GCL

Programmer's Manual

DPS7000/XTA
NOVASCALE 7000

Job Control and IOF

REFERENCE
47 A2 36UJ 05

DPS7000/XTA NOVASCALE 7000 GCL

Programmer's Manual

Job Control and IOF

August 1999

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 36UJ 05

Trademarks and Acknowledgements

Preface

Scope and Objectives	This manual explains the GCL interface with GCOS 7 in interactive and batch modes.
Intended Readers	This publication is intended for all GCOS 7 users. It complements the information given in the *IOF Terminal User's Reference Manual* to which the reader must refer for detailed information on the set of system-level commands, directives and utilities. Other aspects for handling the system are treated in the *IOF Programmer's Manual*.
Prerequisites	It describes all the essential GCL functions. The primitives dealt with are in GPL and COBOL, and where applicable, the FORTRAN and C language equivalents are given.

Structure

Section 1	describes the general requirements of programming in GCL and the creation of new GCL commands.
Section 2	treats GCL in terms of basic commands, system variables and builtin functions.
Section 3	deals with command management and how to handle libraries, their members and the workspace.
Section 4	describes how to use GCL in accessing GCOS files and lists the commands available.
Section 5	deals with the GCL batch job and the parametrization of statements and input enclosures.
Section 6	treats aspects for TRACE and DEBUG facilities when handling GCL procedures.
Section 7	describes primitives and the use of help texts for assuring the Programmatic Interface for GCL translation.

Bibliography

GCOS 7 System Administrator's Manual (V7) *47 A2 41US*
GCOS 7 System Operator's Guide (V7).. *47 A2 47US*
GCOS 7 System Administrator's Guide (V8 and V9)............................... *47 A2 54US*
GCOS 7 System Operator's Guide (V8 and V9)...................................... *47 A2 53US*

IOF Terminal User's Reference Manual
Part 1... *47 A2 38UJ*
Part 2... *47 A2 39UJ*
Part 3... *47 A2 40UJ*

IOF Programmer's Manual... *47 A2 37UJ*

JCL Reference Manual..*47 A2 11UJ*
JCL User's Guide *47 A2 12UJ*

GPL Reference Manual *47 A2 35UL*
GPL User's Guide...*47 A2 36UL*
GPL System Primitives Reference Manual...*47 A2 34UL*

C Language User's Guide...*47 A2 60UL*
C Language Primitives Reference Manual...*47 A2 64UL*

COBOL 85 Reference Manual...*47 A2 05UL*
COBOL 85 User's Guide...*47 A2 06UL*

FORTRAN 77 Reference Manual *47 A2 15UL*
FORTRAN 77 User's Guide...*47 A2 06UL*
GCOS 7 File Recovery Facilities User's Guide *47 A2 37UF*

Text Editor User's Guide *47 A2 05UP*

FULL Screen Editor User's Guide *47 A2 06UP*

Library Maintenance Reference Manual...*47 A2 01UP*
Library Maintenance User's Guide...*47 A2 02UP*

Data Management Utilities User's Guide *47 A2 26UF*

Syntax Notation

The following notation is used in syntax formats:

ITEM	Capitals represent a keyword that is to be entered as-is.
item	Small italics represent a metalanguage term for which the user supplies a value.
	In the following, the non-italic `item` represents either a keyword or a metalanguage term:
[item]	An item within square brackets is optional.

```
{ item1 }
{ item2 } or { item1 | item2 | item3 }
{ item3 }
```

A set of items within braces:

- either in a column
- or in a line separated by |

means one item must be selected. The default, if any, heads the list and is underlined.

...	An ellipsis indicates that the item it follows may be repeated.

Table of Contents

7. Programmatic Interface

1. Introduction

GCL Programming involves using GCL commands and directives to be submitted to GCOS 7 for performing repetitive tasks. The startup sequence for user logon is such an example.

Once the program has been developed and tested, the user need not have to key in the program each time it is to be executed:

- by storing the program in a file
- and by expressing parameters as variables for substitution at the execution of the task with user-defined values which can be dynamically altered without modifying the stored GCL sequence.

1.1 Command Language

1.1.1 Purpose of Command Language

GCL is a set of commands that allow the user to request GCOS to perform specific activities in interactive and batch modes. Menus, prompts and helps facilitate the use of GCL.

Commands can be submitted to GCOS:

- either interactively at the terminal
- or stored in a file for multiple executions
- or compiled into procedures that create new commands of the language.

1.1.2 Objects

Objects are the basic elements on which the commands operate. Files, libraries, outputs, catalogs, and programs are objects, just as the user profile is.

Examples of activities that include functions and procedures to be executed on objects are:

- to create the environment that controls resources and their allocations to applications
- to manage files being copy, save, restore, compare, create, delete and load
- to define new commands
- to test programs, and to observe and change system values in the process
- and to ensure the security of objects against unauthorized use.

1.1.3 Variables and Expressions

Wherever a value can be specified for a parameter, either in keyword or positional notation, this value can be specified as being the value of a variable or the result of evaluating an expression. Variables may be user-defined or system-defined. Expressions are combinations of builtin functions whose arguments may be variables or other builtin functions.

User variables are denoted by a name preceded by a percent (%) sign (for example, %V, %PRICE, %NEW-RATIO).

The names of system variables begin with a number sign character (for example, #PL, #NOVICE).

Builtin functions also have names which begin with a number sign and their arguments, if any, are enclosed in parentheses (for example, #PLUS(6, %V) or #DIVIDE(%QUANTITY, %RATIO)).

A command can be coded:

```
   DLLIB LIB=%LB BYPASS=0 FORCE=0 SILENT=%S
or DLLIB %LB,0,0,%S
```

When a parameter is a list of like values, it may also be replaced by a variable or an expression, provided that the value of the variable or the result of the expression is a list of suitable type and length. Elements in such lists may also be expressions or variables. A builtin function is also provided for building up a list from elements that may be variables, expressions, or constants. In lists of unlike values, individual elements may be substituted by variables or expressions. For example, one could write:

```
CPF A B DYNALC=TEMPRY ALLOCATE=(SIZE=%S,12,%U,1,1)
```

But the list as a whole cannot be substituted by a variable or an expression; expressions always produce lists of like values.

1.2 Global Variables

Parameters of commands and directives can be referred to either as literal values or by symbolic names. The statement:

```
PRINT_FILE MYDIR.TEST.REL.FILE2
```

only prints a specific file and performs no other operation.

If a series of operations is to be performed on the same file, its name would have to be reentered each time.

GCL allows naming parameters by their symbolic names. The parameter can then be referred to later on by its symbolic name which functions as a variable. A variable that applies for the duration of the session is ***global*** variable. This Section discusses how to use variables in GCL programs to facilitate command entry and the resulting repetitive processing.

There are certain limits concerning the number of global variables. For each job or for each IOF session, the user has available:

- 170 optimized variables present in memory during each jobstep, namely:
  - 40 variables with a length of 1 byte
  - 40 variables with a maximum length of 8 bytes
  - 40 variables with a maximum length of 16 bytes
  - 30 variables with a maximum length of 32 bytes
  - 20 variables with a maximum length of 80 bytes.
- 97 variables with no limitation on length, allocated in a subfile of the user's SYS.POOL.

A maximum of 100 global variables can be declared in the same procedure.

Continuing on from the above example, the following two commands would make the variable known and assign a value to it:

```
GLOBAL F2 FILE
LET F2 MYDIR.TEST.REL.FILE2
```

Any further reference to that value can be in the form of %F2, for example:

```
PRINT_FILE %F2
DELETE_FILE %F2
COMPARE_FILE %F2 MYDIR.TEST.REL.FILE1
...
```

The percentage sign (%) introduces the reference to the value of the variable and distinguishes it from the file name F2.

1.2.1 Declaring Global Variables

A global variable must be declared before being used. The GLOBAL directive specifies the mandatory name of the variable and optional values that the variable can take. A complete list of directives is treated in the *IOF Terminal User's Reference Manual*.

Syntax:

```
{ GLOBAL }
{        }
{ GB     }

  NAME=name31

  [      { CHAR                                 }]
  [ TYPE={ BOOL | DEC | FILE | FSET | HEXA | LIB }]
  [      { NAME | OUTPUT | RFILE | STAR | VOLUME }]

  [ LENGTH=dec3 ]

  [ NUMVAL=( dec2 [ dec2 ])]

  [ VALUES=( condition [ condition ]...)]
  [ PROMPT=char40 ]
```

Parameters:

NAME	mandatory: up to 31 alphanumeric characters beginning with a letter, defining the name of the variable.
	To redefine a variable with different attributes, the previous definition must be first deleted by `DELETE_GLOBAL` (`DLGB`).
TYPE	the type of the variable. See Paragraph *"Types"*.
	Default: CHAR (plain character string)

LENGTH — the maximum length for each element of the variable.

Default and maximum applicable values depend on the type of the variable being defined.

For example:

`GLOBAL AGE DEC 3` means a value of up to three decimals.

Type	Maximum Length	Default Length
CHAR	255	80
BOOL	1	1
DEC	31	31
FILE	255	44
FSET	255	80
HEXA	8	8
LIB	255	44
NAME	44	31
OUTPUT	255	80
RFILE	255	80
STAR	88	31
VOLUME	255	80

NUMVAL — a pair of numbers denoting minimum and maximum number of elements in list of variables, `max` => `min` and `dec2` <= 64.

When only one number is provided, both the maximum and minimum are set to that value.

Default: (min=0,max=1)

PROMPT — prompt for user in menu mode to enter a value for the associated variable and is displayed by the command:

```
LET variable-name #
```

where value read from the terminal is assigned to the variable.

Example:

```
GLOBAL X DEC 3 PROMPT='Your age?'
LET X #
Your age?
```

VALUES

list of up to 32 conditions that values assigned to the variable must meet. At least one condition must be satisfied: an OR is performed on the conditions and the value assigned. Conditions must conform to the type and length specified for the variable. Different types of conditions may appear in the same list.

Example 1:

`VALUES=(&9 * $/)` means applies to a variable which starts with a digit, or is an asterisk or contains a slash.

Example 2:

`GLOBAL AGE DEC 3 VALUES=0<=*<=180`
means that AGE takes values in the inclusive range of 0 through 180.
Default: All values conforming to the variable type and length.

Type of Condition	Form	Examples
discrete value	value	A 123 *
relational, see Note	{> }value {< }value {>=}value {<=}value {= }value	>2 <32 >=0 <=XYZ =0

range, see Note	{> } {> }
	value{ }*{ }value
	{>=} {>=}
		A<=*<H
		0<*<=99
		A<=*<=Z

contains	$value	$XYZ $. $',' *contains a comma*
starts with	/value	/AXY /' ' *starts with space* /.
starts with digit	&9	&9
starts with letter	&A	&A
alphanumeric start	&X	&X
not	^any of above	^=* ^$XYZ ^&9

NOTE:
Non-numeric values are compared by their EBCDIC collating sequences.

EXAMPLES:

`GLOBAL COUNT DEC 3 VALUES=>0`	decimal, up to 3 digits, values must be positive
`GLOBAL LIST`	
`NAME 20`	each name is 20 characters long
`NUMBAL=(1,30)`	list of up to 30 names
`VALUES=&A`	beginning with a letter
`GLOBAL REPLY NAME 3`	
`VALUES=(YES NO)`	variable to take only the values YES and NO
`GB C`	defaults to string of 0 to 80 characters
`GLOBAL B BOOL`	boolean variable
`GB D DEC`	
`VALUES=(0 10<=*<=99)`	variable which is 0 or a decimal in range 10-99

❑

Constraints:

- When the GLOBAL directive is used in immediate mode (non-compiled), the values assigned to the parameters of the directive cannot be expressions. They must be literal values.
- When the GLOBAL directive is used in a compiled sequence (GCL procedure, sequence executed by EXECUTE_GCL, sequence executed by GCL in batch), the parameters of the directive can be expressions including variable names *provided that the GLOBAL directive is executed through the SCALL command and the contents of the variable is referenced by #VALUE.*
- When execution of a GCL procedure that contains:
  - keywords declared with variables or expressions as default values and
  - global variables declaration,
- is aborted by the user using "/" character in a prompt screen or by GCL kernel, the global variables declared in the aborted procedure remain active. This way lead to the error message "INCONSISTENT GLOBAL REDEFINITION FOR VARIABLE MYVAR" in case of redefinition with another characteristics.

- In this case the global variable must be deleted by the user using the command DLGB MYVAR.

EXAMPLE:

```
LOCAL N NAME;
LET N #CVNAME(#CAT(#USERID,'_TERMID'));
SCALL GLOBAL NAME=%N,TYPE=NAME;
LET %N #CVNAME(#TERMID);
LET # #VALUE(%N);
...
```

❑

1.2.2 Assigning Values to Variables

Once a variable has been declared, it can be assigned a value. Any attempt to refer to a variable that has not been assigned a value results in the diagnostic `VARIABLE` *n* `NOT ASSIGNED`. The LET directive assigns a value to a variable.

```
LET AGE 62
```

assigns the value 62 to the variable AGE. Since LET is a directive, it can be used to change the value of a variable at any time and the change becomes effective immediately. The next reference to the variable will be a reference to the new value.

There is no percentage sign before the name of the variable.

The construct:

```
LET %VAR 3
```

assigns 3 to the variable whose name substitutes for VAR:

```
LET VAR AGE
LET %VAR 3
```

results in AGE being assigned the value 3 and is equivalent to:

```
LET AGE 3
```

The assigned value must, therefore, meet the requirements of the GLOBAL directive that declared the variable:

- if the type is incorrect, a `TYPE ERROR` diagnostic is returned
- if the value is longer than that specified by LENGTH, a `LENGTH ERROR` is returned
- if the value does not satisfy the VALUES conditions, a `VALUE ERROR` is returned.

EXAMPLE:

Using the declarative for AGE:

```
LET AGE 1234        results in LENGTH ERROR
LET AGE ABC         results in TYPE ERROR
LET AGE -3          results in VALUE ERROR
```

When the declared variable is a list, the values to be assigned are enclosed within parentheses.

If `LIST` is declared, for example:

```
GLOBAL LIST DEC 2 NUMVAL=(3,6) VALUES=>0
```

a list of 3 through 6 decimal values, each one up to two digits and positive, the directive:

```
LET LIST (1,3,5,7)
```
or
```
LET LIST (1 3 5 7)
```

assigns `LIST` a set of four values.

If an attempt is made to assign fewer values than the minimum or more than the maximum, `INDEX ERROR` is returned. For example:

```
LET LIST 6                  --> INDEX ERROR  (only 1 value)
LET LIST (6 9)              --> INDEX ERROR  (2 values)
LET LIST (2 4 6 8 10 12 14)--> INDEX ERROR  (7 values)
```

If any one of the elements in the list fails to meet one of the GLOBAL requirements (LENGTH, TYPE, VALUES) the appropriate diagnostic is returned.

❑

1.2.3 Types

Since GCL is a typed language, the assignments and references of variables and expressions must be consistent with their type. For example, if AGE is decimal, assigning the name of a file to it will result in an error diagnostic. The twelve types in GCL are:

`BOOL`	Boolean value 0 or 1
`CHAR`	Character string, quoted or unquoted
`DEC`	Decimal value, signed or unsigned
`FILE`	File for example, A.B.FILE or A.B.MYLIB..SF
`FSET`	Fileset for example, A.*.B
`HEXA`	Hexadecimal value for example, A23F
`LIB`	Library for example, A.B.MYLIB
`NAME`	Name for example, JOE, A-B, MYPG
`OUTPUT`	Output for example, X123:2:1
`RFILE`	Remote File for example, $LYON:A.B.C
`STAR`	Star-Name for example, A*B
`VOLUME`	Volume for example, VOL2:MS/D500

A literal value assigned to a variable must be valid for the type of variable.

One variable can be assigned the value of another variable by the following construct:

```
LET V1 %V2
```

where the variable V1 is assigned the current value of the variable V2. *In this case, the variables V1 and V2 must be of the **same** type.* A limited number of implicit conversions are supported by GCL. These are summarized in the following diagram:

```
---^---------^---------^-------CHAR--------^---------^---------^--
   |         |         |         |         |         |         |
   |         |         |         |         |         |         |
 OUTPUT    VOLUME     DEC      RFILE      FSET      STAR      HEXA
                       ^         |         |         ^
                       |         |         |         |
                       |         +----^----+         |
                      BOOL            |             NAME
                                      |
                                     FILE
                                      ^
                                      |
                                      |
                                     LIB
```

In the diagram, a CHAR variable may be assigned the value of a variable of another type. A DEC value may be assigned the value of a DEC or BOOL variable. But a BOOL variable cannot be assigned a DEC value, nor can OUTPUT be assigned a VOLUME value.

Explicit conversions assign the value of one variable to another. These conversions are provided through a set of dedicated builtin functions, for example:

```
LET AGE #CVDEC(%C)
```

If C is a variable, say CHAR, whose current value is a valid for a DEC value, then assign it to AGE. Otherwise return a `TYPE ERROR`. A set of dedicated builtin functions ensures whether a given variable has a value that is acceptable to a variable of another type.

```
LET B #ISITDEC(%C)
```

will assign the boolean variable B the value 1 if the value of C is a valid for a DEC value. Otherwise, it will assign the boolean value 0.

1.2.4 References to Variables

In all commands and directives, a value that can be specified for a parameter can be replaced by a reference to a variable as in the examples above. The actual process used involves substituting the current value of the variable for its reference wherever it appears.

References to variables are introduced by a percentage sign (%) to distinguish them from literal name values. The value denoted by the variable must be acceptable in the context where it is used, otherwise an error diagnostic is reported. For example:

```
CLEAR_LIBRARY %B
```

where B is a boolean value cannot be accepted since a boolean value is not treated as a library.

Using lists is illustrated in the following example.

The COBOL command accepts as its first parameter a list of names or star-names denoting the names of source COBOL programs to be compiled. The line coding:

```
COBOL (A B C)
```

requests the compilation of three source COBOL programs. Assuming that three following global variables have been declared:

```
GLOBAL G1 NAME
GLOBAL G2 NAME
GLOBAL L  NAME NUMVAL=(0,4)
```

resulting in two scalar name variables G1 and G2, and L a list of up to four names. Assuming that the following assignments have taken place:

```
LET G1 A
LET G2 B
LET L  (C D E)
```

then:

COBOL %G1	stands for COBOL A
COBOL %G2	stands for COBOL B
COBOL %L	stands for COBOL (C D E)
COBOL (%G1 %G2)	stands for COBOL (A B)
COBOL (%L %G1)	stands for COBOL (C D E A)
COBOL (%G2 X Y %L)	stands for COBOL (B X Y C D E)

The example shows how a variable can:

- stand for a list
- be an element of a list
- or be combined with other variables and with literal values to build up a list.

Local and global variables referenced in a procedure must be declared either in the procedure or before the procedure is activated. A sequence of GCL statements executed through the GCL command EXECUTE_GCL is similar to a called procedure. The variables declared in the sequence are not known to the procedure that executes the EXECUTE_GCL.

The following sequence is correct:

```
PROCEDURE A;                              PROCEDURE B;
GLOBAL G.....;                            Local V...;
CALL B;                                   LET V %G;
                                          ENDPROC;
```

The following sequence is wrong:

```
PROCEDURE A;                              PROCEDURE B;
LOCAL V...;                               GLOBAL G.....;
CALL B;                                   ENDPROC;
LET V G%
```

1.3 Expressions

Variables can be operated on and combined into expressions to build new values. In GCL, expressions are created from builtin functions. A builtin function is denoted by #, followed by a name and an optional list of arguments enclosed within parentheses, e.g.:

```
LET AGE #PLUS(%AGE,1)
```

assigns to the variable AGE the value of the variable AGE plus 1 to increments the value of AGE. In this example, #PLUS is a builtin function with two arguments.

Builtin functions can have any number of arguments. Following are some examples:

#TIME and #DATE	are builtins with no argument give today's time and date.
#LENGTH(a)	is a builtin with one argument that gives the length of its argument expressed as a number of characters.
#CVDEC(a), #ISITDEC(a)	are also builtins with one argument.
#PLUS(a,b), #MINUS(a,b)	are arithmetic builtins with two arguments.

Some builtins may have a varying number of arguments. For example, #PLUS may have two or more arguments:

```
LET A #PLUS (%A,%B,%C,%D)
```

adds up A, B, C, and D into A.

Similarly, the #CAT builtin returns a result which is the concatenation (that is, combining together in a single string) of its arguments:

```
#CAT (ABC, XYZ, QP)
```

yields string ABCXYZQP.

Builtins are numerous; they are fully discussed in Section 2. There is a builtin for most types of operations to perform on variables.

Arguments of builtins may be literal values, variable references, system variables, or other builtins. Builtin functions can be nested as deeply as desired. For example:

```
#CAT(X,#PLUS(1,#TIMES(4,#PLUS(2,3))))
```

results in the string of characters X21.

Arguments of the builtins must be of a certain type:

- #PLUS requires DEC arguments
- #CAT accepts CHAR arguments.

Similarly the result of a builtin also has a type:

- the result of #PLUS, #MINUS or #CVDEC is DEC
- the result of #ISITDEC is BOOL
- and the result of #CAT is CHAR.

Implicit and explicit conversion rules are the same as for the assignment of variables.

Wherever a variable may be used, an expression in the form of a builtin function can also be used. This also applies to lists, where a builtin may be used to denote a list or a single element within a list:

```
LET LIST (6 3 #PLUS(4,3) #MINUS(6,2))
```

has the same effect as:

```
LET LIST (6 3 7 4)
```

1.4 Reading and Writing Values

The construct to ascertain the value of an expression is:

```
LET # expression
```

It results in printing the expression at the terminal.

```
LET # %LIST
```

displays the value of the variable LIST. The expression to the right of # can be as complex as necessary.

The # alone is a particular instance of a system variable denoting the current line on the output device being either the IOF terminal, or SYSOUT or PRTFILE in batch. Assigning a value to it results in printing that value on the output device.

For IOF:

When used in an expression like:

```
#PLUS (#,1)
```

1 is added to a value which is to be entered at the terminal. Enter the value when the prompt #? appears. The value keyed in will replace # in the expression, and evaluation continues. In this particular case, the value entered must be numeric to be acceptable to the #PLUS builtin.

The most use of # in an expression is in constructs like:

```
LET V #
```

to assign to a variable a value read in from the terminal, in which case:

- the prompt to enter the value will be in the form V?
- the value supplied will be assigned to variable V
- however, if variable V was declared with PROMPT, this *prompt* will appear to request the user to supply a value.

The following sequence:

```
GLOBAL AGE DEC 3 PROMPT='What is your age?'
LET  AGE #
What is your age? 22
LET  # %AGE
22
```

is an easy way to build up any type of dialog required. It is particularly useful when using stored sequences of GCL.

For Batch:

`'LET V #'` cannot be used.

To assign to a variable, a value read in from the terminal of the job submitter, use `'READ_FROM_CONSOLE'`.

1.5 Stored Sequences of GCL

The commands of a task performed repeatedly can be stored in a library member. The system can then fetch the commands from that member at the user's request rather than for the user to key in the task sequence at the terminal.

The following sequence of GCL commands will compile, link and execute a COBOL program, then print the contents of its output file:

```
COBOL MYPG
LINK  MYPG
EXEC_PG MYPG FILE1=IFN1 ASG1=.MYFILE
PRINT_FILE .MYFILE
```

This task can be repeatedly performed for as many times as there are different programs and files.

The following sequence of GCL commands stored in a source library:

```
COBOL %P
LINK  %P
EXEC_PG %P FILE1=IFN1 ASG1=%F
PRINT_FILE %F
```

has elements which are to be replaced by references to variables or parameters. Here the library member storing the sequence is called TEST.

