or 'S' for string.

CNAME(IAREA,ICNTRL,ICNAME)

Brief Description:

This subroutine returns the name of a control.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:

- ICNAME: Name of the control, in a single NSU variable, left justified.
COFF(IAREA, ICNTRL, ICOFF)

Brief Description:

This subroutine returns the offset for a control as it is stored in the work array. This indicates how many NSU's into a vector the control is stored at.

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number

OUTPUT ARGUMENTS:

- ICOFF: The offset of the control into a vector of control values.

NCONTR(IAREA, N)

Brief Description:

This subroutine returns the number of controls defined for an area of the Operation calling it.

INPUT ARGUMENTS:

- IAREA: Control area

OUTPUT ARGUMENTS:

- N: Number of controls in Area.
LCONTR(IAREA, N)

Brief Description:

This subroutine returns the total number of NSU's required to store a vector of all controls defined for a particular Area.

INPUT ARGUMENTS:

* IAREA: Control area

OUTPUT ARGUMENTS:

* N: Size of control Vector in NSU's.

AGADDR(IAREA, ICNTRL, ITEM, IADDR)

Brief Description:

This subroutine returns the first word address of a control aggregate

INPUT ARGUMENTS:

* IAREA: Control area
* ICNTRL: Control number
* ITEM: Item number

OUTPUT ARGUMENTS:

* IADDR: Location in GENPRO work array of the first word of the aggregate
GRADDR(IAREA, ICNTRL, ITEM, NGROUP, IADDR)

Brief Description:

This subroutine returns the first word address of a group of a control aggregate

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number
- NELEM: Element number

OUTPUT ARGUMENTS:

- IADDR: Location in GENPRO work array of the first word of the group

ELADDR(IAREA, ICNTRL, ITEM, NGROUP, NELEM, IADDR)

Brief Description:

This subroutine returns the first word address of an element of a control aggregate

INPUT ARGUMENTS:

- IAREA: Control area
- ICNTRL: Control number
- ITEM: Item number
- NGROUP: Group number
- NELEM: Element number

OUTPUT ARGUMENTS:

- IADDR: Location in GENPRO work array of the first word of the element
Dependent Variable Routines

**IOADDR(NVAR,LIN,LOUT)**

Brief Description:

Returns the locations in the GENPRO work array where data for a particular variable is picked up as input data to an Operation and where it is passed to the next Operation.

**INPUT ARGUMENTS:**

- **NVAR:** The number of the Dependent variable

**OUTPUT ARGUMENTS:**

- **LIN:** Location of where to get the data for the current operation
- **LOUT:** Location of where to put the data for the next operation

**IORATE(NVAR,INRATE,IRATE)**

Brief Description:

This subroutine returns the input and output rates of a Dependent Variable.

**INPUT ARGUMENTS:**

- **NVAR:** The number of the Dependent variable

**OUTPUT ARGUMENTS:**

- **INRATE:** Input rate
- **IRATE:** Output rate
GETCYC(IPAST,INOW,IFUT)

Brief Description:

Returns the number of Cycles of data available on the current call to the Operation broken down into the number of past, current, and future Cycles

OUTPUT ARGUMENTS:

- IPAST: Number of past cycles (of overlap)
- INOW: Number of current cycles (of data to be operated on)
- IFUT: Number of future cycles (of overlap)
<table>
<thead>
<tr>
<th>Driver Information</th>
<th>GBITCH(N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description:</td>
<td>Returns the number of bits per character for the internal character representation of the host machine.</td>
</tr>
<tr>
<td>OUTPUT ARGUMENTS:</td>
<td></td>
</tr>
<tr>
<td>* N: Number of bits per character</td>
<td></td>
</tr>
</tbody>
</table>

| GBITNS(N) |
| Brief Description: | Returns the number of bits per Numeric Storage Unit of the host machine |
| OUTPUT ARGUMENTS:  |   |
|   * N: Number of bits per character |

| GETVIV(VAL) |
| Brief Description: | Returns the value of the Independent Variable at the beginning of the data being passed to the current Operation |
| OUTPUT ARGUMENTS:  |   |
|   * VAL: value of the Independent Variable |

| GETCA(LCA,NCA) |
| Brief Description: | Returns the location and length of the control array for the current Operation within the GENPRO work array |
| OUTPUT ARGUMENTS:  |   |
|   * LCA: location (index) of the control array |
|   * NCA: length of the control array |
GETBUF(IOP, BUF, NBUF)

Brief Description:

Returns the location and length of a data buffer maintained by an Operation

INPUT ARGUMENTS:

• IOP Operation number of the Operation maintaining the data buffer.

OUTPUT ARGUMENTS:

• LBUF: location (index) of the data buffer
• NBUF: length of the data buffer

GETDAT(IOP, LDAT, NDAT)

Brief Description:

Returns the location and length of a the Dependent Variable data area written by an Operation

INPUT ARGUMENTS:

• IOP Operation number of the Operation writing the data area

OUTPUT ARGUMENTS:

• LDAT: location (index) of the data area
• NDAT: length of the data area