The commands which allow executing such a sequence are:

- ALTER_INPUT (AI) in IOF
- EXECUTE_GCL (EXGCL) in batch.

See *IOF Terminal User's Reference Manual* for the description and syntax of these commands.

1.5.1 ALTER_INPUT Command

The ALTER_INPUT command allows replacing entries normally keyed in at the terminal by entries in a file. The entry stream that can be executed by ALTER_INPUT can contain:

- commands belonging to levels S: and C:
- and data to be processed by a processor at level I:, for example.

The entry stream can be parameterized using Global Variables which are referenced in the commands to be executed *but not in the data*.

In the preceding example:

- two Global Variables must be declared:
  ```
  GLOBAL P NAME;
  GLOBAL F FILE;
  ```
- their desired values allocated:
  ```
  LET P MYPG;
  LET F .MYFILE;
  ```
- and the sequence executed by:
  `ALTER_INPUT TEST` *or* AI TEST;

Basic GCL Commands cannot be executed in a sequence activated by ALTER_INPUT.

1.5.2 EXECUTE_GCL Command

- When in the current domain, the EXECUTE_GCL command allows executing:
  – commands of the current domain
  – all the directives
  – and all basic GCL commands.
- At level S:, the EXECUTE_GCL command allows executing:
  – all the commands in the IOF domain
  – all the directives in the H_NOCTX domain
  – and all basic GCL commands.
- At level C: of MNLIB SL, the EXECUTE_GCL command allows executing:
  – all the commands in the LIBMAINT_SL domain
  – all the directives in the H_NOCTX domain
  – and all basic GCL commands.

The file to be executed by EXECUTE_GCL can only contain commands. Commands can be parameterized as for ALTER_INPUT by using Global Variables.

The commands in the file TEST can be executed by:

```
EXGCL TEST;
```

However, the VALUES parameter of EXECUTE_GCL can also be used to pass values expected by the sequence to be executed. In this case, these expected parameters must be described in the GCL basic command KWD (values may be determined by default).

The file TEST previously described:

- the equivalent of the sequence would be:
  ```
  KWD P NAME;
  KWD F FILE;
  COBOL %P;
  LINK  %P;
  EXEC_PG %P FILE1=IFN1 ASG1=%F;
  PRINT_FILE %F;
  ```
- and the sequence executed by:
  ```
  EXGCL TEST, LIB=MYLIB, VALUES=(P=MYPG,F=.MYFILE);
  ```

1.6 STARTUP Sequences

IOF offers a facility whereby a stored sequence of commands or lines can be automatically executed when the user logs on. This sequence is known as a startup sequence. Startup sequences can be defined at system, project, and user levels. Their execution may be mandatory or optional.

A typical use of a startup sequence would be to place the user under the control of a certain processor without having to request it.

An optional startup sequence can be bypassed. (Mandatory sequences may not be bypassed except by the project SYSADMIN.)

1.6.1 IOF Startups

Startup sequences are described in *IOF Terminal User's Reference Manual*.

When the user logs on to IOF, the system simulates an AI directive to the startup sequence of the user or the project associated with the user. In the startup sequence all the commands required can be stored to set the user's normal operating context.

Following is an annotated example of what a project level startup might look like:

```
LET # 'BEGINNING OF PROJECT XYZ STARTUP';  issue a message
MDP NOVICE=0 GCLFORM=LINE;                 modify elements of
                                           profile

MWINLIB SL .REF                            define working libraries
MWLIB SL <WORK
MWINLIB BIN .CMDS
MWLIB BIN X$TEMPRY

$$AI >JONES IF=#EQ(JONES,#USERID)          test if user JONES
$$AI >HENRY IF=#EQ(HENRY,#USERID)          test if user HENRY
$$AI NORMAL-STUFF                          for other users execute
$$AI SPECIAL-STUFF IF=some-condition       additional sequences and
$$AI >END                                  leave

$JONES:                                    Jones is a PASCAL fan
MWLIB BIN .JONES.SPECIAL                   who wants automatically
PASCAL                                     to enter the PASCtAL
$$AI >END                                  compiler at logon
```

```
$HENRY:                          Henry wants no mail
MAIL OFF                         additional and has
MWLIB CU .CU.REF                 working libraries
MWINLIB CU .CU.WORK
MWLIB LM .LMS

$END:
```

Another kind of project-level startup could have the following structure:

```
LET # 'BEGIN STARTUP';          issue a message
MWLIB SL .WORK
MWLIB                           define the working libraries
   .                            and other common rules
   .
MWINLIB

AI #CVNAME(#CAT(#USERID,-STP))  then ask to execute a user-dependent
                                sequence built up from the user's
                                name followed by -STP

LET # 'END STARTUP';            issue a message
```

Each user is free to put in a personalized sequence, for example:

- JONES-STP will contain the GCL to enter PASCAL
- HENRY-STP will contain additional library definitions and the MAIL OFF command.

Startups are solely at the discretion of the system administrator.

1.6.2 Batch Startups

For GCL batch jobs, specific startups are executed before the execution of the submitted GCL sequence but after its compilation. As in IOF, 2 startup sequences can be attached to a Project in the catalog, namely, a mandatory startup and an optional one.

Possible startups are:
a) SITE_GCL_B
b) project_GCL_B
c) project_user_GCL_B.

In a batch startup:

- the following can be used:
  - commands of the IOF domain
  - directives
  - and operator commands through EXDIR.
- however, the following may not be used:
  - GCL basic commands, *however, the EXECUTE_GCL command is authorized in a startup and allows executing sequences containing GCL basic commands*
  - statements processed by the Input Reader
  - input enclosures
  - and parameterization through VALUES.

1.7 Creating New GCL Commands

All GCL commands so far are standard system-supplied commands. However, IOF provides a means whereby each user can define personalized commands or commands to be used by other users. This facility is described in Section 3.

Creating a new command involves:

- writing a program that defines the command and its function; the program is a GCL procedure which is a particular construct of GCL commands, some of which are dedicated to this purpose
- compiling the GCL procedure through the MAINTAIN_COMMAND (Command Management) processor which also provides some facilities for handling compiled GCL procedures in libraries
- then storing the compiled GCL procedure in a BINary library; if this library is one of the user's input binary libraries defined by the MWINLIB BIN command, the command is immediately accessible to that user.

1.7.1 Domains, Libraries and Search Rules

The new command can be used:

- at system level
- within a given processor
- or as a directive.

The scope of the command is its domain, for example:

- a command that compiles, links and executes a program is only accessible at system level and belongs to the IOF domain
- a command that edits and renumbers a source program is only accessible from within MAINTAIN_LIBRARY SL and belongs to the LIBMAINT_SL domain
- a command used as a directive is accessible in all contexts and belongs to the H_NOCTX domain.

The *System Administrator's Manual* gives the list of standard domains. Section 3 gives the list of all existing domains.

A user-defined command belonging to a particular domain may make use of all commands and directives of that domain but not those of another domain since the context is no longer valid. For example, a command used at system level may be expressed in terms of other system-level commands and directives but may not refer to MAINTAIN_LIBRARY SL or other processors' commands.

Storing the command determines who is the user of the command:

- to be the sole user of that command,
  - store the definition in a binary library
  - and ensure that the library is named in a MWINLIB BIN command for it to be in the input search path
- to allow the command to be available to a set of users such as a project, store the definition in a library that is in the search path of all its potential users by including a suitable MWINLIB BIN command in the startup sequence common to all these users
- if the command is to be used by all the users of an installation, store the definition
  - either in the SYS.HBINLIB library to which only the System Administrator has write access
  - or in any other library that is in all users' search paths, possibly by including a reference to it in a MWINLIB BIN command in the site-level startup sequence.

The number of binary libraries allowed in the GCL search path as well as SYS.HBINLIB depends on the GCL15BIN CONFIG parameter declared at GCOS 7 system configuration. At execution, GCL can process a maximum of 4,090 command names or aliases, and a maximum of 3,276 prompts.

When a command is keyed in at the terminal:

- the system looks for its definition in the user's binary library input search path, if any, in the order in which the search path is defined in the MWINLIB BIN command, ***first INLIB1, then INLIB2 and lastly INLIB3 or INLIB15*** until it finds a definition of that name
- if the search fails, the definition is taken from the system library SYS.HBINLIB containing all system-supplied command definitions.

If the same command name exists in several libraries of the user's search path, only the one in the first library referred to as the *most local* one will be accessible. Altering the search path enables modifying the set of commands to access. The following example illustrates this:

The search path is defined by `MWINLIB BIN (.L1 .L2 .L3)`

- `.L1` contains commands `C1`, `C2`, `C3`
- `.L2` contains commands `C2`, `C4`
- `.L3` contains commands `C3`, `C5`

then:

- `C1` refers to the definition in `.L1`
- `C2` refers to the definition in `.L1`
- `C3` refers to the definition in `.L1`
- `C4` refers to the definition in `.L2`
- `C5` refers to the definition in `.L3`
- `C6` refers to the definition in `SYS.HBINLIB`

Modifying the search path as below:

```
MWINLIB BIN (.L3 .L2 .L1)
```

results in:

- `C1` referring to the definition in `.L1`
- `C2` referring to the definition in `.L2`
- `C3` referring to the definition in `.L3`
- `C4` referring to the definition in `.L2`
- `C5` referring to the definition in `.L3`
- `C6` referring to the definition in `SYS.HBINLIB.`

1.7.2 MAINTAIN_COMMAND

MAINTAIN_COMMAND (MNCMD) is the compiler for command definitions. It is described in Section 3.

MAINTAIN_COMMAND enables working on a copy of a command definition held in a workspace and not operating on it directly. Handling the workspace involves:

- storing the workspace contents in the binary library through the SAVE or RESAVE command
- loading the command definition from the library into the workspace through the LOAD command.

Command definition proceeds as follows:

- creating the command in the workspace through the CREATE command
- adding lines in the workspace through the APPEND command
- and editing the contents of the workspace through the LEDIT command.

MAINTAIN_COMMAND is an incremental compiler which checks each line of a command definition and compiles it as it is entered. However, complete command definitions stored in a source library can be submitted for compilation through the COMPILE command.

Menus and prompts are available on request:

- to assist in keying in the definition
- or when an error is detected in an entry.

Other MAINTAIN_COMMAND commands:

- define the DOMAIN and the library (BINLIB) to which the command definition belongs
- LIST the names of the commands of a particular domain
- and PRINT the definition of a particular command.

Up to 2045 command names or aliases may be compiled in the same binary library (BINLIB) for the same domain.

1.7.3 GCL Procedures

A command is defined by a GCL procedure consisting of a sequence of GCL commands embedded between a PROC and an ENDPROC command.

The commands that can appear within a procedure are the following:

- dedicated GCL procedure commands dealt with in Section 2
- directives treated in the IOF Terminal User's Reference Manual
- any system-supplied or user-defined command that belongs to the domain of the command being defined.

PROC Command

The PROC command:

- heads the definition
- provides the name of the command
- its aliases
- and the short explanatory text to be displayed with the name of the command in the menu.

EXAMPLE:

```
PROC (SORT_NAMES SRTN)
     PROMPT='to sort out the list of user names';
```

A GCL procedure created with the same name as a system GCL command such as CBL, will override the system command which will no longer be available.

❑

KWD Command

The KWD command:

- introduces each parameter of the command
- provides
  - the name of the parameter with its possible abbreviations
  - type
  - length
  - and shape
  - the condition that it should satisfy
  - its default value, if any
  - and a short explanatory text for display alongside the screen or serial prompt.

EXAMPLE:

```
KWD ORDER NAME 4
    VALUES=(ASC DESC) DEFAULT=ASC
    PROMPT='ASCending or DESCending order?';
```

defines a parameter named ORDER that can take values ASC or DESC, the first being its default value.

❑

LOCAL Command

The LOCAL command defines variables that are local to the procedure.

GLOBAL Command

The GLOBAL command defines variables that pertain to the entire interactive session. A maximum of 100 variables may be declared or referenced in the same GCL procedure.

CONTROL Command

The CONTROL command specifies:

- further controls that must be satisfied for the parameters to be accepted
- the conditions under which check is made
- and the message to be issued if it fails.

EXAMPLE:

```
CONTROL WHEN=#EXIST(K1)
        CHECK=#NOT(#EXIST(K2))
        MSSG='K1 and K2 are mutually exclusive';
```

will check that two parameters `K1` and `K2` may not be simultaneously specified.

❑

Other Commands

Structuring commands may be used to direct the flow of control within the procedure such as:

- IF
- ELSE
- ENDIF
- WHILE
- UNTIL
- GOTO.

Calling commands such as SCALL and VCALL can be activated in the procedure.

ENDPROC must be the last command in the definition and serves as a RETURN command, if none is explicitly specified.

Limits on GCL Procedures

The limits concerning a procedure are:

- The maximum length of a source statement is 1536 bytes
- the maximum number of statements allowed for one procedure is 4680
- the maximum number of KWD statements is 255
- the maximum size for a KWD parameter is one screen
- a maximum of 100 global variables can be declared in the same procedure
- the maximum size of the binary code generated by MAINTAIN_COMMAND for a procedure is 64K, although no element of the procedure can exceed 32K.
  - An example of an element is a group of executable statements or the zones reserved for KWD and LOCAL.
- Due to the optimization done by the GCL compiler, it is not possible to compute the size of the elements of a procedure. If, during the compilation phase, an error message beginning with "TOO MANY" appears, you must split the procedure into several smaller ones.

1.7.4 Example 1: Creating a Directive

On asynchronous links, when the terminal has not been active for some time, the connection is automatically interrupted after timeout defined at system installation. To define a directive that will prevent the terminal from being disconnected when inactive, use the #KLN builtin function within a procedure. See Section 2.

The #KLN builtin has two arguments:

- *n* being the number of seconds that *string* appears until a break is issued
- and *string* being the message.

Its result is always the boolean 1. The following procedure will create a new directive that will solve the problem of keeping the connection alive even when the terminal is inactive.

```
PROC (KEEP_LINE,KLN)
     PROMPT='keep the line active';
LOCAL B BOOL;
LET B #KLN(60,'I am busy');
ENDPROC;
```

Once compiled and stored in the H_NOCTX domain of a suitable library, the KLN directive becomes available for use at system level:

- within any processor
- or as a directive by prefixing it with the directive identifier:

  ```
  $$KLN
  ```

An optional parameter allows further control over the frequency at which the message is issued.

```
PROC (KEEP_LINE KLN)
     PROMPT='keep the line active';
KWD  (FREQUENCY FREQ)
     DEC 3 VALUES=>0
     PROMPT='frequency, in seconds';
LOCAL B BOOL;
IF #NOT(#EXIST(FREQ));
    LET FREQ 60;
ENDIF;
LET B #KLN(%FREQ,'I am busy');
ENDPROC;
```

In the above example:

- the KWD command introduces the characteristics of the parameter `DEC 3` that is 3 positive decimals
- the three commands starting with the IF command assign a default value to the parameter when it is not supplied
- the LET command refers to the value of the parameter as an argument of the builtin #KLN.
- the new KLN directive may then be used as:

```
    $$KLN            meaning 60 seconds
    $$KLN 30
or  $$KLN FREQ=30
```

Introducing the following command allows further defining a second parameter to change the busy message:

```
KWD (MESSAGE MSSG)
    NUMVAL=(1,1)
    DEFAULT='I am busy'
    PROMPT='Message to be issued';
```

and changing the last but one command to:

```
LET B #KLN(%FREQ,%MSSG);
```

1.7.5 Example 2: Creating a new IOF Command

The next example illustrates how a new command can be created to compile and link a named COBOL source program. This command belongs to the IOF domain and must be therefore stored in a suitable binary library in that domain.

```
PROC (COMPILE_LINK CLC)
     PROMPT='Compile and link';
KWD  PROG NAME 32 NUMVAL=(1,1)
     PROMPT='name of the program';
COBOL %PROG;
LINK %PROG;
ENDPROC;
```

`NUMVAL=(1,1)` in the KWD command indicates that the parameter is mandatory.

The above command can be activated as:

```
   CLC PROG=X
or CLC X
```

Linking will proceed even if the compilation has failed. This can be avoided by entering the following three lines after the COBOL command:

```
IF #GE(#SEV,3);
   ABORT 'Compilation fails';
ENDIF;
```

which will abort the procedure with a suitable message, if the severity level of the compilation is 3 or more.

The above command covers the case when only one program is to be linked in the load module. The next example shows several source programs to be linked in a single load module, the first program named being the main program:

```
PROC (COMPILE_LINK CLC)
     PROMPT='Compile and link';
KWD  PROG NAME 32 NUMVAL=(1,16)
     PROMPT='Programs (first is main)';
COBOL %PROG;
IF #GE(#SEV,3);
     ABORT 'Compilation fails';
ENDIF;
LINK #ELEM(%PROG,1);
ENDPROC;
```

In this example:

- `NUMVAL=(1,16)` declares PROG as accepting up to 16 names
- the COBOL command compiles all source programs, the first one provided as the argument of the LINK command through the #ELEM builtin.

An alternative would be to compile and link several source programs into the same number of load modules.

```
PROC CLS
     PROMPT='Compile and link several programs';
KWD  PROG NAME 32 NUMVAL=(1,16)
     PROMPT='Program names';
LOCAL LN NAME 32;
UNLIST LN %PROG;
     COBOL %LN;
     IF #GE(#SEV,3);
       LET # #CAT('Compilation of ', %LN,' failed');
     ELSE;
       LINK %LN;
     ENDIF;
ENDUNLIST;
ENDPROC;
```

In this example:

- the UNLIST command performs the following:
  - extracts each name of the list PROG
  - assigns it to the local variable LN
  - and iterates the processing that follows up to the ENDUNLIST command, for each element in the list.
- if one of the compilations fails, a message is issued and the next element of the list is processed
- if the command is to cease executing if a compilation fails, the corresponding command would have to be changed to:

  ```
  ABORT #CAT('Compilation of ', %LN, ' failed');
  ```

1.8 Absentee Jobs

The submission of system-level commands for execution in batch, in parallel with the interactive session is described in the *IOF Terminal User's Reference Manual*. Commands that can be so submitted are not restricted to the system-supplied ones. User-defined commands in the IOF domain can also be submitted such as:

```
EJ PROC=CLS VALUES=MYPG;
```

```
EJ PROC=CLS VALUES=(PROG=(X1,X2,X3));
```

Search rules also apply in that case. When the parameter LIB is not specified in the EJ directive, commands are searched for in the binary input libraries defined by the MWINLIB BIN command.

It is recommended that the default binary output library be defined as being also the first one of the binary input libraries.

Thus commands created by use of MAINTAIN_COMMAND become immediately available for interactive absentee execution.

```
MWLIB   BIN  .MYBLIB;
MWINLIB BIN (.MYBLIB, others...);
```

When absentee mode is used, the cataloged binary input libraries must be cataloged:

- either in an auto-attachable catalog
- or in the SYS.CATALOG or SITE.CATALOG.

Constraints:

When absentee mode is used, the switch number 2 cannot be used in user commands as it is reserved for the system.

1.9 SYS.SPOOL Files

1.9.1 Purpose of SYS.SPOOL Files

SYS.SPOOL files are GCL work files containing:

- the Global Variables and System Variables for each user
- and the binary code of the GCL commands copies of which are generated by the MAINTAIN_COMMAND processor.

1.9.2 Use of SYS.SPOOL Files

A broad definition of code and variables can be summarized as follows:

- the binary code of GCL commands can be divided into two categories:
  - the code shared by all users of the system
  - and the private code of each user
- while the Global Variables and the System Variables are private to each user.

Shared Code:

Shared Code corresponds to commands of *pre-initialized domains* regularly used by GCOS 7:

- IOF containing the commands to start various processors
- H_NOCTX containing those commands accessible at all levels
- MAIN containing those commands available only to the Main Operator
- BREAK domain commands
- and those domains specified by the command SPOOL.

The shared code is loaded from system library SYS.HBINLIB into each of the SYS.SPOOL files during every RESTORE of the system. This code is preserved in the SYS.SPOOL files. It is reloaded:

- at RESTORE
- if command SPOOL is specified at RESTART
- if a command of a pre-initialized domain is modified in SYS.HBINLIB and if no GCL batch job is restarted at a checkpoint or at the beginning of a step.

User's Private Code:

User's Private Code is that of commands in the user's private libraries specified within the search rules of command MWINLIB BIN. This code is loaded:

- either by MWINLIB BIN for domains IOF and H_NOCTX
- or by INITGCL.

It is retained:

- until the end of the IOF session
- on termination of the job
- or until execution of the next MWINLIB BIN command.

If a command of a domain is modified in a library other than SYS.HBINLIB, the code of the command will become accessible on leaving MAINTAIN_COMMAND. *The exception is H_NOCTX, where a new MWINLIB BIN command must be executed to validate the code.*

Global and System Variables:

Global and System Variables are contained in a member created for and assigned to each user in the SYS.SPOOL file on the first use of GCL. This member is retained:

- until the end of the IOF session
- or termination of the job.

1.9.3 Number of SYS.SPOOL Files

A SYS.SPOOL file may be used by a maximum of 25 users. However, where possible, the recommended ratio is one SYS.SPOOL file to every 10 users. The maximum number of SYS.SPOOL files is 10.

1.9.4 Size of SYS.SPOOL Files

The size of SYS.SPOOL files depends on:

- the number of users connected
- the number of different processors used
- the stipulation of search rules
- and the number of global and system variables used.

Because of these factors, it is difficult to gauge an accurate size. The DISPLAY_GCL_INFO directive displays the size and contents of each SYS.SPOOL file.

1.9.5 Access Rights

All users must be authorized READ and WRITE access to the SYS.SPOOL files.

1.9.6 GCL Commands Applicable to SYS.SPOOL Files

The following GCL commands are available:

- the command SPOOL at ISL:
  - provides a list of domains to be pre-initialized as well as the implicit list of domains
  - and specifies the size and use of GCL cache.
- the commands reserved for the Main Operator are:
  - DISPLAY_GCL_CACHE
  - HOLD_GCL_CACHE
  - and RELEASE_GCL_CACHE.
- the command DISPLAY_GCL_INFO which displays for each SYS.SPOOL file:
  - its size
  - its contents
  - and the percentage of space used.

❑

2. GCL Basic Language

2.1 GCL Basic Commands

Basic GCL commands cannot be redefined. They are used for defining new commands inside a GCL procedure. Other commands can be specified outside, as well as inside, a GCL procedure.

The BASIC GCL commands are:

ABORT	ENDUNTIL	RETRY
CASE	ENDWHILE	RETURN
CASEOF	GOTO	SCALL
CHAIN	IF	SYSTEM
CONTROL	KWD	UNLIST
ELSE	LABEL	UNTIL
ENDCASEOF	LOCAL	VCALL
ENDIF	OTHER (OTHERWISE)	VCHAIN
ENDPROC	OTHERWISE	WHILE
ENDUNLIST	PROC	

The basic commands are categorized as follows:

Declarative

- CONTROL
- ENDPROC
- KWD
- LABEL
- LOCAL
- PROC

Linkage

- ABORT
- CHAIN
- RETRY
- RETURN
- SCALL
- SYSTEM
- VCALL
- VCHAIN

Conditional

- CASE
- CASEOF
- ELSE
- ENDCASEOF
- ENDIF
- ENDUNLIST
- ENDUNTIL
- ENDWHILE
- IF
- GOTO
- OTHER
- OTHERWISE
- UNLIST
- UNTIL
- WHILE

When the command entered is not a *basic* GCL command, an implicit "CALL" is generated.

Declarative commands introduce the objects such as variables, parameters and labels that are handled by procedures.

Conditional commands direct the flow of control such as jumps, loops and choices through the body of the procedure.

Linkage commands define the relations of the current procedure to other procedures that it can use.

A GLOBAL directive is considered a declarative command inside a GCL procedure as a LOCAL *basic* GCL command. This means that the declared variable is known at command call, wherever the GLOBAL statement appears in the GCL procedure. So the sequence:

```
PROC NEWCOM;
DLGB;
GLOBAL NEWVAR CHAR 10;
LET NEWVAR #;
ENDPROC;
```

leads to error `"VARIABLE NEWVAR NOT DECLARED"` at execution.

This means too that LOCAL and GLOBAL variables must be declared ***before*** executing a command that references them. So the sequence:

```
PROC COM1;                 PROC COM2;
CALL COM2;                 GLOBAL NEWVAR;
LET # %NEWVAR;             LET NEWVAR #;
ENDPROC;                   ENDPROC;
```

also leads to error `"VARIABLE NEWVAR NOT DECLARED"` at execution of the CALL.

A block is a set of commands enclosed within:

- CASEOF...ENDCASEOF
- IF...ENDIF
- PROC...ENDPROC
- UNLIST...ENDUNLIST
- UNTIL...ENDUNTIL
- WHILE...ENDWHILE.

Blocks, except PROC...ENDPROC, may be nested as parts of other blocks. In general, wherever a single command is allowed, a whole block may be substituted for that command. Since the block contains an ***integral*** number of other blocks, blocks may not overlap. Examples follow:

```
+-----                                      +-----
|  IF ....                                  |  IF ....
|                                           |
|    +----                                  |
|    |  IF ....                          +--|
|    |                                   |  |
|    |  +----                            |  |
|    |  |  WHILE ....                    |  |  WHILE ....
|    |  |                                |  |
|    |  |  ENDWHILE;                     |  |  ENDIF;
|    |  +----                            |  +-----
|    |                                   |
|    |  ENDIF;                           |
|    +----                               |
|                                        |  ENDWHILE;
|  ENDIF;                                |
+----                                    +----
```

valid nesting ***invalid nesting***

When a command is executing, all blocks that contain the command are *active*. Transferring control from within an *active* block to an inactive one is not allowed.

Commands within a procedure:

- must end with a semicolon
- may expand on more than one line
- may not share the same line with another command or with part of it (no multi-statement lines)

irrespective of the current setting of the format (line or free). All other rules that pertain to GCL commands also apply to the GCL basic commands.

Before a GCL procedure can be used, it must be compiled by means of the MAINTAIN_COMMAND (MNCMD) processor whose commands are described in Section 3. Examples of procedures, and general hints may be found in Paragraph *"Creating new GCL Commands"*.

2.1.1 ABORT

Purpose:

To abort the execution of the procedure. If the procedure is called by a sequence of procedure calls, all of the calling procedures abort. A new command is then requested from the input stream or from the user's terminal. A message can be specified to report the abort of the procedure.

Syntax:

```
ABORT

   [ MESSAGE=char78 ]

   [{ SEVERITY | SEV }=dec1 ]
```

Parameters:

MESSAGE	the text of a message to be displayed before control is returned.
SEVERITY	the value from 0 through 4, to be assigned to the System Variable #SEV on completion of the command. Default: #SEV is left unchanged

Constraints:

None

Examples:

`ABORT;`	no message issued
`ABORT MESSAGE='invalid reply';`	literal message
`ABORT %ABTMESS;`	message is a variable

```
ABORT #CAT('INVALID LENGTH: MUST BE LESS THAN ',%MAXLGTH);
```

message is an expression

2.1.2 CASE

Purpose:

Denotes the beginning of a *clause* consisting of one or more commands in a CASEOF block, and ending with:

- another *clause* introduced by a new CASE
- an OTHERWISE command
- or an ENDCASEOF command.

Syntax:

```
CASE

   EXPRESSION=( expression [ expression ]...)
```

Parameters:

EXPRESSION	list of up to 32 expressions. CASE is executed if the value of one of the expressions matches the value of the expression specified in the CASEOF command: – a literal of any type – a variable, either user-defined or system – an expression of one or more builtin functions.

Constraints:

- CASE must be included within a CASEOF block.
- Once CASE has been executed, control passes to the command following ENDCASEOF.
- If no match is found among the CASE expressions, OTHERWISE is executed. If no OTHERWISE is specified and no match is found, the procedure aborts with the appropriate error diagnostic.
- The comparison between the result of the expression specified in CASE and that specified in CASEOF is a comparison of *character* types. For a comparison of *decimal* type, use the builtin #CVDEC on the results obtained after removing non-significant leading zeroes that can cause a faulty comparison.

Examples:

`CASE -1;`	matches if CASEOF *expresssion*=-1
`CASE (1,2,3,100);`	matches if CASEOF *expresssion*=1, 2, 3 or 100
`CASE #PLUS(%X,%Y);`	matches if CASEOF *expresssion*=%X + %Y
`CASE (0,#MINUS(#PL,%RNG));`	matches: – CASEOF *expresssion*=0 – CASEOF *expresssion*=#PL - %RNG}

2.1.3 CASEOF

Purpose:

To head a CASEOF block, several of which can be nested. Depending on the *expression* the next command to be executed, is:

- either CASE
- or OTHERWISE.

Syntax:

```
CASEOF

   EXPRESSION=expression
```

Parameters:

EXPRESSION	the expression must be scalar and may be: – a literal value of any type – a variable, either user-defined or system – an expression of one or more builtin functions.

Constraints:

- CASEOF must be matched by ENDCASEOF to denote the end of the block
- CASEOF can only be followed by:
  - CASE
  - or OTHERWISE
  - or ENDCASEOF.
- No other command may appear immediately after the CASEOF.

Example:

```
CASEOF %I;
   CASE 1;              if %I=1 execute the following command(s)
          LET VAR 3;
   CASE (3, 4, 5);      if %I=3, 4 or 5 execute the following command(s)
          LET VAR 9;
   OTHERWISE;           otherwise execute the following command(s)
          LET VAR 0;
ENDCASEOF;
```

2.1.4 CHAIN

Purpose:

Activates the named procedure by continuing from the procedure currently executing. When the *chained* procedure terminates, control passes to the command following the one that initiated the procedure containing CHAIN.

Syntax:

```
CHAIN

   COMMAND=command-name31  [ parameter-reference ]...
```

Parameters:

COMMAND	the name of the command to be executed. This must not be an expression; it must be a literal name value.
parameter-reference	Keyword, positional parameter or self-identifying value notation to be passed to the command. See the *IOF Terminal User's Reference Manual*.

Constraints:

- All parameters specified must agree in type, number, length and value with their specifications in the body of the command definition.
- command-name must refer to a command definition which must be:
  - in the terminal user's search path, see
    Section 1 of this manual
    and the *IOF Terminal User's Reference Manual*
  - accessible from the user's current environment, see the *IOF Terminal User's Reference Manual.*

Example:

```
PROC P;       +-------> PROC Q;       +------> PROC R;
  .           |           .           |          .
  .           |           .           |          .
  .           |           .           |          .
CALL Q; ------+           .           |          .
                        CHAIN R; ------+          .
                          .                    RETURN; ------+
COMMAND 2 <------+        .                      .           |
  .              |        .                      .           |
  .              |        .                      .           |
  .              |      ENDPROC;               ENDPROC;      |
ENDPROC;         |                                           |
                 +-------------------------------------------+
```

2.1.5 CONTROL

Purpose:

Specifies global checks to be performed on procedure parameters or other variables before execution of the procedure starts.

Syntax:

```
CONTROL

  CHECK=( bool [ bool ]...)

  { MSSG }
  {      }=char78
  { MSG  }
  [ WHEN=( bool [ bool ]...)]
```

Parameters:

CHECK	Lists up to 32 checks to be performed: – succeeds if all the expressions are true (=1) – fails if any one of the expressions is not true (=0).
MSSG	Message to be displayed if check fails.
WHEN	Lists up to 32 conditions for which checking is done: – if all conditions are true (=1), CHECK is processed – if one is false (=0), the rest of CONTROL is ignored and the next command is processed.

Constraints:

- All checks specified in the KWD statements are verified before the procedure executes.
- The check fails if the condition in CHECK is not 1, resulting in the following:
  - the procedure aborts
  - and the message specified in MSSG is displayed.
- In case of an UNLOCKED procedure, CONTROL messages can be prefixed by proc-name/line-number. This prefix is displayed when:
  - the terminal works in 'visual mode' (FORMS supported) and DEBUG mode is set (in this case the message can be truncated)
  - the terminal works in 'serial mode' (like MAIN console).

  In both cases the line number given in the prefix has no meaning.

Examples:

```
CONTROL  WHEN=#EXIST(COMMAND)
         CHECK=#NOT(#EXIST(COMFILE))
         MSSG='COMMAND AND COMFILE ARE MUTUALLY EXCLUSIVE';
```

- if COMMAND exists but not COMFILE, execute the command
- if both COMMAND and COMFILE exist, issue message and cancel execution.

```
CONTROL  CHECK=#GT(%MAXVAL,%MINVAL)
         MSSG='MAXVAL MUST BE GREATER THAN MINVAL';
```

- if MAXVAL is greater than MINVAL, execute the command
- if MAXVAL is less than or equal to MINVAL, issue message and abort execution.

2.1.6 ELSE

Purpose:

Denotes an alternative in an IF block.

Syntax:

```
ELSE
```

Parameters:

None

Constraints:

- The conditional expression in IF command is evaluated as follows:
  - if =1, the commands following IF execute but those following ELSE are skipped
  - if =0, control is transferred to the command following ELSE, if any.
- ELSE must be within an IF block.

Examples:

```
IF #EQ(%A,%B);
   IF #EQ(%A,%C);
      LET X %Y;          command to be executed if %A=%B=%C
   ELSE;
      LET X %Z;          command to be executed if %A=%B and %A^=%C
   ENDIF;
ENDIF;
```

```
IF #EQ(%A,%B);
   IF #EQ(%A,%C);
      LET X %Y;          command to be executed if %A=%B=%C
   ENDIF;
ELSE;
   LET X %Z;             command to be executed if %A^=%B
ENDIF;
```

2.1.7 ENDCASEOF

Purpose:

Denotes the end of a CASEOF block.

Syntax:

```
ENDCASEOF
```

Parameters:

None.

Constraints:

ENDCASEOF must match its corresponding CASEOF that heads the block.

Example:

```
CASEOF %KIND;
   CASE AVERAGE;      if %KIND=AVERAGE execute command(s) following
        CALL AVERAGE;
   CASE STATISTIC;    if %KIND=STATISTIC execute command(s) following
        CALL STATISTIC;
   CASE CHECK;        if %KIND=CHECK execute command(s) following
        CALL CHECK;
   OTHER;             otherwise execute command(s) following
        CALL ERROR;
ENDCASEOF;
```

2.1.8 ENDIF

Purpose:

Denotes the end of an IF block.

Syntax:

```
ENDIF
```

Parameters:

None.

Constraints:

ENDIF must match its corresponding IF that heads the block.

Example:

```
IF #EQ(%K,TRUE);
   LET # TRUE;                command to be executed if %K=TRUE
ELSE;
   LET # FALSE;               command to be executed if %K is not=TRUE
ENDIF;
```

2.1.9 ENDPROC

Purpose:

Denotes the end of a procedure.

Syntax:

```
ENDPROC
```

Parameters:

None

Constraints:

Each procedure must begin with PROC and end with ENDPROC.

Example:

```
PROC P ....;
  .  }
  .  }   commands that define procedure P
  .  }
ENDPROC;
```

2.1.10 ENDUNLIST

Purpose:

Denotes the end of an UNLIST block.

Syntax:

```
ENDUNLIST
```

Parameters:

None

Constraints:

ENDUNLIST must match UNLIST that heads the block.

Example:

```
UNLIST ITEM %LIST;
    .  }
    .  }   sequence of commands to be repeatedly executed until all
    .  }   elements of LIST have been supplied to variable ITEM
    .  }
ENDUNLIST;
```

2.1.11 ENDUNTIL

Purpose:

Denotes the end of an UNTIL block.

Syntax:

```
ENDUNTIL
```

Parameters:

None

Constraints:

ENDUNTIL must match its corresponding UNTIL that heads the block.

Example:

```
UNTIL #OR (#LT(%A,0), #EQ(%S, STOP));
    .  }
    .  }   sequence of commands to be repeatedly
    .  }   executed until %A<0 or %S=STOP
    .  }
ENDUNTIL;
```

2.1.12 ENDWHILE

Purpose:

Denotes the end of a WHILE block.

Syntax:

```
ENDWHILE
```

Parameters:

None.

Constraints:

ENDWHILE must match its corresponding WHILE that heads the block.

Example:

```
WHILE #OR (#EQ(%I,0), #EQ(%I,1));
    .  }
    .  }   sequence of commands to be repeatedly
    .  }   executed while %I=0 or 1
    .  }
ENDWHILE;
```

2.1.13 GOTO

Purpose:

Modifies the flow of execution by branching to a label defined by a LABEL command.

Syntax:

```
GOTO
   LABEL=label-name31
```

Parameter:

LABEL	Must be a name literal: label to branch to defined by a LABEL command. Expressions are not allowed.

Constraints:

The label must be unique within the procedure defined by LABEL in:

- either the same active block
- or an active outer block which includes the current block.

Examples:

`GOTO ERROR;`	branch to ERROR
`GOTO SCAN_INPUT;`	branch to SCAN_INPUT
`GOTO END_OF_PROCESS;`	branch to END_OF_PROCESS

```
       +-----                              +-----
       |                                   |
       | +----                             |----
       | |                                 | |
       | |                                 | |
       | |  GOTO --+--+                    | |  GOTO ---+
       | |         |  |                    | |          |
       | |         |  |                    | |          |
       | |     <--+   |                    | +----      |
       | +----        |                    | +----      |
       |              |                    | |          |
       |              |                    | |      <---
       |              |                    | |
       |       <-----+                     | +----
       |                                   |
       +-----                              +-----
```

valid branches ***invalid branches***

2.1.14 IF

Purpose:

Tests the value of a specified expression in an IF block to control the flow of execution according to the result of the test. IF blocks can be nested.

Syntax:

```
IF  CONDITION=( bool [ bool ]...)
```

Parameter:

CONDITION	Lists up to 32 conditions to be checked: – the *expression* is true if all conditions =1 – otherwise it is false.

Constraints:

- Conditions are evaluated as follows:
  - if all conditions =1:
    - commands following IF are executed
    - but those following ELSE if specified, are skipped
  - if one condition =0, control passes to:
    - either the command following ELSE if specified
    - or the command following ENDIF if ELSE is omitted.
- IF must be matched by ENDIF to denote the end of the IF block.
- ELSE can optionally be included in the block.

Examples:

```
IF #EQ(%A,%B)
    .  }
    .  }   commands to be executed if %A = %B
    .  }
ENDIF;
    .  }
    .  }   commands to be executed whether %A=%B or not
    .  }
    .
IF #GT(%C,%D);
    .  }
    .  }   commands to be executed if %C>%D
    .  }
ELSE;
    .  }
    .  }   commands to be executed if %C<=%D
    .  }
ENDIF;
```

2.1.15 KWD

Purpose:

Defines the name and characteristics of a *keyword* being a procedure argument.

Syntax:

```
KWD

   NAME=( name31 [ name31 ]...)

   [      { CHAR                                  }]
   [ TYPE={ BOOL | DEC | FILE | FSET | HEXA | LIB }]
   [      { NAME | OUTPUT | RFILE | STAR | VOLUME }]

   [ LENGTH=dec3 ]

   [ NUMVAL=( dec2 [ dec2 ])]

   [ DEFAULT=( value78 [ value78 ]...)]

   [ VALUES=( condition [ condition ]...)]

   [ PROMPT=char40 ]

   [ NOTE=( char78 [ char78 [ char78 ]])]

   [ HELP=name30 ]

   [{ MSG | MSSG }=char78 ]

   [ ASK=dec2 ]

   [ CONCEAL=bool ]

   [{ DLGTH | DISPLAY_LENGTH }=dec3 ]
```

Parameters:

NAME	Lists up to 32 names: – the first being the main name for the keyword – the following being aliases. The keyword can be referred to by its name or alias.
TYPE	Type of variable. See Paragraph *"Types"*. Default: CHAR (plain character string)
LENGTH	Maximum length of each element of the variable. Default and maximum applicable values depend on the type of the variable being defined.

Type	**Maximum Length**	**Default Length**
CHAR	255	80
BOOL	1	1
DEC	31	31
FILE	255	44
FSET	255	80
HEXA	8	8
LIB	255	44
NAME	44	31
OUTPUT	255	80
RFILE	255	80
STAR	88	31
VOLUME	255	80

NUMVAL	Pair of numbers denoting minimum and maximum number of variables, where max => min and dec2 <= 64. When only one number is provided, both the maximum and minimum are set to that value. Default: (min=0,max=1)
DEFAULT	Lists up to 64 values defining the initial value of the variable as any expression, including list expressions and literals. Variables referenced in such expressions are: – either System Variables – GLOBAL Variables or KWDs in calling procedure context.

	DEFAULTs must be consistent with the attributes specified in TYPE, LENGTH, NUMVAL and VALUES.
	Default: Value is initially undefined and trying to use it causes an error. Errors in computing dynamic defaults do not affect execution of the procedure and no message appears.
VALUES	Lists up to 32 conditions that values assigned to the variable must match. At least one of the conditions must be satisfied: an OR is performed on the conditions and the value assigned. Conditions must be consistent with the TYPE and LENGTH specified for the variable.
	Default: Values compatible with the variable TYPE and LENGTH are assigned. The conditions applying to individual elements of the variable are summarized in the following table. Different types of conditions may appear in the same list.

Example:
`VALUES=(&9 * $/)` means applies to a variable which starts with a digit, or is an asterisk or contains a slash.

Type of Condition	Form	Examples
discrete value	value	A 123 *
relational, see Note	{> }value {< }value {>=}value {<=}value {= }value	>2 <32 >=0 <=XYZ =0

range, see Note	{> } {> }
	value{ }*{ }value
	{>=} {>=}
	
	{< } {< }
	value{ }*{ }value
	{<=} {<=}
		100>*>=10
		0>*>=-20
		
		A<=*<H
		0<*<=99
		A<=*<=Z

contains	$value	$XYZ $. $',' *contains a comma*
starts with	/value	/AXY /' ' *starts with space* /.
starts with digit	&9	&9
starts with letter	&A	&A
alphanumeric start	&X	&X
not	^any of above	^=* ^$XYZ ^&9

NOTE:
Non-numeric values are compared by their EBCDIC collating sequences.

PROMPT

Prompt for user in menu mode to enter a value for the associated variable and is displayed by the command:

```
LET variable-name #
```

where value read from terminal is assigned to the variable.

	Example:
	`GLOBAL X DEC 3 PROMPT='Your age?'`
	`LET X #`
	`Your age?`

NOTE	Lists up to 3 lines of text to be displayed before the prompt. Default: Only the prompt appears.
HELP	Name of member in SITE.HELP library storing Help text to give the user information on the keyword. Helps are created through the CREATE_HELP_TEXT utility.
MSSG	Error message to be displayed if a value inconsistent with the TYPE, NUMVAL and VALUES, is entered. Default: Standard system error message
ASK	Number of fields to be prompted: – maximum value is 64 – if ASK=0, the keyword is not prompted. `NUMVAL`*`min`* `<= ASK <= NUMVAL`*`max`* Default: NUMVAL`max`
CONCEAL	Determines if information keyed in is displayed:
`=1`	Conceal input
`=0`	Default: Display input.
DISPLAY_LENGTH	Length reserved on the screen for a value or values in a list. The user can extend this field by entering #CC as the last non-blank character of the field. Default: LENGTH

NOTES:

Keywords with NUMVAL min >0 are called, mandatory keywords.

Keywords with NUMVAL min =0 and ASK = 1 are called, optional keywords.

Keywords with NUMVAL min =0 and ASK = 0 are called, optional hidden keywords.

Constraints:

- Restrictions on values are:
  - Each keyword is treated as a Local Variable within the procedure in which it is defined.
  - A keyword can be referenced positionally, namely, %1, %2, %3...: `IF #EQ(%2,20)` tests if the second keyword equals 20.
  - Any value subsequently assigned to the keyword must be compatible with the values specified in the TYPE, LENGTH, NUMVAL and VALUES.
- During a session, more than one keyword with the same name can be active:
  - only the most local one accessed
  - the others are *masked* for the duration of the procedures redefining them with the exception of the SCALL command.
- Restrictions on prompts and displays are:
  - DISPLAY_LENGTH must be less than or equal to the explicit or implicit LENGTH.
  - The number of fields prompted is specified by ASK and cannot exceed one screen.
- Restrictions on keywords declaration order:
  - mandatory keywords must be declared first.
  - optional keywords must be declared next.
  - optional hidden keywords must be declared at the end.

Examples:

```
KWD (PRINTING-WIDTH,PW) DEC;      decimal with two names

KWD COUNT DEC 3 VALUES=>0;        decimal up to three positive digits

KWD LIST NAME 20                  each name having up to 20 characters
    NUMVAL=(1,30)                 list of one to 30 names
    VALUES=&A;                    and beginning with a letter

KWD REPLY NAME 3
       VALUES=(YES NO);           variable to take values only YES
                                  and NO

KWD C;                            defaults to string of up to 80 characters

KWD B BOOL;                       boolean variable

KWD D DEC VALUES=(0 10<=*<=99);   variable 0 or decimal from 10
                                  through 99

KWD PAGES DEC LENGTH=2
    DEFAULT=1 VALUES=1<=*<99      if decimal variable not between 1
    PROMPT='Number of Pages'      and 98  (default 1), issue error message
    MSSG='Invalid number of Pages';

KWD CLASS LENGTH=5 DEFAULT=ALL    character variable list
    NUMVAL=(0,8)                  with default value ALL and
    NOTE:=('BUILTIN CLASS:',      three lines of text
    'ALL RELAT BOOL CHAR',        before the keyword PROMPT
    'LIST OBMGT IS_IT ARITH')
    PROMPT='SELECT ONE OF 8 CLASSES';

KWD KIND NUMVAL=(3,15)
    HELP=HELP-KIND                help in SYS.HELP or SITE.HELP
    ASK=3;                        character variable list of 3  elements

KWD PASSWORD LENGTH=3 CONCEAL;    character variable with attribute
                                  conceal
```

2.1.16 LABEL

Purpose:

Defines a label referred to in a GOTO command.

Syntax:

```
LABEL

   LABEL = label_name_31
```

Parameter:

LABEL	Must be a name literal value: Defines the label used to refer to the command following. An expression is not allowed.

Constraints:

The label name must be unique within the procedure in which it is defined.

Example:

```
IF #EXIST (NM);
   GOTO RTRY;
   .
   .
   .
LABEL RTRY;
RETRY MSSG='Name specified already exists: reenter name';
```

2.1.17 LOCAL

Purpose:

Defines a local GCL variable.

Syntax:

```
LOCAL

   NAME=name31

   [      { CHAR                                }]
   [ TYPE={ BOOL | DEC | FILE | FSET | HEXA | LIB }]
   [      { NAME | OUTPUT | RFILE | STAR | VOLUME }]

   [ LENGTH=dec3 ]

   [ NUMVAL=( dec2 [ dec2 ])]

   [ DEFAULT=( value78 [ value78 ]...)]

   [ VALUES=( condition [ condition ]...)]

   [ PROMPT=char40 ]
```

Parameters:

NAME	Name of the local variable
TYPE	Type of variable. See Paragraph *"Types"*.
LENGTH	Maximum length of each element of the variable.

Default and maximum applicable values depend on the type of the variable being defined.

Type	**Maximum Length**	**Default Length**
CHAR	255	80
BOOL	1	1
DEC	31	31
FILE	255	44
FSET	255	80
HEXA	8	8
LIB	255	44
NAME	44	31
OUTPUT	255	80
RFILE	255	80
STAR	88	31
VOLUME	255	80

NUMVAL

Pair of numbers denoting the minimum and maximum number of variables, where `max` => `min` and `dec2` <= 64.

When only one number is provided, both the maximum and minimum are set to that value.

Default: (min=0,max=1)

DEFAULT

Lists up to 64 values defining initial value of variable as any expression, including list expressions and literals.

Variables referenced in such expressions are:
- either System Variables
- GLOBAL Variables or KWDs in calling procedure context.

DEFAULTs must be consistent with the attributes specified in TYPE, LENGTH, NUMVAL and VALUES.

	Default: Value is initially undefined and trying to use it causes an error. Errors in computing dynamic defaults do not affect execution of the procedure and no message appears.
PROMPT	Prompt for user in menu mode to enter a value for the associated variable and is displayed by the command:

```
LET variable-name #
```

Example:

```
GLOBAL X DEC 3 PROMPT='Your age?'
LET X #
Your age?
```

VALUES	Lists up to 32 conditions that values assigned to the variable must match. At least one of the conditions must be satisfied: an OR is performed on the conditions and the value assigned. Conditions must be consistent with the TYPE and LENGTH specified for the variable.
	Default: Values compatible with the variable TYPE and LENGTH are assigned. The conditions applying to individual elements of the variable are summarized in the following table. Different types of conditions may appear in the same list.

Example:

`VALUES=(&9 * $/)` means applies to a variable which starts with a digit, or is an asterisk or contains a slash.

Type of Condition	Form	Examples
discrete value	value	A
		123
		*
relational, see Note	{> }value	>2
	{< }value	<32
	{>=}value	>=0
	{<=}value	<=XYZ
	{= }value	=0
range, see Note	{> } {> }	100>*>=10
	value{ }*{ }value	0>*>=-20
	{>=} {>=}	
	{< } {< }	A<=*<H
	value{ }*{ }value	0<*<=99
	{<=} {<=}	A<=*<=Z
contains	$value	$XYZ
		$.
		$',' *contains a comma*
starts with	/value	/AXY
		/' ' *starts with space*
		/.
starts with digit	&9	&9
starts with letter	&A	&A
alphanumeric start	&X	&X
not	^any of above	^=*
		^$XYZ
		^&9

NOTE:
Non-numeric values are compared by their EBCDIC collating sequences.

Constraints:

- Expressions are not allowed for parameters in LOCAL. *Only literal values can be assigned.*
- During a session, more than one keyword with the same name can be active:
  - only the most local one accessed
  - the others are *masked* for the duration of the procedures redefining them.

Examples:

```
LOCAL COUNT DEC 3                decimal up to three positive digits
       VALUES=>0;

LOCAL LIST NAME 20               each name up to 20 characters
       NUMVAL=(1,30)             list of one to 30 names
       VALUES=&A;                beginning with a letter

LOCAL REPLY NAME 3
       VALUES=(YES NO);          variable taking only YES or NO

LOCAL C;                         defaults to string of up to 80 characters

LOCAL B BOOL;                    define a boolean variable

LOCAL D DEC
       VALUES=(0 10<=*<=99);     variable 0 or decimal from 10 through 99

LOCAL DELTA DEC DEFAULT=1515;     decimal with initial value 1515
```

2.1.18 OTHERWISE (OTHER)

Purpose:

Denotes the beginning of an OTHERWISE *clause* in a CASEOF block. If CASEOF values do not match any CASE values, OTHERWISE is executed.

Syntax:

```
{ OTHERWISE }
{           }
{ OTHER     }
```

Parameters:

None.

Constraints:

Restrictions on positioning and processing:

- OTHERWISE must be within a CASEOF block and must come after all CASEs
- If OTHERWISE is omitted, and no match is found among CASE expressions, the procedure aborts with the appropriate error diagnostic.

Examples:

```
CASEOF %I;
       CASE (1,2,3)        if %I=1, 2 or 3 execute following command(s)
       LET VAR 1;
       OTHER;              otherwise execute following command(s)
       LET VAR 0;
ENDCASEOF;

CASEOF %I;
       CASE (1,2,3);       if %I=1, 2 or 3 execute following command
       LET VAR 1;
ENDCASEOF;                 OTHER absent: procedure aborts if
                           %I not=(1,2,3)
```

2.1.19 PROC

Purpose:

Heads a procedure definition and defines its characteristics.

Syntax:

```
PROC

   NAME=( name31 [ name31 ]...)

   [ PROMPT=char40 ]

   [ HELP=name31 ]

   [{ PRTY | PRIORITY }={ 0 | dec3 }]

   [ ACCESS={ -1 | ( dec3 [ dec3 ]...)}]

   [ HIDE={ 0 | ( dec3 [ dec3 ]...)}]

   [ OPACC=( dec2 [ dec2 ]...)]

   [ OPHID={ 0 | ( dec2 [ dec2 ]...)}]

   [ LOCK={ 0 | bool }]

   [ LIMITED_ACCESS={ 0 | bool }]
```

Parameters:

NAME	Lists up to 32 names by which the procedure is referenced: – the first is the main name for the procedure – the following are aliases.

Names of GCL basic commands must not be used, namely:

```
- ABORT      - ENDUNTIL     - RETRY
- CASE       - ENDWHILE     - RETURN
- CASEOF     - GOTO         - SCALL
- CHAIN      - IF           - SYSTEM
- CONTROL    - LABEL        - UNLIST
- ELSE       - KWD          - UNTIL
- ENDCASEOF  - LOCAL        - VCALL
- ENDIF      - OTHER        - VCHAIN
- ENDPROC    - OTHERWISE    - WHILE
- ENDUNLIST  - PROC
```

A GCL procedure created with the same name as a system GCL command such as CBL, will override the system command which will no longer be available.

PROMPT	Prompt displayed with command name when user requests list of commands for the domain containing the command.
HELP	Name of member stored in SITE.HELP library containing help texts for the command generated by the CREATE_HELP_TEXT utility.
ACCESS	Lists up to 32 families to which the command belongs, each family identified from 1 through 256. Default: -1, all families have access to the command.
HIDE	Lists up to 32 families from which the command is to be hidden, each family identified from 1 through 256:
=-1	Hide the command from all the families to which it belongs.
=0	Default: Command is unhidden and prompted for all families.

OPACC	Each value specified in OPACC must also be specified in OPHID: Lists up to 8 categories of users who have access to the command. The categories are specified through catalog rights listed below: `1` Main Operator (default for domain MAIN) `4` GCL (default for domains other than MAIN) `5` All IOF Users `6` Station Operator – OPACC is a second level of access control operating on families specified by ACCESS – OPACC can only impose further restrictions in not granting more access rights than specified in ACCESS. If list of values is specified, user's project must have at least one of the access rights in the list to use command.
OPHID	Each value specified in OPHID must also be specified in OPACC: Lists up to 8 categories of users from whom the command is to be hidden. The categories are specified through catalog rights listed for OPACC.
`=-1`	Equivalent to `OPHID=(1,4,5,6)`: Command is available but hidden from all categories.
`=0`	Default: Command is unhidden for all categories in OPACC.
PRIORITY	Order in which command names are displayed in list of commands for domain. Maximum is 255, lowest priority. Default: 0, highest priority
LOCK	Determines user's access to procedure:
`=1`	User cannot list or modify procedure definition. Error messages do not refer to individual lines of locked procedure.
`=0`	Default: Procedure is not locked. Values of CONFIG parameters GCLKPROJ and GCLKSADM determine who can modify a procedure or the value of LOCK:

GCLKPROJ	
`=NO`	Only the user who compiled the procedure can: – access the source and modify the value of LOCK – and delete the procedure.
`=YES`	All users belonging to the same project have the same rights as user who compiled the procedure.
GCLKSADM	
`=NO`	Users of project SYSADMIN have no special rights to procedures that have the attribute LOCK.
`=YES`	Users of project SYSADMIN have the same rights as the user who compiled the procedures.
LIMITED_ACCESS	Determines how the procedure is called. There is no link between attribute LIMITED_ACCESS and environments. A command having this attribute is not directly accessible for all projects, except SYSADMIN. Values:
`=0`	(default) the procedure can be called directly.
`=1`	the procedure can only be called by another procedure or by a user of project SYSADMIN.

Constraints:

- Each procedure begins with PROC and ends with ENDPROC.
- Only literal values can be specified in PROC; *expressions are not allowed.*

Examples:

```
PROC P;

PROC NAME=(CREATE_PROJECT,CRP)
     PROMPT='create a project and its attributes'
     HELP=H_DCT_CRP
     LOCK;

PROC (DELETE_LM,DLM)
     PROMPT='Delete an LM library'
     HELP=HLP_DLM
     ACCESS=(1,4,5,200);

PROC DNP
     PROMPT='Define New Password'
     ACCESS=(1,4,200)
     HIDE=(1,4);

PROC (CHANGE_UN,CUN)
     PROMPT='Change User Name'
     ACCESS=(4,24,25,200)
     OPACC=(1,4,6);

PROC ALCIS
     PROMPT='Allocate an Indexed Sequential file'
     ACCESS=(4,24,25,200)
     OPACC=(1,4,6)
     OPHID=(5,6);

PROC G
     PROMPT='Command displayed with a lower priority'
     PRTY=100;
```

2.1.20 RETRY

Purpose:

Retries executing the procedure in which it appears by:

- passing control to
  - either the procedure specified
  - or the beginning of the outermost procedure initiating the current procedure
- and re-prompting the parameters to be reentered by the user for retry.

Syntax:

```
RETRY

   [ MESSAGE=char78 ]

   [{ SEV | SEVERITY }=dec1 ]

   [{ PROC | PROCEDURE }=name31 ]

   [ HIGHLIGHT=( name31[.dec2 ][ name31[.dec2 ]]...)]
```

Parameters:

MESSAGE	Any valid character expression: Message to be displayed when control is returned.
SEVERITY	Value from 0 through 4 to be assigned to System Variable #SEV when the command is handled like an ABORT command. Default: #SEV is left unchanged.
PROCEDURE	Name of procedure to which control passes. Control passes to the first command of: – either current procedure – or procedure in the active stack. Control passes to the outermost procedure that initiated the current procedure when: – parameter is omitted – or procedure does not match one in the stack.

HIGHLIGHT	Keywords to be highlighted as prompts on re-prompting. When keyword is a list, the rank of the element in the list can be specified in the format keyword.rank. (Up to 64 keyword names can be specified in this list.)

Constraints:

If the user cannot intervene such as when commands are being read from a file, RETRY is treated as ABORT.

Examples:

`RETRY;`	no message
`RETRY MESSAGE='Invalid character in string S';`	literal message
`RETRY %RTYMESS;`	variable message
`RETRY #CAT('INVALID VALUE IN GLOBAL G->',%G);`	message is expression

2.1.21 RETURN

Purpose:

Terminates the execution of the procedure in which it appears to return control to the calling procedure.

Syntax:

```
RETURN
```

Parameters:

None.

Constraints:

- Transfer of control depends on how the procedure was activated:
  - if by CALL, VCALL or SCALL, control passes to next command
  - if by CHAIN or VCHAIN, control passes to command following that which activated the procedure.
- ENDPROC is treated as a RETURN when reached in the flow of execution.

Examples:

```
 .   +-----> P1   +-----> P2         .   +-----> P1   +-----> P2
 .   |        .   |        .         .   |        .   |        .
 .   |        .   |        .         .   |        .   |        .
 .   |        .   |        .         .   |        .   |        .
 .   |        .   |        .         .   |        .   |        .
CALL P1      CALL P2    RETURN      CALL P1      CHAIN P2   RETURN
 .                .        |         .                         |
 . <------+       . <--+   |         . <------+                |
 .        |       .    |   |         .        |                |
 .        |       .    |   |         .        |                |
 .        |       .    +---+         .        |                |
 .        |       .                  .        |                |
 .        +----- RETURN              .        +----------------+
```

2.1.22 SCALL

Purpose:

Scans a string of characters as the set of parameter values to be passed to the specified procedure. If the set is illegal or invalid, prompts the user for correction.

Syntax:

```
SCALL

   COMMAND=name31

   [ STRING=char250 ]
```

Parameters:

COMMAND	Name of the procedure to be activated. Unlike CALL and VCALL, this can be a name expression yielding a value denoting the name of an existing command.
STRING	String of characters possibly an expression, denoting the parameter values when calling the procedure.

Constraints:

- Restrictions on command function:
  – if STRING denotes a valid parameter set, SCALL behaves like CALL
  – if not, it behaves like VCALL.
- Keywords of *calling* procedure redefined with the same names, TYPEs, LENGTHs and NUMVALs as the *called* procedure are assigned initial values of corresponding parameters of the *called* procedure.

Example:

```
LET PN MYPROC; LET ST 'P1=2 P2=4'; SCALL %PN %ST;
```

is equivalent to:

`CALL MYPROC P1=2 P2=4` *or* `VCALL MYPROC P1=2 P2=4`

depending on whether P1=2 and P2=4 are valid parameters for MYPROC

2.1.23 SYSTEM

Purpose:

Passes a string to the calling processor.

Syntax:

```
SYSTEM
   STRING=char250
```

Parameter:

STRING	All references to variable or system variable or builtin is replaced by the corresponding value before result is passed to the calling processor.

Constraints:

Only allowed in user-specific domains.

Examples:

`SYSTEM %A;`	value of %A is passed
`SYSTEM #CAT('DEL_',%1,';');`	result of expression is passed
`SYSTEM 'OPTIONS=' %A ';';`	value passed if %A=YNY is: OPTIONS=YNY;

2.1.24 UNLIST

Purpose:

A sequence of commands between UNLIST and ENDUNLIST is executed for each element in the specified list. At each execution, the scalar variable is given the value of the next element in the list, starting with the first. UNLIST blocks can be nested.

Syntax:

```
UNLIST

   SCALAR=name31

   LIST=list-expression
```

Parameters:

SCALAR	Any name expression: the name of scalar variable to control the loop.
LIST	List expression of up to 64 elements to be UNLISTed.

Constraints:

- UNLIST must be matched by its corresponding ENDUNLIST denoting end of block.
- Restrictions on values:
  - if list is empty, the block is not executed
  - if values in list within the block are modified, results are unpredictable.
- Restrictions on scalar variable:
  - when all list elements have been supplied to the scalar variable, the UNLIST loop terminates and the command following ENDUNLIST is executed
  - SCALAR and LIST must be of compatible TYPEs, LENGTHs and VALUEs, so that all elements of LIST can be assigned to SCALAR.

Examples:

`UNLIST CURRENT %PROGRAM_LIST;`
`  COBOL SOURCE=%CURRENT;`
`ENDUNLIST;`
	perform loop for all elements of PROGRAM_LIST

`UNLIST I #INDEX_SET(30);`
`     .`
`     .`
`     .`
`ENDUNLIST;`
	perform loop for I=1 through 30

2.1.25 UNTIL

Purpose:

A sequence of commands in the UNTIL block is repeatedly executed until the conditional expression =1. UNTIL blocks can be nested.

Syntax:

```
UNTIL

   CONDITION=( bool [ bool ]...)
```

Parameter:

CONDITION	List of up to 32 conditions to be checked. The expression is true if all conditions =1.

Constraints:

- UNTIL must be matched by its corresponding ENDUNTIL denoting end of block.
- If all conditional expressions =1 when UNTIL is encountered:
  - the UNTIL block is not executed
  - and execution proceeds with the command following ENDUNTIL.

Example:

```
UNTIL #EQ(%CNT,10);
     .  }
     .  }     sequence of statements to be repeatedly executed until %CNT=10
     .  }
     .
ENDUNTIL;
```

2.1.26 VCALL

Purpose:

Verbose CALL: executes the named procedure with display of specified parameters for the user to confirm or modify.

Syntax:

```
VCALL

   COMMAND=command-name31

   [ parameter-reference ]...
```

Parameters:

COMMAND	Must be a name literal value: the name of the command whose execution is requested. An expression is not allowed.
parameter-reference	Keyword, positional parameter or self-identifying value notation to be passed to the command. See the *IOF Terminal User's Reference Manual*.

Constraints:

- The difference between VCALL and CALL is that before execution, VCALL displays the parameters being passed for the user to accept, reject or modify.
- Parameter values displayed comprise defaults and those explicitly specified.
- If the user cannot intervene such as when commands are being read from a file, VCALL is identical to CALL.

Examples:

`VCALL MYPROC;`	MYPROC procedure, no parameters
`VCALL AVERAGE %LIST;`	AVERAGE procedure with one parameter %LIST

2.1.27 VCHAIN

Purpose:

Verbose CHAIN: executes the named procedure displaying the specified parameters for the user to confirm or modify. VCHAIN replaces the current procedure with this new one.

Syntax:

```
VCHAIN

        COMMAND=command-name31

        [ parameter-reference ]...
```

Parameters:

COMMAND	Must be a name literal value: the name of the command whose execution is requested. An expression is not allowed.
parameter-reference	Keyword, positional parameter or self-identifying value notation to be passed to the command. See the *IOF Terminal User's Reference Manual*.

Constraints:

- The difference between CHAIN and VCHAIN is that before execution, VCHAIN displays all parameters being passed for the user to accept, reject or modify.
- Values displayed comprise defaults and those explicitly specified.
- If user cannot intervene such as when commands are being read from a file, VCHAIN is identical to CHAIN.
- The *chained* procedure replaces the current one, so on return, control passes not to the procedure containing VCHAIN, but to the command following.

Examples:

`VCHAIN P1;`	procedure P1 with no parameters
`VCHAIN LISTER LGT=%L1;`	LISTER procedure with one parameter LGT=%L1

2.1.28 WHILE

Purpose:

The sequence of commands in the block is repeatedly executed for as long as the conditional expression is WHILE =1. WHILE blocks can be nested.

Syntax:

```
WHILE

   CONDITION=( bool [ bool ]...)
```

Parameter:

CONDITION	List of up to 32 conditions to be checked. The expression is true if all conditions =1.

Constraints:

- WHILE must be matched by its corresponding ENDWHILE denoting end of block.
- If one of conditional expressions =0 when WHILE is encountered:
  – the WHILE block is not executed
  – and execution proceeds with the command following ENDWHILE.

Example:

```
WHILE #LT (%A,%CNT);
    .  }
    .  }   sequence of commands to be repeatedly executed while %A<%CNT
    .  }
    .
ENDWHILE;
```

2.2 System Variables

System variable names begin with the special character #. They can be used in expressions like any other user-defined variable. The only difference is that assigning a value to a system variable, in addition to modifying its current value, will also alter some characteristics of the user operating environment.

System variables are parameters through which the system supports the user's session. By altering their values, the user can tailor system behavior to his own requirements. The set of all system variables constitutes the *user's profile* listing the user's requirements and performance characteristics. The DISPLAY_PROFILE (DP) directive can be used to display part or all of this user profile.

System variables have the same characteristics as other variables such as:
- type
- shape
- and length.

They may be undefined. The normal rules for the assigning values to a variable also apply to system variables. Some system variables but not all, have a default value being an initial value set at the beginning of the session but can be changed later.

This Section lists all available system variables with:
- their meanings
- applicable values
- and effects on the environment.

NOTE:

The following system variables cannot be used in batch:

```
#AUTOLF      #EXPTABS     #PL          #TABS
#CC          #INVCHAR     #PROMPT      #WSTATION
#CSET        #PAGEMODE    #PW          #ZOK
#DI          #PAGETOP     #ROLL
```

The system variable # can be used in batch mode only in the context "LET # expression" but not in the context "LET V #".

Description of System Variables

System variables are listed in alphabetical order. Each system variable is described under the following headings:

1. Meaning: The function of the system variable.
2. Type: The type of the system variable:
   - `char` any character string
   - `bool` a boolean value being 0 or 1
   - `lib` a library reference
   - `dec` a decimal value
   - `name` a name value.
3. Shape: The form of the system variable, expressed as:
   - a minimum number of elements
   - a maximum number of elements.

   Values that a shape can take are:
   - (1,1) denotes a scalar that must be initialized
   - (0,1) denotes a scalar that may be initialized
   - (0,3) denotes a list of zero to three elements
   - (32,32) a list of exactly 32 elements.
4. Length: The maximum length of any single element.
5. Default value: The value of the variable at the beginning of the session. An undefined or uninitialized variable is:
   - considered empty
   - and denoted by ().
6. Constraints: The conditions values must satisfy for assignment to system variable.
7. Effect: The result of assigning a value to system variable.
8. Level of assignment:
   Some system variables may be modified in all contexts, others may be modified only when at system level. System variables may appear anywhere, except on the left side of a LET directive in all contexts.
9. Alternate statement:
   Some system variables can be modified by means of specific directives or commands. Directive DISPLAY_PROFILE (DP) displays the values of most system variables, in addition to some other general purpose information. These directives and commands are described in the *IOF Terminal User's Reference Manual*.

2.2.1 #: Terminal Line

Meaning:	Symbolic representation of the user's terminal.
Type:	char
Shape:	(1,1)
Length:	255
Default value:	none
Constraints:	none
Effect:	The formatted value assigned to # is displayed on the output device (IOF terminal, or SYSOUT or PRTFILE in batch). ***In IOF:*** when # is used in an expression, its value is the string of characters read from the terminal. The prompt for the string is #?. However, with a construct like: `LET A #` where the value read from the terminal is assigned to variable A, the prompt issued is that of variable A, if any. ***In Batch:*** # may not be used in the context `'LET A #'`.
Level of assignment:	any
Alternate statement:	none

2.2.2 #AUTOLF: Auto Line Feed

Meaning:	If 1: line feed is generated by pressing return or transmit If 0: line feed is generated by the system.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	As specified in the network generation.
Constraints:	none
Effect:	#AUTOLF is assigned a value when the NETGEN option does not match the actual terminal setting.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.3 #BINLIB: Binary Input Libraries

Meaning:	Current binary library search path. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape (see Note 1):	(0,3) or (0,15)
Length:	190
Default value:	() empty
Constraints:	The shape depends on the configuration parameter GCL15BIN.
Effect:	Modifying #BINLIB redefines the search rules for accessing command definitions and object forms.
Level of assignment:	system level
Alternate statement:	MWINLIB

NOTES:

1. For more information about shape, see Paragraph *"Domains, Libraries, and Search Rules"*.
2. #BINLIB can support up to 3 or 15 Binlibs depending on the GCL15BIN parameter.

2.2.4 #BLIB: Binary Output Library

Meaning:	Current binary output library. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,1)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #BLIB redefines the default binary output library used by processors such as MAINTAIN_LIBRARY, MAINTAIN_COMMAND and MAINTAIN_FORM. It also affects the search rules for object forms.
Level of assignment:	system level
Alternate statement:	MWLIB

2.2.5 #BRKPMODE: Break Processing Mode

For more information, see paragraph *"Break Processing"*.

2.2.6 #BRK: Break

For more information, see paragraph *"Break Processing"*.

2.2.7 #CC: Continuation Character

Meaning:	Character indicating that input continues on the next line.
Type:	char
Shape:	(1,1)
Length:	1
Default value:	- (minus)
Constraints:	none
Effect:	Any character can be assigned to #CC to become the new symbol for denoting input continuation when entering a multi-line command in line format. If the value blank ( ) is assigned to #CC, then it is as if there were no continuation character. See the *IOF Terminal User's Reference Manual*.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.8 #CINLIB: Compile Unit (CU) Input Libraries

Meaning:	Current CU (Compile Unit) search path. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,3)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #CINLIB redefines search path in which LINK_PG searches for CUs to link into a Load or Sharable Module.
Level of assignment:	system level
Alternate statement:	MWINLIB

2.2.9 #CLIB: Compile Unit (CU) Output Library

Meaning:	Current CU (Compile Unit) output library. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,1)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #CLIB redefines the default CU library in which compilers store their outputs, and from which LINK_PG retrieves CU to link into a Load or Sharable Module.
Level of assignment:	system level
Alternate statement:	MWLIB

2.2.10 #CSET: character set

Meaning:	Character set (0 through 7) for coding terminal characters.
Type:	dec
Shape:	(1,1)
Length:	1
Default value:	1
Constraints:	- `0` for code C101 (EBCDIC) - `1` for PLW (Pluri Lingual West) or C127 - `2` for APL - `3` for code C114 (Arabic character set) - `4` for code C118 (Greek character set) - `5` for code C113 (Cyrillic character set) - `6` for code C094 (Chinese character set) - `7` for PLE (Pluri Lingual East).
Effect:	Refer to the IOF Programmer's Manual.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.11 #DEBUG: Debug GCL Procedures

Meaning:	Displays each line executed in unlocked GCL procedures.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	0
Constraints:	none
Effect:	#DEBUG=1: The line executed is displayed with evaluated variables prefixed by procedure name and line number. In batch, a missing procedure name is replaced by the RON.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.12 #DI: Directive Identifier

Meaning:	String of characters used to force directives. See the *IOF Terminal User's Reference Manual*.
Type:	char
Shape:	(1,1)
Length:	3
Default value:	$$
Constraints:	none
Effect:	Assigning a new value to #DI redefines the prefix used for forcing directives. Assigning an empty string ('') means that directives cannot be forced.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.13 #EDITCTL: Text Editor Controls

Meaning:	Characters to be used as substitutes for ^ $ and [in Text Editor, Full Screen Editor and SCANNER.
Type:	char
Shape:	(3,3)
Length:	1
Default value:	(^ $ [)
Constraints:	Each element must be exactly length 1 (no empty string allowed).
Effect:	#EDITCTL redefines the three editing control characters: ^ for the first line or beginning of line, in a regular expression $ for the last line, or end of line, in a regular expression [for introducing an escape sequence.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.14 #ENVT: Working Environment

Meaning:	Current working environment. See Section the *IOF Terminal User's Reference Manual*.
Type:	char
Shape:	(1,1)
Length:	12
Default value:	Project's default environment as set in the system catalog.
Constraints:	Value must be either a name or blanks; name must be that of an environment that is accessible to the project.
Effect:	See Section the *IOF Terminal User's Reference Manual*.
Level of assignment:	system level
Alternate statement:	MWENVT

2.2.15 #EXPTABS: Expand Tabulations

Meaning:	Expands the tabulation characters into spaces.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	1
Constraints:	none
Effect:	When #EXPTABS is 1, all tabulation characters keyed in are replaced by an appropriate number of spaces, as defined by the #TABS system variable. When #EXPTABS is 0, tabulation characters are transmitted as they are.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.16 #FORMLANG: MAINTAIN_FORM Generation Language

Meaning:	Programming language for which MAINTAIN_FORM generates data structures or declaratives. See IOF Programmer's Manual.
Type:	name
Shape:	(1,1)
Length:	12
Default value:	COBOL
Constraints:	Acceptable values are COBOL, GPL, RPG2 and CL (for C Language).
Effect:	See IOF Programmer's Manual.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.17 #GCLFORM: GCL Format

Meaning:	GCL format (line or free).
Type:	dec
Shape:	(1,1)
Length:	1
Default value:	0
Constraints:	Values are 0 and 1 for free format and line format respectively. No other values are allowed.
Effect:	If #GCLFORM is 0: all commands must end with a semicolon. If #GCLFORM is 1: the end of line acts as a command terminator.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.18 #INVCHAR: Invalid Character Representation

Meaning:	Character to be used to represent characters that cannot be displayed on the terminal such as control codes.
Type:	char
Shape:	(1,1)
Length:	1
Default value:	. (period)
Constraints:	none.
Effect:	Modifying #INVCHAR results in redefining the representation for invalid characters. If #INVCHAR is assigned the empty string (''), invalid characters are left unchanged.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.19 #JCLCOMP: JCL Compatibility Mode

Meaning:	Compatibility option with JCL (Job Control Language).
Type:	dec
Shape:	(1,1)
Length:	1
Default value:	0
Constraints:	Values may only be 0 or 2.
Effect:	If #JCLCOMP is 0: only GCL is accepted as the command language. If #JCLCOMP is 2: only JCL is accepted at system level; certain facilities are unavailable.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.20 #JOBLANG: Default Command Language for Batch Jobs

Meaning:	Default Command Language used in the commands EJR and RUN_JOB when submitting Batch Jobs.
Type:	name
Shape:	(1,1)
Length:	3
Default value:	none
Constraints:	Values may only be JCL or GCL
Effect:	When executing EJR or RUN_JOB, the contents of the variable #JOBLANG, if any, is used as default value of the JOBLANG parameter of EJR or RUN_JOB.

2.2.21 #LANG: National Language

Meaning:	Terminal user's national language.
Type:	dec
Shape:	(1,1)
Length:	1
Default value:	0
Constraints:	none
Effect:	#LANG is used to specify the user's national language in which, for example, Help texts are displayed. Value 0 always refers to the English language. Other values, in the range 1 through 9, have installation-defined meanings.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.22 #LINLIB: Load Module (LM) Input Libraries

Meaning:	Current Load Module search path. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,3)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #LINLIB redefines the Load Module search path used by MAINTAIN_LIBRARY.
Level of assignment:	system level
Alternate statement:	MWINLIB

2.2.23 #LLIB: Load Module (LM) Output Library

Meaning:	Current Load Module output library. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,1)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #LLIB redefines the default Load Module library, where LINK_PG stores executable modules, and from which these modules are loaded for execution by EXEC_PG.
Level of assignment:	system level
Alternate statement:	MWLIB

2.2.24 #MENU: Dialog Through Menus and Prompts

Meaning:	Whether menus and prompts are to be displayed.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	1
Constraints:	none
Effect:	If #MENU is 1: menus and prompts are displayed on explicit request or when an invalid command is entered. If #MENU is 0 and #NOVICE is 0: menus and prompts are never displayed, even if explicitly requested.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.25 #NO: negative Replies

Meaning:	"no" in reply to questions from the system.
Type:	name
Shape:	(1,3)
Length:	8
Default value:	(NO, N, 0)
Constraints:	none
Effect:	System variable #NO accepts up to three names that are taken to mean "NO" in reply to questions such as "Do you want to save ?".
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.26 #NOVICE: Novice Mode

Meaning:	Specify whether the terminal user is a novice user or not.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	1
Constraints:	none
Effect:	#NOVICE=1 and terminal operating in formatted mode: displays a menu when the system expects a command input. #NOVICE=0 or terminal does not support formatted mode: commands are prompted by single letter followed by colon: S: for system level C: for MAINTAIN_LIBRARY commands.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.27 #PAGEMODE: Page Mode

Meaning:	Defines action when the screen is filled or at the end of page.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	terminal-dependent
Constraints:	none
Effect:	#PAGEMODE=1: For a video terminal, +++ appears when display fills the bottom of the screen to prompt the user. For a printer, performs form feed at the end of each page. #PAGEMODE=0 and #ROLL=0: Automatically skips to the top of screen; applies only for terminals connected over low speed communication lines. #PAGEMODE=0 and #ROLL=1: Scrolls up data by one line.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.28 #PAGETOP: Page Top

Meaning:	In interactive mode, specifies whether or not a "top of page" will be generated by many of the GCL commands of files or volumes management before, the display of the profile contents, or not.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	1
Constraints:	none
Effect:	#PAGETOP=1: The jump is automatically generated on the screen before the display of the profile contents; this was the normal behavior provided by many of the IOF files and volumes management commands in the previous releases of GCOS 7. #PAGETOP=0: No jump is generated; on the screen, the profile contents are displayed directly after the "transmit" which has launched the command.
Level of assignment:	any
Alternative statement:	MODIFY_PROFILE (MDP)

2.2.29 #PL: Page Length

Meaning:	Number of lines per screen or page (for a printer).
Type:	dec
Shape:	(1,1)
Length:	3
Default value:	terminal-dependent
Constraints:	10 < page length <= terminal's actual page length.
Effect:	#PL specifies the number of lines before form feed or some other action is performed. See #PAGEMODE and #ROLL.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.30 #PROMPT: Prompting on the Terminal

Meaning:	Whether prompts are used when an error message is displayed in IOF.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	1
Constraints:	none
Effect:	#PROMPT=1: The prompt of the command is displayed when an error occurs. #PROMPT=0: The error is displayed without prompting the erroneous command.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MP)

2.2.31 #PRTLIB: Printout Library

Meaning:	Current listing output library. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,1)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #PRTLIB redefines the default listing output library where compilers and other processors store their reports.
Level of assignment:	system level
Alternate statement:	MPRTLIB

2.2.32 #PW: Printing Width

Meaning:	Maximum number of characters per line.
Type:	dec
Shape:	(1,1)
Length:	3
Default value:	terminal-dependent
Constraints:	Values must be greater than 39 and less than or equal to the physical width of the terminal.
Effect:	Modifying #PW redefines the portion of the screen in which lines are displayed. Overlength lines are folded and an appropriate character (-) indicates that folding has occurred.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.33 #ROLL: Roll Mode

Meaning:	Selects roll mode (#ROLL=1) or wrap mode (#ROLL=0).
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	As specified at NETGEN.
Constraints:	none
Effect:	For VIP7804, the system may select roll/wrap, so setting #ROLL also sets the terminal mode. #ROLL is used when NETGEN options do not match the actual terminal setting.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.34 #SEV: Severity

Meaning:	Session severity level. See #STATUS.
Type:	dec
Shape:	(1,1)
Length:	1
Default value:	0
Constraints:	Value may only be 0, 1, 2, 3 or 4
Effect:	Modifying #SEV alters the session severity level and the system variable #STATUS:

#SEV		#STATUS	Meaning
0	-->	0	normal completion
1	-->	100	warning
2	-->	1000	error
3	-->	10000	severe error
4	-->	20000	fatal error

Level of assignment:	any
Alternate statement:	none

2.2.35 #SINLIB: Source Language (SL) Input Libraries

Meaning:	Current Source Language library search path. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,3)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #SINLIB redefines SL library search path used by MAINTAIN_LIBRARY, the Text Editor and Full Screen Editor, and compilers and processors.
Level of assignment:	system level
Alternate statement:	MWINLIB

2.2.36 #SLIB: Source Language (SL) Output Library

Meaning:	Current value of the Source Language output library. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,1)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #SLIB redefines default SL library for storing the output of the Text Editor, Full Screen Editor and processors.
Level of assignment:	system level
Alternate statement:	MWLIB

2.2.37 #STATUS: Session Status

Meaning:	Session completion status. See #SEV.
Type:	dec
Shape:	(1,1)
Length:	5
Default value:	0
Constraints:	Value must be positive and less than 32768
Effect:	Modifying #STATUS alters the session completion status and the system variable #SEV:

#STATUS		#SEV	Meaning
0-99	-->	0	normal completion
100-999	-->	1	warning
1000-9999	-->	2	error
10000-19999	-->	3	severe error
20000-32767	-->	4	fatal error

Level of assignment:	any
Alternate statement:	none

2.2.38 #SWITCHES: Program Switches

Meaning:	Program switches that can be tested and set.
Type:	bool
Shape:	(32,32)
Length:	1
Default value:	all 0
Constraints:	The program switches are numbered from 0 to 31. The elements of #SWITCHES are numbered from 1 to 32. This shift must be taken into account in the programming. E.g.: `#ELEM(SWITCHES,4)` refers to switch number 3.
Effect:	#SWITCHES facilitates conveying 32 binary pieces of information to a program, for exchanging information between programs or between GCL and programs.
Level of assignment:	any
Alternate statement:	none

2.2.39 #TABS: Tabulation Stops

Meaning:	Positions of tabulation stops on the terminal, expressed in columns starting with 1 for the leftmost character position.
Type:	dec
Shape:	(0,16)
Length:	3
Default value:	() empty
Constraints:	Each element must be positive and less than or equal to the physical width of the terminal. #TABS cannot be assigned a value if the terminal has no facility for tabulation stops.
Effect:	Assigning values to #TABS redefines positions of tabulation stops. Values are also used for the number of spaces to be substituted for tabulation characters when #EXPTABS is 1.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.40 #TRACE: Trace GCL Procedure Execution

Meaning:	Request a trace of the GCL procedure calls.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	0
Constraints:	none
Effect:	#TRACE=1: all CALLs, CHAINs, VCALLs, VCHAINs and SCALLs executed in an unlocked procedure and referencing a locked procedure are listed with their parameters.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.41 #WD: Working Directory

Meaning:	Working directory. See the *IOF Terminal User's Reference Manual*.
Type:	char
Shape:	(0,1)
Length:	32
Default value:	Name of the user's project if it is also the name of a directory; otherwise empty `()`.
Constraints:	The value assigned must yield the name of a valid directory.
Effect:	See the *IOF Terminal User's Reference Manual*.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.42 #WSTATION: Working Station

Meaning:	Name of the logical station to which the user's session is attached by default. See the *IOF Terminal User's Reference Manual*.
Type:	char
Shape:	(1,1)
Length:	8
Default value:	Name of the station specified at logon or its default value.
Constraints:	Assigned value must be the name of an accessible station, or blank to denote the log-on default.
Effect:	See the *IOF Terminal User's Reference Manual*.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.43 #XINLIB: Sharable Module (SM) Input Libraries

Meaning:	Current Sharable Module search path. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,3)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #XINLIB redefines the Sharable Module search path used by MAINTAIN_LIBRARY.
Level of assignment:	system level
Alternate statement:	MWINLIB

2.2.44 #XLIB: Sharable Module (SM) Output Library

Meaning:	Current Sharable Module output library. See the *IOF Terminal User's Reference Manual*.
Type:	lib
Shape:	(0,1)
Length:	190
Default value:	() empty
Constraints:	none
Effect:	Modifying #XLIB redefines the default Sharable Module library where LINK_PG stores TPRs that are executable under TDS.
Level of assignment:	system level
Alternate statement:	MWLIB

2.2.45 #YES: Positive Replies

Meaning:	"yes" in reply to questions from the system.
Type:	name
Shape:	(1,3)
Length:	8
Default value:	(YES,Y,1)
Constraints:	none
Effect:	System variable #YES accepts up to three names that are taken to mean "YES" in reply to questions such as "Do you want to save?".
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.2.46 #ZOK: Busy Message

Meaning:	Request a busy prompt when the terminal receives no message.
Type:	bool
Shape:	(1,1)
Length:	1
Default value:	1
Constraints:	none
Effect:	#ZOK=1: the system issues a busy message every minute if it works for the terminal user but has no message to send. #ZOK=0: no busy message is issued.
Level of assignment:	any
Alternate statement:	MODIFY_PROFILE (MDP)

2.3 Builtin Functions

GCL builtin functions provide a means for obtaining information on the system status, for querying the execution environment, or for manipulating GCL objects.

The names of builtins all begin with the special character #. They may have zero, one, two, three, or more arguments, which may in turn be expressions. The arguments are enclosed in parentheses after the name of the builtin. For example:

```
#PLUS(3,5)                        -->      8
#PLUS(#TIMES(2,2),3)              -->      7
#PLUS(#TIMES(3,6),#TIMES(3,3))    -->     27
```

A builtin can be used in place of a literal or a variable provided that the result of the builtin is of a type acceptable in the context where it is used. For example, the builtin #PLUS gives a numeric result and therefore, can only be used where a numeric value is acceptable.

Conversely, the arguments of a builtin must also match certain type requirements. For example, the arguments of builtin #PLUS must be numeric.

NOTES:

1. The following builtins cannot be used in batch:

#KLN	#MASTER	#READ	#TERMID
#L	#QUERY	#READL	#TTYPE

2. These builtins may be read as any system variable in COBOL or GPL programs.

#BILLING	#LSYS	#RON	#USERID
#CPU	#MDAY	#TERMID	#WDAY
#DATE	#MODE	#TIME	#YDAY
#ELAPSED	#PROJECT	#TTYPE	#FW
#EXTDATE			

This Section lists builtin functions under the following information:

Purpose:	the object of the builtin and its effect
Arguments:	the types of the arguments, if any
Result:	the type of the result
Constraints:	constraints on the values of the arguments or on the result, if any
Examples:	annotated examples of how to use the builtin.

Most builtins operate on scalars, and produce scalar results. When arguments or results can be lists, "list" is fixed to the end of type. "Any" means any type. Limited implicit conversions are available, as with the LET directive.

Builtins are classified under the following functions:

- arithmetic	- terminal handling	- "Is it ?"
- relational	- list handling	- conversion
- boolean	- object management	- file handling.
- character handling	- context handling	

Acceptable values for arguments are:

`BOOL`	Boolean value 0 or 1 *Expressions of type bool; bool literals*
`CHAR`	Character string, quoted or unquoted *Any expression; any literal*
`DEC`	Decimal value, signed or unsigned *Expressions of types bool or dec; dec literals*
`FILE`	File for example, A.B.FILE or A.B.MYLIB..SF *Expressions of types file or lib; file literals*
`FSET`	Fileset for example, A.*.B *Expressions of types file, lib or fset; fset literals*
`HEXA`	Hexadecimal value for example, A23F *Expressions of type hexa; hexa literals*
`LIB`	Library for example, A.B.MYLIB *Expressions of type lib; lib literals*
`NAME`	Name for example, JOE, A-B, MYPG *Expressions of type name; name literals*
`OUTPUT`	Output for example, X123:2:1 *Expressions of type output; output literals*
`RFILE`	Remote File for example, $LYON:A.B.C *Expressions of types file, lib, rfile; rfile literals*
`STAR`	Star-Name for example, A*B *Expressions of types name or star; star literals*
`VOLUME`	Volume for example, VOL2:MS/D500 *Expressions of type volume; volume literals*

Other combinations will result in a TYPE ERROR.

2.3.1 Arithmetic Builtins

2.3.1.1 #ABS

Purpose:	Computes the absolute value of the argument.
Arguments:	dec
Result:	dec
Constraints:	none

Examples:	`#ABS(-3) --> 3`
	`#ABS(0)  --> 0`
	`#ABS(5)  --> 5`

2.3.1.2 #DIVIDE

Purpose:	Computes the integral quotient of the two arguments.
Arguments:	(dec, dec)
Result:	dec
Constraints:	The second argument must be non-zero.

Examples:	`#DIVIDE(18, 6)  -->  3`
	`#DIVIDE(-7, 2)  --> -3`
	`#DIVIDE(-4, -2) -->  2`

2.3.1.3 #MAX

Purpose:	Computes the arithmetically largest value among arguments.
Arguments:	(dec, dec[, dec]...)
Result:	dec
Constraints:	none

Examples:	`#MAX(6, 4)         -->  6`
	`#MAX(-7, 4, 6)     -->  6`
	`#MAX(-12, -15, -7) --> -7`

2.3.1.4 #MIN

Purpose:	Computes the arithmetically smallest value among arguments.
Arguments:	(dec, dec[, dec]...)
Result:	dec
Constraints:	none
Examples:	

```
#MIN(6, 4)          -->   4
#MIN( -7, 4, 6)     -->  -7
#MIN(-12, -15, -7) --> -15
```

2.3.1.5 #MINUS

Purpose:	Computes the value of arg1 minus arg2.
Arguments:	(dec, dec)
Result:	dec
Constraints:	The result may not exceed 32 digits in length.
Examples:	

```
#MINUS(82, 31) -->  51
#MINUS(-12, 7) --> -19
#MINUS(6, -3)  -->   9
```

2.3.1.6 #MOD

Purpose:	Computes value of arg1 modulo arg2 expressed as: res = arg1 - floor(arg1/arg2)*arg2
Arguments:	(dec, dec)
Result:	dec
Constraints:	The second argument must be non-zero.
Examples:	

```
#MOD(7, 2)   -->  1
#MOD(7, -2)  --> -1
#MOD(-7, 2)  -->  1
#MOD(-7, -2) --> -1
```

2.3.1.7 #PLUS

Purpose:	Computes the sum of all arguments.
Arguments:	(dec, dec[, dec]...)
Result:	dec
Constraints:	The result may not exceed 32 digits in length.
Examples:	

```
#PLUS(6, 3)         -->  9
#PLUS(12, -16, 3)   --> -1
#PLUS(6, 8, -1, -7) -->  6
```

2.3.1.8 #SIGNUM

Purpose:	Computes the arithmetic sign of the argument (-1 if negative, 0 if zero, 1 if positive).
Arguments:	dec
Result:	dec
Constraints:	none

Examples:	`#SIGNUM(-7) --> -1`
	`#SIGNUM(0)  -->  0`
	`#SIGNUM(6)  -->  1`

2.3.1.9 #TIMES

Purpose:	Computes the product of all arguments.
Arguments:	(dec, dec[, dec]...)
Result:	dec
Constraints:	The result may not exceed 32 digits in length.

Examples:	`#TIMES(2, 7)       -->  14`
	`#TIMES(6, 2, -3)   --> -36`
	`#TIMES(4, 2, 3, 5) --> 120`

2.3.2 Relational Builtins

Relational builtins compare the values of their two arguments and yield a boolean value denoting whether or not the relation is true.

Values to be compared are:
- either numeric being decimal or literal values denoting decimals, in which case, an arithmetic comparison is performed
- or non-numeric being character representations, in which case, a character comparison is performed in the EBCDIC collating sequence where the shorter of the two strings is padded with trailing spaces to equalize the lengths.

2.3.2.1 #EQ

Purpose:	Returns 1 if both arguments are equal: returns 0 otherwise.
Arguments:	(any, any)
Result:	bool
Constraints:	none
Examples:	see below

```
#EQ(02, +2)       --> 1
#EQ(AC, 'ac')     --> 0
#EQ('', ' ')      --> 1
#EQ('AB', 'AB  ') --> 1
```

2.3.2.2 #GE

Purpose:	Returns 1 if arg1 is greater than or equal to arg2; returns 0 otherwise.
Arguments:	(any, any)
Result:	bool
Constraints:	none
Examples:	see below

```
#GE(02, +2)   --> 1
#GE(ABC, DEF) --> 0
#GE(AA, A)    --> 1
```

2.3.2.3 #GT

Purpose:	Returns 1 if arg1 is greater than arg2; returns 0 otherwise.
Arguments:	(any, any)
Result:	bool
Constraints:	none

Examples:	`#GT(2, 02) --> 0`
	`#GT(A, AB) --> 0`
	`#GT(BC, B) --> 1`

2.3.2.4 #LE

Purpose:	Returns 1 if arg1 is less than or equal to arg2; returns 0 otherwise.
Arguments:	(any, any)
Result:	bool
Constraints:	none

Examples:	#LE(2, 02) --> 1
	`#LE(A, AB) --> 1`
	`#LE(BC, B) --> 0`

2.3.2.5 #LT

Purpose:	Returns 1 if arg1 is less than arg2; returns 0 otherwise.
Arguments:	(any, any)
Result:	bool
Constraints:	none

Examples:	`#LT(2, 02) --> 0`
	`#LT(A, AB) --> 1`
	`#LT(BC, B) --> 0`

2.3.2.6 #NE

Purpose:	Returns 1 if arg1 is not equal to arg2; returns 0 otherwise.
Arguments:	(any, any)
Result:	bool
Constraints:	none

Examples:	`#NE(2, 02)    --> 0`
	`#NE(A, AB)    --> 1`
	`#NE(AB, 'ab') --> 1`

2.3.3 Boolean Builtins

2.3.3.1 #AND

Purpose:	Performs *AND* on the arguments.
Arguments:	(bool, bool [, bool]...)
Result:	bool
Constraints:	none
Examples:	

%A	%B	#AND(%A, %B)
0	0	0
0	1	0
1	0	0
1	1	1

2.3.3.2 #NAND

Purpose:	Negates *AND* on the arguments.
Arguments:	(bool, bool)
Result:	bool
Constraints:	none
Examples:	

%A	%B	#NAND(%A,%B)
0	0	1
0	1	1
1	0	1
1	1	0

2.3.3.3 #NOR

Purpose:	Negates *OR* on the arguments.
Arguments:	(bool, bool)
Result:	bool
Constraints:	none
Examples:	

%A	%B	#NOR(%A, %B)
0	0	1
0	1	0
1	0	0
1	1	0

2.3.3.4 #NOT

Purpose:	Negates the argument.
Argument:	bool
Result:	bool
Constraints:	none
Examples:	

%A	**#NOT(%A)**
0	1
1	0

2.3.3.5 #OR

Purpose:	Performs *OR* on the arguments.
Arguments:	(bool, bool [, bool]...)
Result:	bool
Constraints:	none
Examples:	

%A	**%B**	**#OR(%A, %B)**
0	0	0
0	1	1
1	0	1
1	1	1

2.3.3.6 #XOR

Purpose:	Performs *exclusive OR* on the arguments.
Arguments:	(bool, bool)
Result:	bool
Constraints:	none
Examples:	

%A	**%B**	**#XOR(%A, %B)**
0	0	0
0	1	1
1	0	1
1	1	0

2.3.4 Character Handling Builtins

2.3.4.1 #CAT

Purpose:	Concatenates the arguments into a single string.
Arguments:	(char, char [, char]...)
Result:	char
Constraints:	The length of the resulting string must not exceed 255.

Examples:	`#CAT(A, BC)           --> ABC`
	`#CAT('Hello Boys', !) --> Hello Boys!`
	`#CAT(A, BC, DEF, G)   --> ABCDEFG`
	`#CAT(AB, '', CD)      --> ABCD`

2.3.4.2 #CTN

Purpose:	Returns 1 if arg1 contains arg2; returns 0 otherwise.
Arguments:	(char, char)
Result:	bool
Constraints:	none

Examples:	`#CTN(ABCDE, BC)  --> 1`
	`#CTN(ABCDE, BD)  --> 0`
	`#CTN(ABCDE, DEF) --> 0`

2.3.4.3 #INDEX

Purpose:	If arg2 is contained in arg1, returns the rank of the character at which it starts; otherwise returns 0.
Arguments:	(char, char)
Result:	dec
Constraints:	none

Examples:	`#INDEX(ABCDEF, DE) --> 4`
	`#INDEX(ABCDEF, AB) --> 1`
	`#INDEX(ABCDEF, AC) --> 0`

2.3.4.4 #LC

Purpose:	Converts the argument to lowercase letters.
Arguments:	char
Result:	char
Constraints:	none

Examples:	`#LC('HELLO BOYS') --> hello boys`
	`#LC(ABCDEF)       --> abcdef`
	`#LC('a*B+C')      --> a*b+c`

2.3.4.5 #MODIFY

Purpose:	Modifies the value of *arg1*, replacing the substring starting at position *arg2* of length *arg3*, by *arg4*. Returns 1 or 0 depending on whether modification is successful or not.
Arguments:	(char, dec, dec, char)
Result:	bool
Constraints:	*arg2* > 0 and *arg3* > 0 and *arg3* >= length of *arg4*. if *arg3* > *arg4*, *arg4* is padded with trailing spaces.

Examples:	If variable V is ABCDEFG in all following examples:
	`#MODIFY(%V, 2, 3, XYZ)->1:`
	`V` becomes `AXYZEFG`
	`#MODIFY(%V, 2, 3, X)  ->1:`
	`V` becomes `AX  EFG`
	`#MODIFY(%V, 3, 3, '') ->1:`
	`V` becomes `AB FG`
	`#MODIFY(%V, 8, 4, XX) ->0:`
	`V` unchanged

2.3.4.6 #QUOTE

Purpose:	Encloses a string between quotes, doubling each contained quote.
Arguments:	char
Result:	char
Constraints:	The length of the resulting string must not exceed 255.

Examples:	`#QUOTE(ABCD)       --> 'ABCD'`
	`#QUOTE('ABC''DEF') --> 'ABC''DEF'`
	`#QUOTE('AB CD')    --> 'AB CD'`

2.3.4.7 #SUBSTITUTE

Purpose:	Modifies the value of *arg1*, replacing the substring starting at position *arg2* of length *arg3*, by *arg4*. Returns 1 or 0 depending on whether or not modification is successful: – if *arg2*, *arg3* and the length of *arg4* are all non-zero and positive, the substring of *arg1* starting at position *arg2* of length *arg3*, is replaced by string *arg4*. – if *arg3* is 0, *arg4* is inserted into *arg1* just before the substring starting at position *arg2*. – if the length of *arg4* is 0 (''), the *arg3* characters starting at position *arg2* are suppressed in *arg1*.
Arguments:	(char, dec, dec, char)
Result:	bool
Constraints:	0 < *arg2* <= maximum length of arg1. *arg3* and *arg4* cannot be simultaneously equal to 0.

Examples:	If variable `V` is `ABCDE` in all following examples:
	`#SUBSTITUTE(%V,4,3,FGH) ->1:`
	`V` is now `ABCFGH`
	`#SUBSTITUTE(%V,1,0,'1 ')->1:`
	`V` is now `'1 ABCDE'`
	`#SUBSTITUTE(%V,2,0,X)   ->1:`
	`V` is now `AXBCDE`
	`#SUBSTITUTE(%V,2,1,'')  ->1:`
	`V` is now `ACDE`
	`#SUBSTITUTE(%V,10,1,'Z')->1:`
	`V` is now `'ABCDE Z'`

2.3.4.8 #SUBSTR

Purpose:	Extracts a substring from arg1; arg2 is the position of the first character in the substring; arg3 is the length of the substring.
Arguments:	(CHAR, DEC, [DEC])
Result:	CHAR
Constraints:	arg2 > 0; arg3 ≥ 0 If arg2> length of arg 1, result is string of arg3 spaces, where 0 ≤ arg3 ≤ 255. If arg3 is not specified, the default value for arg3 is: arg3 = length of arg1-arg2+1. Arg3 must be specified if arg2>length of arg1. *Resulting string is padded with trailing spaces if it exceeds the original one.*

Examples:	`#SUBSTR(ABCDEFG, 4, 2) --> DE`
	`#SUBSTR(ABCDEFG, 5)    --> EFG`
	`#SUBSTR(ABCDEFG, 6, 6) --> FG + 4` spaces
	`#SUBSTR(ABCDEFG, 5, 0) -->` empty string `('')`

2.3.4.9 #UC

Purpose:	Converts the argument to uppercase letters.
Arguments:	char
Result:	char
Constraints:	none

Examples:	`#UC('abcdef')     --> ABCDEF`
	`#UC('a+b-C')      --> A+B-C`
	`#UC('Hello Boys') --> HELLO BOYS`

2.3.4.10 #UNQUOTE

Purpose:	If the value of the argument is a quoted string, returns its unquoted value; otherwise returns the argument unchanged.
Arguments:	char
Result:	char
Constraints:	none

Examples:	`#UNQUOTE('''ABC''')         --> ABC`
	`#UNQUOTE('''''''ABC''''''') --> 'ABC'`
	`#UNQUOTE(ABC)               --> ABC`

2.3.4.11 #VERIFY

Purpose:	Checks if the characters in arg1 are in the set defined by arg2: – Returns 1 if all arg1 characters are in arg2, or if arg1=0 – Returns 0 if at least one character of arg1 is not in arg2, or if arg2=0.
Arguments:	(char, char)
Result:	bool
Constraints:	none

Examples:	`LET # #VERIFY(AB,XACDBE) --> 1`
	`LET # #VERIFY('.X',XYZ)  --> 0`
	`LET # #VERIFY('',ABCD)   --> 1`
	`LET # #VERIFY(ABCD,'')   --> 0`

2.3.5 Terminal Handling Builtins

All Terminal Handling builtins are applicable only in interactive mode. *They are meaningless if specified in batch.*

2.3.5.1 #KLN

Purpose:	Keeps the line busy, displaying arg2 every arg1 seconds, until a break is entered. Result is always 1.
Arguments:	(dec, char)
Result:	bool
Constraints:	arg1 > 0

Examples:	`#KLN(10, 'DO NOT DISTURB PLEASE')`
	`#KLN(60, '')`

2.3.5.2 #L

Purpose:	Reads elements of a list from the terminal using arg2 as a prompt and assign them to the variable whose name is arg1. Returns 1 if the assignment is successful.
Arguments:	(name, char)
Result:	bool
Constraints:	Conditions for data entry are: – the user is prompted to supply data by the prompt defined in arg2; data can be entered in several lines. – a semicolon marks the end of the data keyed in; elements in a list are separated by commas or spaces; a break or a slash in an input line cancels the builtin. – slashes, spaces, commas and semicolons must be protected if supplied as data. – data supplied must be consistent with the definition of the variable arg1, namely type, length, values, and minimum and maximum number of elements. – if an error is detected, an appropriate message is issued and the data must be reentered from the beginning. – an empty list is assigned with just a semicolon; this is only possible when the first element of NUMVAL is 0.

Examples:	`GLOBAL D DEC 8 NUMVAL=(0,8);`
	`GLOBAL B BOOL;`
	`LET B #L(D,'ENTER D:');`
	`ENTER D:  6,12,14,5,7`
	`-:  8,9;`
	`D` now contains `(6, 12, 14, 5, 78, 9);` `B` is `1`.
	`LET B #L(D,'ENTER D:');`
	`ENTER D:  ;` `D` is now unassigned and `B` is `1`.

2.3.5.3 #MASTER

Purpose:	Returns 0 if the current session is attached to a slave terminal; returns 1 if attached to a master terminal.
Arguments:	none
Result:	bool
Constraints:	none

2.3.5.4 #QUERY

Purpose:	Asks the user the question in the argument; returns 1 if the reply is yes. *See system variable #YES in Paragraph 2.2.40.*
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#QUERY('DO YOU WANT TO CONTINUE?')`
	`#QUERY(CONTINUE?)`

2.3.5.5 #READ

Purpose:	Reads input from the terminal or from a subfile, if an AI directive is active, using the argument as a prompt, and returns its value.
Arguments:	char
Result:	char
Constraints:	The string entered must not exceed 255 characters in length.

Examples:	`#READ(OPTION?)`
	`#READ('What is your name ?')`

2.3.5.6 #READL

Purpose:	as for #L, but with data supplied from the active input stream. If the data is wrong, the stream aborts.
Arguments:	(name, char)
Result:	bool
Constraints:	see #L

2.3.6 List Handling Builtins

The builtins described in previously all operate on single elements or scalars. *The builtins described here operate on lists.*

A list is a finite set of elements of the same type:

- when the list has one element, it is a scalar
- when it has no elements, it is empty or uninitialized
- when it has more than one element, it is a true list.

A list is declared:

- by the NUMVAL parameter specifying a range bounded by a minimum and a maximum number of elements, each up to 64
- in the GLOBAL, LOCAL and KWD commands defining the name of a variable.

EXAMPLE:

`GLOBAL V DEC 3 NUMVAL=(2,5)`	declares V as a list of 2 to 5 elements, each element having a length of 3 decimals
`LET V (3,5,7)`	V now contains three elements
`LET V (12,37)`	V now contains two elements
`LET V 12`	
`LET V (1,2,3,4,5,6)`	erroneous because the complete number of elements has not been assigned

❑

2.3.6.1 #ELEM

Purpose:	Extracts the element whose position is arg2 in the arg1 list.
Arguments:	(any-list, dec)
Result:	same type as arg1
Constraints:	0 < arg2 <=number of elements in arg1.

Examples:	`LET V (6, 7, 12, 18)`
	`#ELEM(%V, 4) --> 18`
	`#ELEM(%V, 2) -->  7`
	`#ELEM(%V, 1) -->  6`

2.3.6.2 #FMT

Purpose:	Returns the string of characters that is the formatted representation for its argument.
Arguments:	any-list
Result:	char
Constraints:	The result may not exceed 255 characters in length.

Examples:	`LET V (6, 7, 8, 9, 10)`
	`#FMT(%V) --> (6 7 8 9 10)`
	`LET X ('Hello Boys', 'John')`
	`#FMT(%X) --> ('Hello Boys' 'John')`

2.3.6.3 #INDEX_SET

Purpose:	Returns a list of the first n consecutive integers.
Arguments:	dec
Result:	dec-list
Constraints:	0<argument<=64

Examples:	`#INDEX_SET(3)  --> (1, 2, 3)`
	`#INDEX_SET(6)  --> (1, 2, 3, 4, 5, 6)`
	`LET V 5`
	`#INDEX_SET(%V) --> (1, 2, 3, 4, 5)`

2.3.6.4 #LCOUNT

Purpose:	Returns the number of times the element arg2 is equal to an element of arg1 list. The comparison rules are the same as those for relational builtins.
Arguments:	(any-list, any)
Result:	dec (0<=result<=64)
Constraints:	none

Examples:	`LET V (A B C D A B A)`
	`#LCOUNT(%V,A) --> 3`
	`#LCOUNT(%V,D) --> 1`
	`#LCOUNT(%V,Z) --> 0`

2.3.6.5 #LCTN

Purpose:	Returns 1 if the arg1 list has an element equal to arg2; 0 if otherwise. Comparison rules are as for relational builtins.
Arguments:	(any-list, any)
Result:	bool
Constraints:	none

Examples:	`LET V (2, 3, 5, 7, 9)`
	`#LCTN(%V, 3) --> 1`
	`#LCTN(%V, 4) --> 0`

2.3.6.6 #LINDEX

Purpose:	If the first argument list contains at least one element equal to arg2, returns the rank of its first occurrence; otherwise returns 0. Comparison rules are as for relational builtins.
Arguments:	(any-list, any)
Result:	dec
Constraints:	none

Examples:	`LET V (2, 4, 6, 8, 10)`
	`#LINDEX(%V, 6)  --> 3`
	`#LINDEX(%V, 10) --> 5`
	`#LINDEX(%V, 3)  --> 0`

2.3.6.7 #LLENGTH

Purpose:	Returns current number of elements in the argument list.
Arguments:	any-list
Result:	dec
Constraints:	none

Examples:	`LET V (2, 4, 6, 8)`
	`#LLENGTH(%V) --> 4`
	`LET V 3`
	`#LLENGTH(%V) --> 1`
	`LET V ()`
	`#LLENGTH(%V) --> 0`

2.3.6.8 #MAXLLENGTH

Purpose:	Returns the maximum elements of the list. If argument is an expression or literal value, returns the current number.
Arguments:	any-list
Result:	dec
Constraints:	none
Examples:	`GLOBAL V DEC 3 NUMVAL=(2, 5)`

```
GLOBAL V DEC 3 NUMVAL=(2, 5)
#MAXLLENGTH(%V)             --> 5
#MAXLLENGTH(3)              --> 1
#MAXLLENGTH(#INDEX_SET(3)) --> 3
```

2.3.6.9 #MINLLENGTH

Purpose:	Returns minimum number of elements of the list. If argument is an expression or literal value, returns the current number.
Arguments:	any-list
Result:	dec
Constraints:	none
Examples:	

```
GLOBAL V DEC 3 NUMVAL=(2, 5)
#MINLLENGTH(%V)             --> 2
#MINLLENGTH(6)              --> 1
#MINLLENGTH(#INDEX_SET(6)) --> 6
```

2.3.6.10 #REPLACE

Purpose:	Replaces the arg2th element of the list whose name is arg1 with arg3. Returns 1 if successfully replaced, 0 if otherwise.
Arguments:	(name, dec, any)
Result:	bool
Constraints:	arg3 must be of type and value compatible with arg1.
Examples:	

```
GLOBAL V DEC 3 NUMVAL=(2, 5)
LET V (1, 2, 3, 4, 5)
#REPLACE(V, 4, 7) ->1: V is now
(1, 2, 3, 7, 5)
#REPLACE(V, 6, 8) ->0: V unchanged
```

2.3.6.11 #STRING

Purpose:	Concatenates all the elements of the argument lists in new list.
Arguments:	(any-list, any-list [any-list]...)
Result:	a list of the same type as the arguments.
Constraints:	All arguments must be of the same type; the resulting list may not have more than 32 elements.

Examples:

```
GLOBAL V1 DEC 3 NUMVAL=(2,5)
GLOBAL V2 DEC 4 NUMVAL=(0,4)
LET V1 (1,2,3)
LET V2 (22,23,24,25)
#STRING(%V1,%V2)    ->(1,2,3,22,23,
                       24,25)
#STRING(%V1,%V2,%V1)->(1,2,3,22,23,
                       24,25,1,2,3)
LET V2 ()
#STRING(%V1,%V2,%V1)->(1,2,3,1,2,3)
```

2.3.6.12 #STRIP

Purpose:	Deletes the arg2 first (if arg2 > 0) or arg2 last (if arg2 < 0) elements of the arg1 list.
Arguments:	(any-list, dec)
Result:	a list with the same type as arg1.
Constraints:	none

Examples:

```
GLOBAL V DEC 3 NUMVAL=(2, 5)
LET V (2, 4, 6, 8, 10)
#STRIP(%V, 2)  --> (6, 8, 10)
#STRIP(%V, -3) --> (2, 4)
#STRIP(%V, 0)  --> (2, 4, 6, 8, 10)
#STRIP(%V, 6)  --> ()
```

2.3.7 Object Management Builtins

2.3.7.1 #CHECKSTAR

Purpose:	Returns 1 if arg2 name matches arg1 star-name; 0 if not.
Arguments:	(star, name)
Result:	bool
Constraints:	none

Examples:	`#CHECKSTAR(A*B, AXXXB)   --> 1`
	`#CHECKSTAR(A*B, ABD)     --> 0`
	`#CHECKSTAR(A*$>AI, AJBC) --> 1`
	`#CHECKSTAR(A*$>AI, AHX)  --> 0`

2.3.7.2 #DROP

Purpose:	Deletes the existing global variable whose name is the argument. Returns 1 if delete is successful; 0 if otherwise.
Arguments:	name
Result:	bool
Constraints:	none

Examples:

`#DROP(V1)`	deletes variable	`V1 -> 1`	if it existed
		`0`	otherwise
`#DROP(%NAME`	deletes variable	`V2 -> 1`	if it existed
		`0`	otherwise

2.3.7.3 #DROPGB

Purpose:	Deletes all global variables and returns the number deleted.
Arguments:	none
Result:	dec
Constraints:	none

2.3.7.4 #EXIST

Purpose:	Returns 1 if the variable whose name is the argument both exists and has a value (that is, has one or more elements).
Arguments:	name
Result:	bool
Constraints:	none

Examples:

```
#EXIST(V)      --> 1 if V both exists and has
                     a value
               --> 0 if V non-existent or
                     uninitialized
LET NAME V2
#EXIST(%NAME) --> 1 if V2 both exists and has
                     a value
               --> 0 if V2 non-existent or
                     uninitialized
```

2.3.7.5 #LENGTH

Purpose:	Returns the length in number of characters of the argument.
Arguments:	any
Result:	dec
Constraints:	none

Examples:

```
#LENGTH(1234)         -->  4
#LENGTH('Hello Boys') --> 10
#LENGTH('')           -->  0
```

2.3.7.6 #LISTGB

Purpose:	Lists the names and characteristics of all existing global variables. Returns the number of global variables.
Arguments:	none
Result:	dec
Constraints:	none

2.3.7.7 #MLENGTH

Purpose:	Returns the maximum length of the argument. If the argument is an expression or a literal, returns its current value.
Arguments:	any
Result:	dec
Constraints:	none
Examples:	see below

```
GLOBAL C CHAR 20
LET C 'Hello Boys'
#MLENGTH(%C)   --> 20
#MLENGTH(1234) -->  4
```

2.3.7.8 #NEXIST

Purpose:	Returns 1 if the variable whose name is the argument either does not exist or is empty.
Arguments:	name
Result:	bool
Constraints:	none
Examples:	see below

```
#NEXIST(V)     --> 1  if V non-existent or empty
               --> 0  if V both exists and has
                      a value
LET NAME V2
#NEXIST(%NAME)--> 0  if V2 both exists and has
                      a value
               --> 1  if V2 non-existent or empty
```

2.3.7.9 #VALUE

Purpose:	Returns the value of the variable whose name is the argument.
Arguments:	name
Result:	type undefined (depends on usage context)
Constraints:	none
Examples:	see below

```
LET V 1234
#VALUE(V)     --> 1234
LET NAME V
#VALUE(NAME)  --> V
#VALUE(%NAME) --> 1234
```

2.3.8 Context Handling Builtins

All Context Handling builtins:

- may be read as any system variable in COBOL or GPL programs. *The exceptions are #DOMAINID and #WAIT which do not apply.*
- can be used in both interactive and batch modes. *The exceptions are #TERMID and #TTYPE which cannot be used in batch.*

2.3.8.1 #BILLING

Purpose:	Returns the billing of the current session.
Arguments:	none
Result:	name
Constraints:	none

2.3.8.2 #CPU

Purpose:	Returns the central processor time consumed in milliseconds since the beginning of the session.
Arguments:	none
Result:	dec
Constraints:	none

2.3.8.3 #DATE

Purpose:	Returns the date as a character string in the form yy/mm/dd.
Arguments:	none
Result:	char
Constraints:	none

2.3.8.4 #DOMAINID

Purpose:	Returns the name of the current GCL domain.
Arguments:	none
Result:	name of the current GCL domain
Constraints:	none
Example:	`if in MAINTAIN_LIBRARY SL at "C:" level #DOMAINID --> LIBMAINT_SL`

2.3.8.5 #ELAPSED

Purpose:	Returns the time elapsed since the beginning of the session expressed in milliseconds.
Arguments:	none
Result:	dec
Constraints:	none

2.3.8.6 #EXTDATE

Purpose:	Returns the date as a character string in the form yyyy/mm/dd.
Arguments:	none
Result:	char
Constraints:	none

2.3.8.7 #FW

Purpose:	Returns the fiscal week corresponding to a given date.
Arguments:	char10 = date in the format [YY]YY/MM/DD or [YY]YY.MM.DD
Result:	dec4 = fiscal week in the format YYWW
Constraints:	none
Examples:	if today is 15 December 1993

```
#FW (#DATE)       --> 9350
#FW (#EXTDATE)    --> 9350
#FW (94/01/02)    --> 9352
#FW (2000/02/29) --> 0009
```

2.3.8.8 #LSYS

Purpose:	Returns the local System Name to which the user is connected either after the log-on mechanism or after the CONNECT_APPLICATION command.
Arguments:	none
Result:	char8
Constraints:	none
Examples:	

```
$*$CN IOFBP50
LET # #LSYS --> BP50
CN BP60
LET # #LSYS --> BP60
```

2.3.8.9 #MDAY

Purpose:	Returns the rank of the current day in the month.
Arguments:	none
Result:	dec
Constraints:	none

Example:	`if today is 24 May 1992`
	`#MDAY --> 24`

2.3.8.10 #MODE

Purpose:	Returns 1 if used in an interactive session; returns 0 otherwise.
Arguments:	none
Result:	bool
Constraints:	none

2.3.8.11 #PROJECT

Purpose:	Returns the project of the current session.
Arguments:	none
Result:	name
Constraints:	none

2.3.8.12 #RON

Purpose:	Returns RON (Run Occurrence Number) of current session.
Arguments:	none
Result:	dec
Constraints:	none

2.3.8.13 #TERMID

Purpose:	Returns the identification of the terminal.
Arguments:	none
Result:	char12 (Form: NODE char4 TERMINAL-NAME char8)
Constraints:	The result is meaningless if terminal is switched.
Example:	`#TERMID --> BP06V6C3`

2.3.8.14 #TIME

Purpose:	Returns time of day as character string in format hh:mm:ss.
Arguments:	none
Result:	char
Constraints:	none

2.3.8.15 #TTYPE

Purpose:	Returns the terminal type.
Arguments:	none
Result:	name
Constraints:	none

2.3.8.16 #USERID

Purpose:	Returns the user's name.
Arguments:	none
Result:	name
Constraints:	none

2.3.8.17 #WAIT

Purpose:	Wait arg1 seconds before resuming processing. Returns 1 if no interrupt (break) occurs during arg1 seconds; 0 if an interrupt occurs.
Arguments:	dec
Result:	bool
Constraints:	arg1 > 0

2.3.8.18 #WDAY

Purpose:	Returns rank of day in week (Monday is 1; Sunday is 7).
Arguments:	none
Result:	dec
Constraints:	none
Example:	`if today is Thursday 17 September 1992` `#WDAY --> 4`

2.3.8.19 #YDAY

Purpose:	Returns the rank of the current day in the year.
Arguments:	none
Result:	dec
Constraints:	none

Example:	`if today is 28 December 1992`
	`#YDAY --> 363`

2.3.9 "Is it?" Builtins

The "Is it?" Builtins check only the value of the argument, which must be converted if necessary to the appropriate type before use.

EXAMPLE:

```
GB C CHAR 10;
LET C 31;
LET # #ISITDEC(%C);  --> 1
LET # #TIMES(3,#CVDEC(%C));
```

❑

2.3.9.1 #ISITBOOL

Purpose:	Returns 1 if the argument is boolean; 0 if otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITBOOL(0)   --> 1`
	`#ISITBOOL(ABC) --> 0`

2.3.9.2 #ISITDATE

Purpose:	Returns 1 if the date given in the format [YY]YY/MM/DD or [YY]YY.MM.DD is correct; 0 if otherwise.
Arguments:	char10
Result:	bool
Constraints:	none

Examples:	`#ISITDATE (93/12/10)   --> 1`
	`#ISITDATE (93/02/29)   --> 0`
	`#ISITDATE (2001/12/31) --> 1`
	`#ISITDATE (1993/0A/01) --> 0`

2.3.9.3 #ISITDEC

Purpose:	Returns 1 if the argument is numeric; returns 0 otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITDEC(ABC)   --> 0`
	`#ISITDEC(-1234) --> 1`

2.3.9.4 #ISITFILE

Purpose:	Returns 1 if the argument is a file; returns 0 otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITFILE(a/b/c)    --> 0`
	`#ISITFILE(A.B.C..S) --> 1`

2.3.9.5 #ISITFSET

Purpose:	Returns 1 if the argument is a fileset; returns 0 otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITFSET(A.**) --> 1`
	`#ISITFSET(A/B)  --> 0`

2.3.9.6 #ISITHEXA

Purpose:	Returns 1 if the argument is hexadecimal; 0 if otherwise.
Arguments:	char
Result:	bool
Constraints:	length of the argument must be even and < or = 8.

Examples:	`#ISITHEXA(AB2F) --> 1`
	`#ISITHEXA(123H) --> 0`

2.3.9.7 #ISITLIB

Purpose:	Returns 1 if the argument is a library; returns 0 otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITLIB(A.B.C)    --> 1`
	`#ISITLIB(A.B.C..D) --> 0`

2.3.9.8 #ISITNAME

Purpose:	Returns 1 if the argument is a name; returns 0 otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITNAME(ABCD)    --> 1`
	`#ISITNAME(A.B.C)   --> 0`

2.3.9.9 #ISITOUTPUT

Purpose:	Returns 1 if the argument is an output; returns 0 otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITOUTPUT(X234:2:3) --> 1`
	`#ISITOUTPUT(X22/7)    --> 0`

2.3.9.10 #ISITRFILE

Purpose:	Returns 1 if the argument is a remote file; 0 if otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITRFILE($HERE:A.B.C) --> 1`
	`#ISITRFILE(X234/789)    --> 0`

2.3.9.11 #ISITSTAR

Purpose:	Returns 1 if the argument is a star-name; 0 if otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITSTAR(AB*CD$>ABK) --> 1`
	`#ISITSTAR(A.B.C)      --> 0`

2.3.9.12 #ISITTIME

Purpose:	Returns 1 if the time given in the format HH.MM or HH:MM (H for hour, M for minutes) is correct; 0 if otherwise.
Arguments:	char-5
Result:	bool
Constraints:	none

Examples:	`#ISITTIME(09.24) --> 1`
	`#ISITTIME(24:00) --> 0`

2.3.9.13 #ISITVOLUME

Purpose:	Returns 1 if the argument is a volume; 0 if otherwise.
Arguments:	char
Result:	bool
Constraints:	none

Examples:	`#ISITVOLUME(A.B.C)        --> 0`
	`#ISITVOLUME(K181:MS/D500) --> 1`

2.3.10 Conversion Builtins

2.3.10.1 #BINTODEC

Purpose:	Converts the argument to type dec.
Arguments:	char4
Result:	dec
Constraints:	None.

2.3.10.2 #BYTE

Purpose:	Converts the argument to its binary value; the result is a character string of length 1.
Arguments:	dec
Result:	char1
Constraints:	Ranges from 0 through 255. Binary values may be passed to processors as options but not as arguments of a GCL procedure. See Paragraph *"SYSTEM"*.

2.3.10.3 #CVBOOL

Purpose:	Converts the argument to type bool.
Arguments:	char
Result:	bool
Constraints:	The argument must be valid for a boolean value.

2.3.10.4 #CVDATDEC

Purpose:	Converts a date in the Gregorian calendar from the standard date form YYYY/MM/DD or reduced date form YY/MM/DD to a decimal date form.
Arguments:	char10=date in the format YYYY/MM/DD or YY/MM/DD
Result:	dec6=rank of day, from the referenced date December 31, 1600
Constraints:	none

Examples:	`#CVDATDEC(1995/09/01) --> 144149`
	`#CVDATDEC(#DATE)      --> 144149`
	`(where #DATE=95/09/01)`
	`#CVDATDEC(2000/02/29) --> 145791`
	`#CVDATEC (#EXTDATE)   --> 145791`
	`(where #extdate = 2000/02/29)`
	`#CVDATDEC 2000/02/29  --> 145791`
	`#CVDATDEC(00/02/29)   --> 145791`

Date:	When the form YY/MM/DD is used, the century part of the date is set to 20, except when YY is greater than or equal to 61, then it is set to 19.

2.3.10.5 #CVDEC

Purpose:	Converts the argument to type dec.
Arguments:	char
Result:	dec
Constraints:	The argument must be valid for a numeric value.

2.3.10.6 #CVDECDAT

Purpose:	Converts a date in the Gregorian calendar from the decimal date form to the standard date form YYYY/MM/DD. The reference date is December 31, 1600.
Arguments:	dec6=rank of day, from the reference date.
Result:	char10=date in the format YYYY/MM/DD.
Constraints:	none.

Examples:	`#CVDECDAT(144149) --> 1995/09/01`
	`#CVDECDAT(145791) --> 2000/02/29`

2.3.10.7 #CVFILE

Purpose:	Converts the argument to type file.
Arguments:	char
Result:	file
Constraints:	The argument must be valid for a file.

2.3.10.8 #CVFSET

Purpose:	Converts the argument to type fset.
Arguments:	char
Result:	fset
Constraints:	The argument must be valid for a fileset.

2.3.10.9 #CVHEXA

Purpose:	Converts the argument to type hexa.
Arguments:	char
Result:	hexa
Constraints:	The argument must be valid for a hexadecimal value.

2.3.10.10 #CVLIB

Purpose:	Converts the argument to type lib.
Arguments:	char
Result:	lib
Constraints:	The argument must be valid for a library.

2.3.10.11 #CVNAME

Purpose:	Converts the argument to type name.
Arguments:	char
Result:	name
Constraints:	The argument must be valid for a name.

2.3.10.12 #CVOUTPUT

Purpose:	Converts the argument to type output.
Arguments:	char
Result:	output
Constraints:	The argument must be valid for an output.

2.3.10.13 #CVRFILE

Purpose:	Converts the argument to type rfile.
Arguments:	char
Result:	file
Constraints:	The argument must be valid for a remote file.

2.3.10.14 #CVSTAR

Purpose:	Converts the argument to type star.
Arguments:	char
Result:	star
Constraints:	The argument must be valid for a star-name.

2.3.10.15 #CVVOLUME

Purpose:	Converts the argument to type volume.
Arguments:	char
Result:	volume
Constraints:	The argument must be valid for a volume.

2.3.10.16 #DECTOHEXA

Purpose:	Converts a decimal value to its hexadecimal representation.
Arguments:	dec
Result:	hexa
Constraints:	none

Examples:	`#DECTOHEXA(123)  -->   7B`
	`#DECTOHEXA(6348) --> 18CC`

2.3.10.17 #FB15

Purpose:	Converts the argument to its binary value; the result is a character string of length 2.
Arguments:	dec
Result:	char2
Constraints:	Ranges from -32768 through +32767. Binary values may be passed to processors as options but not as arguments of a GCL procedure. See Paragraph *"SYSTEM"*.

2.3.10.18 #FB31

Purpose:	Converts the argument to its binary value; the result is a character string of length 4.
Arguments:	dec
Result:	char4
Constraints:	Ranges from -2147483648 through +2147483647. Binary values may be passed to processors as options but not as arguments of a GCL procedure. See Paragraph *"SYSTEM"*.

2.3.10.19 #HEXATODEC

Purpose:	Converts a hexadecimal value to its decimal representation.
Arguments:	hexa
Result:	dec
Constraints:	none

Examples:	`#HEXATODEC(7B)   -->  123`
	`#HEXATODEC(18CC) --> 6348`

2.3.10.20 #RJD

Purpose:	Right justifies the decimal number arg1 by inserting leading zeros so that the resulting length is arg2.
Arguments:	(dec, dec)
Result:	dec
Constraints:	number of significant digits of arg1 <= arg2 <= 31

Examples:	`#RJD(123,6) -->  000123`
	`#RJD(3,7)   --> 0000003`

2.3.11 File Handling Builtins

2.3.11.1 #EFN

Purpose:	Extracts the external file name from a file description.
Arguments:	file
Result:	file
Constraints:	none
Examples:	`if #WD is A.B.`

```
if #WD is A.B.
#EFN(.C)                  --> A.B.C
#EFN(F234:K100:MS/D500) --> F234
```

2.3.11.2 #EXPANDPATH

Purpose:	Expands file name its full path name using working directory.
Arguments:	file
Result:	file
Constraints:	none
Examples:	

```
if #WD is A.B
#EXPANDPATH(.D)          --> A.B.D
#EXPANDPATH(<H.I.J)      --> A.H.I.J
#EXPANDPATH(A.B.C)       --> A.B.C
#EXPANDPATH(A:T:MS/D500) --> A:T:MS/D500
```

2.3.11.3 #FSITE

Purpose:	Extracts the name of the site from a remote file description.
Arguments:	rfile
Result:	char
Constraints:	none
Examples:	

```
#FSITE($HERE:A.B.C) --> HERE
#FSITE(C.D.E)       --> ''
```

2.3.11.4 #SUBFILE

Purpose:	Extracts the name of the subfile from a file description.
Arguments:	file
Result:	char
Constraints:	none
Examples:	

```
#SUBFILE(A.B..SF) --> SF
#SUBFILE(A.B.C.D) --> ''
```

3. Command Management

GCL procedures are defined by basic commands. These procedures are then submitted to the MAINTAIN_COMMAND (MNCMD) processor for compilation to be executable.

MAINTAIN_COMMAND also has other functions such as creating, updating, storing and retrieving such procedures. The ON_ERROR command specifies whether MAINTAIN_COMMAND is to abort or to continue execution of commands if a Severity 3 occurs.

3.1 Creating Procedures

A procedure is created in one of two ways:

- The CREATE command allows entering a procedure into the workspace. Each line is compiled as it is entered in *incremental* compilation. The SAVE command then stores the procedure in a binary library.
- The Full Screen Editor or the Text Editor allows entering a source procedure and storing it in a source language library. The COMPILE command then compiles the procedure in *non-incremental* (*bulk*) compilation and stores it in a binary library.

3.2 Binary and Source Libraries, Workspace

Compiled procedures are stored in a binary (BIN) library. Source procedures are stored in a source language (SL) library. Commands BINLIB and SLLIB are used to assign these libraries. *For incremental compilation, a BIN library is always required* but not an SL library.

The workspace is an area in which to create and edit a procedure. Commands which operate on the workspace are:

CREATE	create a procedure in the workspace
LOAD	load a procedure from the binary library into the workspace
RESEQUENCE	renumber the lines in the workspace
APPEND	add lines at the end of the workspace
LEDIT	modify a procedure in the workspace
CLEAR	clear the contents of the workspace.

3.3 Updating Procedures

The tools for modifying a procedure are:

- the Line Editor (LEDIT): a relatively simple editor which operates on the contents of the workspace.
- the Full Screen Editor (FSE): a powerful editor which operates on an SL library member and which can only be used from a terminal with full-screen facilities.
- the Text Editor (EDIT): a powerful editor which operates on an SL library member and which can be used at any terminal.

3.4 Library Management

The following commands maintain libraries of command definitions:

• SAVE	store contents of workspace in the BIN library
• RESAVE	replace procedure in the BIN library with contents of workspace
• COPY	copy procedure(s) from one domain to another
• DELETE	delete procedure(s) from the BIN library
• MODIFY_LOCK	lock/unlock procedures of a domain in the BIN library
• COUNT_ENTRIES	count procedure names, aliases and prompts of a domain in the BIN library
• LIST	display names of procedures in domain
• PRINT	display contents of procedure(s)
• STATUS	display current libraries and other status information
• COMPILE	compile procedure(s) from source text(s)
• DECOMPILE	create source text(s) from compiled procedure(s)
• DOMAIN	specify the current domain
• DISPLAY	display format of command(s)
• DISPLAY_SCREEN	display screens attached to a procedure.
• MERGE	merge two existing domains into a single one.

3.5 Domains

A domain is a set of commands (compiled procedures) available at a certain level of processing.

EXAMPLES:

Domain IOF	System level
Domain LIBMAINT_SL	Processor level
Domain FSE	Processor level

❑

All the commands of a domain are kept in the same subfile in a binary library. The subfile name is identical to the domain name. User created commands can be either additions to an existing domain or belong to a new user domain.

3.5.1 Definition of Domains

A domain is a set of compiled commands or procedures stored in a BIN library member. Commands of a domain specifically belong and pertain to a particular processor.

EXAMPLES OF DOMAINS:

- MAINTAIN_LIBRARY SL commands
- commands of MAINTAIN_COMMAND
- all commands accessible at system level.

❑

3.5.2 Protection of Domains

MAINTAIN_COMMAND does not overwrite an existing domain. It updates the domain by:

- creating a temporary member named `CMDMGT_`*`member`* and writing the records to it
- deleting the previous version of the member
- and giving the name of the previous version to the temporary member.

If MAINTAIN_COMMAND is prevented by job abort or system crash from completing writing successfully, the previous version of the domain still exists in the library.

Even if this procedure were further disrupted, the user can always recover the member under its temporary name. Enough space must therefore be reserved in the library to hold both previous and current versions of the domain. To be on the safe side, provide enough space for all the domains plus extra space to hold the largest of them.

3.5.3 Adding to an Existing Domain

Creating new GCL procedures and storing them in an existing domain provides a new operability to existing features of a domain such as concatenating procedures.

Domains which are associated with the system or system processors are all held in the SYS.HBINLIB library. They are known as standard domains. A list of standard domains where it may be useful to add new commands is given overleaf.

3.5.4 Standard Domains

Commands	Standard Domains
System Level Commands Directives Main Operator Commands	IOF H_NOCTX MAIN
CREATE _FILE CREATE_FILESET	CREATE_BFAS_NONE CREATE_BFAS_SEQ CREATE_LIBRARY CREATE_LIBRARY_FBO CREATE_UFAS_INDEXED CREATE_UFAS_INDEXED_FBO CREATE_UFAS_RANDOM CREATE_UFAS_SEQ_REL CREATE_UFAS_SEQ_REL_FBO
CRPMM (under MAINTAIN_SYSTEM processor)	CREATE_PMM_FUNCTION
CREATE_COMPLEX_GENERATION (in MAIN domain)	CXGEN
CREATE_NETGEN (in MAIN domain)	NG
CREATE_SYSTEM_FILE (under GIUF processor)	TL_DAT TL_DSA TL_FW TL_GCOS TL_GSF TL_OLTD TL_SESSION
Fileset Utilities in driven mode	DMU_FILESET
DEBUG	PCF
DPAN	DPAN DPANCRTR DPANPDTR
EDIT	EDIT
ENTER_GIUF	GIUF
ENTER_RMOS	RMOS
FSE (system level)	F_S_E
FSE (under the LIBMAINT_SL processor)	FSE
Expression "$*$BRK" (break)	H_BREAK
CREATE (under CMDMGT processor)	H_GCL
IQS	IQS
MAINTAIN_AUDIT7	AUDIT7
MAINTAIN_CATALOG	CATMAINT

Commands	Standard Domains
MAINTAIN_COMMAND	CMDMGT
MAINTAIN_DATA_DESCRIPTION	MAINTAIN_DATA_DESCRIPTION
MAINTAIN_FILE	MAINTAIN_FILE MAINTAIN_FILE_FBO
MAINTAIN_FORM	FORMGEN
MAINTAIN_JAS	MAINTAIN_JAS
MAINTAIN_LIBRARY BIN	LIBMAINT_BIN
MAINTAIN_LIBRARY CU	LIBMAINT_CU
MAINTAIN_LIBRARY LM	LIBMAINT_LM
MAINTAIN_LIBRARY SL	LIBMAINT_SL
MAINTAIN_LIBRARY SM	LIBMAINT_SM
MAINTAIN_MFT	MNMFTM
MAINTAIN_MIGRATION	MAINTAIN_MIGRATION
MAINTAIN_STORAGE_MANAGER (in full VOLSET FACILITY/QUOTAS context) alias MAINTAIN_QUOTA	MNSTM_HPS
MAINTAIN_STORAGE_MANAGER in basic VOLSET FACILITY context	MNSTM_AP
MAINTAIN_SYSTEM	MAINTAIN_SYSTEM
MNSYSLM (under MAINTAIN_SYSTEM processor)	MAINTAIN_SYSTEM_LM
MNSYSSM (under MAINTAIN_SYSTEM processor)	MAINTAIN_SYSTEM_SM
MNTZS (under MAINTAIN_SYSTEM processor)	MAINTAIN_TTYPEO_SET
MAINTAIN_EXTENDED_BACKUP	MNXBUP
MAINTAIN_VOLUME	MAINTAIN_VOLUME MAINTAIN_VOLUME_FBO
MODIFY_PMM (under MAINTAIN_PMM processor)	MODIFY_PMM
Maintain_system update commands	MODIFY_SYSTEM_UNIT
PREPARE_TAPESET	PREPARE_TAPESET
MODIFY (under MAINTAIN_LIBRARY processor)	UPDATE
SCANNER	SCANNER
SCAN_VCAM_TRACE	SCAN_VCAM_TRACE

NOTE:

adding private commands or changing access (with MDA command of MAINTAIN_COMMAND processor) on the commands of the H_GCL domain can create garbage in the system.

3.5.5 Creating a New Command

Creating a new command in a standard domain allows personalizing a set of commands. A set of frequently used commands can be set up for the user's convenience and called by a simple name. For instance, a user-defined command named MY_COBOL that calls the standard COBOL command but with pre-defined values. These pre-defined values might include compiler options and/or libraries to be used.

An example is a user procedure that:

- chains the COBOL and LINKER steps for a given source program:
  - the only parameter of the command being the program name
  - the procedure named COBOL-LINK

- and is processed as follows:
  - ***A*** either loaded into a member of an SL library using EDIT to be compiled by MAINTAIN_COMMAND
  - ***B*** or created directly using the CREATE command within MAINTAIN_COMMAND and saved.

Method ***A***:

```
   C: EDIT;
    R: A
     I: PROC COBOL-LINK PROMPT='GCL EXAMPLE' ACCESS=-1
        HIDE=0 OPACC=4 OPHID=0 LOCK=0;
     I: KWD PRG_NAME TYPE=NAME LENGTH=31 NUMVAL=(1,5) ASK=3
        CONCEAL=0;
     I: LOCAL PRG_NAME 31;
     I: UNLIST PRG %PRG_NAME;
     I: COBOL %PRG XREF MAP LEVEL=NSTD;
     I: IF #GT(#SEV,2);
     I: LET # #CAT('COMPILATION OF PROGRAM ',%PRG,' ABORTED');
     I: ELSE;
     I: LINKER %PRG;
     I: ENDIF;
     I: ENDUNLIST;
     I: ENDPROC;
     I: /
    R: W(CMD)COBOL-LINK
   C: /
      .
      .
      .
  S: MAINTAIN_COMMAND;
  >>>14:55 CMDMGT...
   C: SLLIB MYOWN.SL4;
   C: BINLIB MYOWN.BIN5;
   C: DOMAIN IOF;
   C: COMPILE COBOL-LINK;
```

Method ***B**:*

```
S: MAINTAIN_COMMAND;
>>>15:12 CMDMGT...
 C: DOMAIN IOF;
 C: BINLIB MYOWN.BIN5;
 C: CREATE;
  10: PROC COBOL-LINK PROMPT='GCL EXAMPLE' ACCESS=-1
      HIDE=0 OPACC=4 OPHID=0 LOCK=0;
  20: KWD PRG_NAME TYPE=NAME LENGTH=31 NUMVAL=(1,5) ASK=3
      CONCEAL=0;
  30: LOCAL PRG_NAME 31;
  40: UNLIST PRG %PRG_NAME;
  50: COBOL %PRG XREF MAP LEVEL=NSTD;
  60: IF #GT(#SEV,2);
  70: LET # #CAT('COMPILATION OF PROGRAM ',%PRG,' ABORTED');
  80: ELSE;
  90: LINKER %PRG;
  100: ENDIF;
  110: ENDUNLIST;
  120: ENDPROC;
  130: /
 C: SAVE;
```

Line numbers are supplied by the system. The name need not be supplied as this is determined by the PROC statement.

Such a procedure belongs to a standard domain and cannot contain a SYSTEM command. System calls are performed using the CALL command (CALL COBOL or simply COBOL).

3.5.6 Creating a User Domain

Another type of user command is that created in association with a user program. The user creates an interactive program which dialogs through the terminal using a specific set of commands. This set of commands is a user domain which is identified by a name given in the program.

An example of creating a new user domain using the DOMAIN command within the MAINTAIN_COMMAND processor is as follows:

```
S: MAINTAIN_COMMAND;
>>>16:25 CMDMGT .........
 C: DOMAIN MYOWN1;
 C: BINLIB MYOWN.BIN5;
 C: CREATE;
 10: PROC .........;
 20:  .
      .
      .
 130: ENDPROC;
 140: /
 C: SAVE;
```

The domain MYOWN1 will be created when the first procedure is saved in it.

3.6 Libraries

Two kinds of binary libraries exist:

- The system library SYS.HBINLIB contains all the standard commands of all standard domains. These standard commands are available to all users with adequate access and operator rights established in the environment. SYS.HBINLIB belongs implicitly to the library search path of all users.
- Private libraries containing user-defined commands are explicitly declared in the user's binary search path using the MWINLIB command with the BIN option.

Different versions of the same procedure may be stored in different BIN libraries:

- at execution time, the system executes the first one it finds with the name specified in the domain concerned
- the search is done according to the current BIN library search path.

  Consequently, changing the BIN library search path can change the version of a procedure executed in cases where identically named procedures exist in different libraries.

3.7 Access Restrictions

3.7.1 Environments

In defining an environment, it is possible to restrict commands to certain classes of users and also make commands transparent to certain classes of users.

An environment defines a set of commands which are accessible to a certain project regardless of the domain to which the commands belong. The menu of a user working under a certain project presents the visible part of the user's current environment.

The System Administrator defines and personalizes the environments of users at the installation. However, all users who have the appropriate binary library in their binary search path, can define the PROC command without access rights with the appropriate values for ACCESS and HIDE. See Paragraph *"Proc"* for the syntax of the PROC command.

Creating and maintaining environments are treated in the *System Administrator's Manual*.

3.7.2 Access Rights

The following commands manage access rights to commands:

DELETE_ENVT	***reserved for the System Administrator:*** delete an existing environment
ENVT	***reserved for the System Administrator:*** define or modify an environment
LIST_ACCESS	list access rights of procedures of domain(s), see Note
LIST_ENVT	list access rights of environment(s), see Note
LIST_PROJ	list access rights of project(s), see Note
MODIFY_ACCESS	***reserved for the System Administrator:*** modify access rights of procedures of a domain
PROJ	***reserved for the System Administrator:*** specify project's access rights to environments
RESTORE_ACCESS	***reserved for the System Administrator:*** restore access rights of procedures of a domain

RESET	***reserved for the System Administrator:*** reset original access rights of procedures of a domain
SAVE_ACCESS	***reserved for the System Administrator:*** save access rights of procedures of a domain.

NOTE:

The LIST commands display information on objects specified in the command itself and are available to all users.

3.8 Command Management Commands

The set of Command Management Commands are:

APPEND (AP)
BINLIB (LIB)
CLEAR (CLR)
COMPILE (COMP)
COPY (CP)
COUNT_ENTRIES (COUNT)
CREATE (CR)
DECOMPILE (DEC)
DELETE (DL)
DELETE_ENVT (DLENVT)
DISPLAY (D)
DISPLAY_SCREEN (DSCRN)
DOMAIN
EDIT (ED)
ENVT
FSE
LEDIT (LED)
LIST (LS)
LIST_ACCESS (LSA)
LIST_ENVT (LSENVT)
LIST_PROJ (LSPROJ)
LOAD (LD)
MERGE
MODIFY_ACCESS (MDA)
MODIFY_LOCK (MDLK)
ON_ERROR
PRINT (PR)
PROJ
QUIT (Q)
RESAVE (RSV)
RESEQUENCE (RSQ)
RESET
RESTORE_ACCESS (RSTA)
SAVE (SV)
SAVE_ACCESS (SVA)
SLLIB
STATUS (ST)

NOTE:

In the description of the commands, the mention SADMOPT appears. SADMOPT is a CONFIG statement whose parameters declared at GCOS 7 system installation determine the visibility of the commands, namely:

GCLKPROJ	`=NO:`	procedure owner is at user level of the procedure
	`=YES:`	procedure owner is at project level
GCLKSADM	`=YES:`	extension of rights to SYSADMIN
	`=NO:`	no extension of rights to SYSADMIN

3.8.1 APPEND (AP)

Purpose:

To add lines to a procedure definition in the workspace. The system automatically prompts the user with the line number of the line to be entered.

Syntax:

```
{ APPEND }
{        }
{ AP     }

   [ INIT=dec6 ]
   [ STEP={ 10 | dec6 }]
```

Description of Parameters:

INIT	the number of the line after which the new lines are to be added. *If a line before the last line is specified in INIT, all succeeding lines in the workspace are lost.* *Default:* the line number after the last line in the workspace.
STEP	the increment to be used for numbering the new lines. *Default: 10.*

Constraints:

The command is processed as follows:

- each line is checked as it is entered
- if there is an error in the line, an error message appears and the same line number is redisplayed to allow the user to correct the entry
- in novice mode, an extra prompt is displayed
- to terminate the APPEND sequence, enter **/** or **&** or **[F** in the first position on the line.

Examples:

`BINLIB BL1`	assign binary library BL1
`DOMAIN DMN1`	current domain is DMN1
`LOAD P1`	load procedure P1 into workspace from BL1
`AP`	call `APPEND`
`    150:...` }	
`    160:...` }	append new lines
`    170:...` }	
`    180:/`	quit `APPEND`
`RESAVE`	replace procedure P1 with new version

3.8.2 BINLIB (LIB)

Purpose:

To assign a binary library:

- if a BIN library is already assigned, it is replaced by the new one specified
- if the BINLIB command is specified without a library name, the current BIN library is deassigned.

Syntax:

```
{ BINLIB }
{        }
{ LIB    }

   [ LIBRARY=lib78 ]
```

Description of Parameter:

LIBRARY	the name of a binary library *Default:* the current binary library is deassigned.

Constraints:

If no binary library has been assigned, any attempt to operate on binary library such as SAVE will cause an error.

Examples:

`BINLIB BIN1`	assign binary library BIN1
`BINLIB`	deassign current binary library

3.8.3 CLEAR (CLR)

Purpose:

To delete the current contents of the workspace:

- the workspace is empty
- the previous contents cannot be recovered.

Syntax:

```
{ CLEAR }
{       }
{ CLR   }
```

Parameters: *None*

Constraints: *None*

Example:

`CLR`	clear workspace

3.8.4 COMPILE (COMP)

Purpose:

To compile procedures from source texts in source language (SL) library members. The compiled procedures are stored in the binary library as part of the current or specified domain. The COMPILE command provides for non-incremental (bulk) compilation.

Syntax:

```
{ COMPILE }
{         }
{ COMP    }

   { PROC | PROCEDURES }=star62
   [ DOMAIN=name31 ]
   [ BRIEF={ 0 | bool }]
   [ SOURCE={ 1 | bool }]
- - - - - - - - - - - - -  - - - - - - - - - - - - - - - - - - -
   [ OLDVERS={ 0 | bool }]
```

Description of Parameters:

PROCEDURES	Names of the SL library members as a star-name.
DOMAIN	Name of the domain. *Default:* current domain defined by last DOMAIN command.
BRIEF	Extent of Reporting:
=1	Only the lines in error are displayed
=0	*Default:* All lines are displayed and next, the size of the GCL procedure (space used in the assigned BINLIB). The size used in BINLIB depends on the parameter SOURCE value, and it is given in the form: PROCEDURE REAL SIZE (IN BYTES): 129456

SOURCE	If source of procedures is stored in domain subfile of BINLIB:
`=0`	Not stored: so, PRINT, DECOMPILE and LOAD commands cannot be used. *Procedures of standard domains delivered in the library SYS.HBINLIB are compiled with SOURCE=0.*
`=1`	*Default:* Source lines are stored.
OLDVERS	Maximum number of procedures or aliases for different GCOS releases:
`=1`	Maximum limited to 510 and format of the domain subfile is the same as that for previous V3 and V5 releases.
`=0`	*Default:* Maximum can exceed 510 fixed by GCL, in which case, format of domain subfile differs from that of previous releases, and will not be accepted by previous versions of GCL and MAINTAIN_COMMAND.

Constraints:

- COMPILE creates the new compiled procedures as part of:
  - the current domain:
    if the DOMAIN parameter is omitted
    but specified in the DOMAIN command
  - or the specified domain if the DOMAIN parameter is specified.
- Procedures are created with:
  - names provided in the PROC commands that head each procedure definition
  - and line numbers incrementing by 10 regardless of the line numbers used in the source text.
- The owner of a procedure defined with LOCK depends on SADMOPT option:
  - `GCLKPROJ=NO:` the **user** who compiled the procedure
  - `GCLKPROJ=YES:` the **project** of the user who compiled the procedure.
- Entry proceeds as follows:
  - the GCL statement in an SL member may be split into more than one source line
  - two or more GCL statements must not be present in the same source line
  - a procedure containing errors or incomplete is flagged by an error message and is not stored.

- File assignment:
  - BIN and SL libraries must both be assigned through BINLIB and SLLIB commands before COMPILE is used
  - the star-convention applies only to SL members of type CMD.
- Storage of the source code:
  - the source code for compiled procedures with its executable code is stored in SYS.HBINLIB
  - the source code for procedures compiled in standard domains is not saved in the binary library. See Section 10 of the *GCOS 7-V6 System Administrator's Manual* for the list of standard domains.

Examples:

`BINLIB BL1`	assign binary library BL1
`SLLIB SL1`	assign source language library SL1
`DOMAIN DMN`	current domain is DMN
`COMP P2`	compile text P2 and store it in domain DMN
`COMP P1 BRIEF`	compile text P1, do not display
`COMP A*`	compile all texts whose names begin with A

3.8.5 COPY (CP)

Purpose:

To copy one or more procedures of the input domain to the output domain.

Syntax:

```
{ COPY }
{      }
{ CP   }

   { PROCEDURES } { star62                  }
   {            }={                         }
   { PROC       } {( name31 [ name31 ]...)}

   OUTDOM=name31
   INDOM=name31
   [ BRIEF={ 0 | bool }]
   [{ INLIB | IL }=lib78 ]
   [{ REPLACE | RPL }={ 0 | bool }]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   [ OLDVERS={ 0 | bool }]
```

Description of Parameters:

PROCEDURES	Procedure(s) to be copied specified either with the asterisk (star) convention or as a list of up to 8 names.
OUTDOM	Name of the output domain
INDOM	Name of the input domain
BRIEF	Extent of Reporting:
=1	Only the lines in error are displayed
=0	*Default:* All lines are displayed.
INLIB	Library in which the input domain is searched for. *Default:* both *input* and *output domains* are in the same library specified by the BINLIB command.

REPLACE	If procedure from input domain overwrites another with the same name in the output domain.
`=1`	Replace
`=0`	*Default:* No replace.
BRIEF	If names and characteristics of copied procedures are displayed:
`=1`	No display
`=0`	*Default:* Display.
OLDVERS	Maximum number of procedures or aliases for different GCOS releases:
`=1`	Maximum limited to 510 and format of domain subfile is the same as that for previous V3 and V5 releases.
`=0`	*Default:* Maximum can exceed 510 fixed by GCL, in which case, format of domain subfile differs from that of previous releases, and will not be accepted by previous versions of GCL and MAINTAIN_COMMAND.

Constraints:

- The output library must be assigned through the BINLIB command before COPY is submitted.
- Domain requirements:
  - both input and output domains must exist
  - when INLIB is omitted, INDOM and OUTDOM must be different.
- Restrictions on procedures:
  - the star convention applies only to procedure names not to *aliases*
  - procedures locked in the output domain can only be replaced by their owners.

3.8.6 COUNT_ENTRIES (COUNT)

Purpose:

To count the number of procedure names, aliases and prompts of a given domain in the current BIN library. All of the procedure and alias names are counted, even if the procedures are hidden.

Syntax:

```
{ COUNT_ENTRIES }
{               }
{ COUNT         }

   [{ DOMAIN | DOMAINS }=star62 ]
```

Description of Parameter:

DOMAINS	Name(s) of the domain(s). *Default:* current domain defined by the last DOMAIN command.

Constraints:

The BIN library must be assigned through the BINLIB command before COUNT_ENTRIES can be used.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`COUNT_ENTRIES IOF`	count all procedure names, aliases and prompts of the IOF domain in the BIN library BLIB

3.8.7 CREATE (CR)

Purpose:

To create a new procedure in the workspace. Lines are numbered automatically as they are entered.

Syntax:

```
{ CREATE }
{        }
{ CR     }

   [ INIT={ 10 | dec6 }]
   [ STEP={ 10 | dec6 }]
```

Description of Parameters:

INIT	Line number to be assigned to the first line. *Default: 10*
STEP	the increment to be used for numbering each new line entered. *Default: 10*

Constraints:

The command is processed as follows:

- each line is checked as it is entered
- if there is an error in the line, an error message appears and the same line number is redisplayed to allow the user to correct the entry
- in novice mode, an extra prompt is displayed
- to terminate the APPEND sequence, enter **/** or **&** or **[F** in the first position on the line.

Examples:

`BINLIB BLIB1`	assign binary library BLIB1
`DOMAIN DMN1`	current domain is DMN1

```
CR
   10:PROC P1...
   20:...          }
   30:...          }   new procedure definition
   40:...          }
   50:/                leave CREATE
```

`SAVE`	store workspace (procedure P1) in current domain

3.8.8 DECOMPILE (DEC)

Purpose:

To decompile procedures from the binary library. The source texts produced by DECOMPILE are stored as source language (SL) library members in the library assigned by the SLLIB command.

Syntax:

```
{ DECOMPILE }
{           }
{ DEC       }

   { PROC | PROCEDURES }=star62
   [ DOMAIN=name31 ]
   [{ REPLACE | RPL }={ 0 | bool }]
   [ PREFIX=name6 ]
- - - - - - - - - - - - - - - - - - - - - - - - - -
   [ COMPACT={ 0 | bool }]
```

Description of Parameters:

PROCEDURES	Names of the procedures as a star-name.
DOMAIN	Name of the domain to which the procedures belong. *Default:* current domain defined in last DOMAIN command.
REPLACE	Allows members being created in the SL library to replace (overwrite) existing members with the same names.
PREFIX	Prefix used when creating the SL library members. *Default:* Names of members are those of decompiled procedures.

COMPACT	How source statements are handled:
`=1`	Source statements split into more than one line, will be grouped together as a single record up to 255 characters.
`=0`	*Default:* Each source line is written as a separate record in the SL library.

Constraints:

- Restrictions on decompilation:
  - A procedure compiled with `SOURCE=0` in the COMPILE, SAVE or RESAVE commands cannot be decompiled
  - A locked procedure can be decompiled by:
    only its owner (User or Project)
    or a user of the SYSADMIN project.
- Restrictions on DECOMPILE:
  - BIN and SL libraries must both be assigned through BINLIB and SLLIB commands before DECOMPILE is used
  - DECOMPILE cannot be used on procedures compiled in standard domains since their source code is not saved in SYS.HBINLIB. See Section 10 of the *GCOS 7-V6 System Administrator's Manual* for the list of standard domains.
  - if a member of the same name exists in the SL library and is not type CMD, an error is returned and no member is created
  - if the member is type CMD, it is overwritten if `REPLACE=1`; otherwise a message is issued and no member is created.
- DECOMPILE creates new SL members:
  - in the SL library specified by the SLLIB command
  - with names of the corresponding procedure headed by *`prefix`* and truncated, if necessary, to 31 characters.
  - in which the new source texts are given the same line numbers as those used in the compiled texts (those obtained using the PRINT command of MNCMD).

Examples:

`BINLIB BL1`	assign binary library BL1
`SLLIB SL1`	assign source language library SL1
`DOMAIN D`	current domain is D
`DEC P1`	decompile procedure P1, store source in SL1

3.8.9 DELETE (DL)

Purpose:

To delete one or more procedures of a domain in the current binary library.

Syntax:

```
{ DELETE }
{        }
{ DL     }

   { PROCEDURES } { star62                 }
   {            }={                        }
   { PROC       } {( name31 [ name31 ]...)}

   [ DOMAIN=name31 ]
   [ BRIEF={ 0 | bool }]
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names.
DOMAIN	Name of the domain. *Default:* current domain defined in last DOMAIN command.
BRIEF	If names and characteristics of deleted procedures are displayed:
`=1`	No display
`=0`	*Default:* Display.

Constraints:

- The BIN library in which the domain is stored must be assigned before using DELETE; use the BINLIB command to assign it.
- A locked procedure can be deleted only by the User or Project that set the lock, or by a user of project SYSADMIN if the CONFIG SADMOPT statement has `GCLKSADM=YES`.
- Star-name applies only to procedure names, not to aliases.

Examples:

`BINLIB BL1`	assign binary library BL1
`DOMAIN D`	current domain is D
`DL P1`	delete procedure P1 in domain D
`DL * DMN`	delete all procedures in domain DMN
`DL ^P*`	delete all procedures whose names do not begin with P

3.8.10 DELETE_ENVT (DLENVT)

Purpose:

Reserved for the System Administrator: To delete an existing environment. If this environment is not the default environment, then DELETE_ENVT deletes all relations between the environment and all attached projects.

Syntax:

```
{ DELETE_ENVT }
{             }
{ DLENVT      }

   ENVIRONMENT=name12
```

Description of Parameter:

ENVIRONMENT	Name of environment.

Constraints:

- The specified environment must exist.
- It is not deleted if it is the default environment for a project. To delete it, use the PROJ Command.

Example:

`DLENVT ENVT1`	delete environment ENVT1

3.8.11 DISPLAY (D)

Purpose:

To display the syntax of one or more procedures.

Syntax:

```
{ DISPLAY }
{         }
{ D       }

   { PROCEDURES } { star62               }
   {            }={                      }
   { PROC       } {( name31 [ name31 ]...)}

   [ DOMAIN=name31 ]
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names.
DOMAIN	Name of domain. *Default:* current domain defined in last DOMAIN command.

Constraints:

The BIN library in which the domain is stored must be assigned before using DISPLAY; use the BINLIB command to assign it.

Examples:

`BINLIB BL1`	assign binary library BL1
`DOMAIN D`	current domain is D
`D X*`	display syntax of procedures beginning with X* in domain D

3.8.12 DISPLAY_SCREEN (DSCRN)

Purpose:

To display screens attached to a GCL procedure without leaving MNCMD.

The characteristics and the rules for the display of the screens are similar to those in interactive mode menu.

The command process execution is the same as the one executed when proc_name followed by the "?" character is entered at prompting level of a processor, but the execution stops after the display of the screens.

This command is used as a tool to debug the screens look (Keywords order, prompts and notes choice).

Syntax:

```
{ DISPLAY_SCREEN }
{ DSCREEN }
{ DSCRN }
               { PROCEDURE | PROC } = name31
                [ DOMAIN = name31 ]
```

Parameters:

PROCEDURE	Name or alias of the procedure for which the screens will be displayed.(NB: the procedure is not executed).
DOMAIN	Name of the domain. Default: current domain defined by last DOMAIN command.

Constraints:

- Restrictions on displaying_screen:
  - a procedure compiled with SOURCE = 0 in the COMPILE, SAVE or RESAVE cannot be displayed.
  - a locked procedure can be displayed only by:

    its owner (User or Project)

    or a user under SYSADMIN project if GCLKSADM = YES appears in SADMOPT.

- a procedure can be displayed only if the parameters of the statement PROC allows the access and execution of the procedure for an IOF user:

  OPACC: contains at least the value 5.

  OPHID: the procedure is not hidden.

  LIMITED_ACCESS = 0: the procedure can be called directly.

- Restrictions on DISPLAY_SCREEN:
  - the procedure must exist in the specified or current domain.
  - the BIN library must be assigned through BINLIB command before DISPLAY_SCREEN is used and must be included in the GCL search path.
  - DISPLAY_SCREEN cannot be used on procedures compiled in standard domains since their source code is not saved in SYS.HBINLIB.
  - COMFILE and PRTFILE of MNCMD must not be specified when DISPLAY_SCREEN is used.
  - DISPLAY_SCREEN cannot be used in BATCH mode.
  - DISPLAY_SCREEN command is provided as a tool to help the user during the development phase. If more than one user uses this command to display the same procedure on a same BINLIB, at the same time, some error messages may occur.

To suppress most of the constraints in the use of DISPLAY_SCREEN command, we recommended to set the right values of OPACC, OPHID and LIMITED_ACCESS only at the end of the development phase.

Example:

`BINLIB BLIB1`	assign binary library BLIBL1
`DOMAIN MYDMN`	specify domain MYDMN
`DSCRN MYPROC`	display the screens of MYPROC

3.8.13 DOMAIN

Purpose:

To define the current domain. A domain is a set of related compiled procedures stored in the binary library. In the library, the domain is a member and the procedures are stored as sets of consecutive records within the member.

Syntax:

```
DOMAIN

   DOMAIN=name31
```

Description of Parameter:

DOMAIN	Name of existing domain or a domain to be created.

Constraints:

None

Examples:

`BINLIB BL1`	assign binary library BL1
`SLLIB SL1`	assign source language library SL1
`DOMAIN AA`	current domain is AA
`COMPILE P1`	compile source procedure P1, store in domain AA of BL1
`DOMAIN BB`	change current domain to domain BB
`LOAD P1`	load procedure P1 of domain BB in BL1 into workspace

3.8.14 EDIT (ED)

Purpose:

To call the Text Editor, by means of which one can create or modify a source text in the source language (SL) library.

Syntax:

```
{ EDIT }
{      }
{ ED   }
```

Description of Parameters:

None

Constraints:

- The Text Editor can only be used on a source language (SL) library member assigned through the SLLIB command.
- To use the Text Editor on a compiled procedure in a BIN library:
  - first use the DECOMPILE command to convert the procedure to source text
  - store it as an SL library member
  - after the Text Editor session, create a new version of the procedure in the BIN library through the COMPILE command.

The Text Editor is described in the *Text Editor User's Guide*.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`SLLIB SLIB`	assign source language library SLIB
`DOMAIN D1`	current domain is D1
`DECOMPILE MYPROC`	decompile MYPROC from domain D, store text in SLIB
`EDIT`	call Text Editor
`.` `.` `.`	} sequence of Text Editor requests
`COMPILE MYPROC`	compile new version of MYPROC, store it in domain D1

3.8.15 ENVT

Purpose:

To create a new environment (or modify an existing one) by defining the families to which the environment has access. ENVT can be used only by the System Administrator.

Syntax:

ENVT

ENVIRONMENT=*name12*

FAMILIES=(*dec3* [-*dec3*]...[*dec3* [-*dec3*])

[MODE={ CR | ADD | DL }]

Description of Parameters:

ENVIRONMENT	Name of environment being defined or modified.
FAMILIES	List of up to 32 families or ranges of families to which the environment is granted access. A family could be defined as a set of GCL procedures. The set of families to which a procedure belongs is defined through the ACCESS parameter of the PROC command. Each family ranges from 1 through 256.

```
COMMAND-A <----|
               |--- FAMILY-1 <---+
COMMAND-B | <--|                 |
          |                      |
          | <--|                 |<-- ENVT-1
               |                 |
COMMAND-C <----|                 |
               |--- FAMILY-2 |<--|
               |             |
COMMAND-D | <--|             |<--|
          |                      |
          | <--|                 |
               |                 |<-- ENVT-2
COMMAND-E <----|--- FAMILY-3 <---|
               |                 |
COMMAND-F <----|                 |
                                 |
COMMAND-G <-------- FAMILY-4 <---|
```

MODE	How specified families are processed for the environment:
`=ADD`	Add families to the current environment.
`=DL`	Delete families from the current environment.
`=CR`	*Default:* Create or recreate the environment for the families.

Constraints:

None

Examples:

`ENVT ENVT-1 (1,2)`	create environment ENVT-1 with families 1,2
`ENVT ENVT-2 (2,3,4)`	create environment ENVT-2 with families 2,3,4
`ENVT MYENV 10`	create environment MYENV with family 10
`ENVT AA 0`	delete environment AA
`ENVT BB (1,2,3)`	create environment BB with families 1,2,3
`ENVT BB (1,2,3,4)`	redefine environment BB with families 1,2,3,4
`ENVT BB 5`	redefine environment BB with family 5
`ENVT BB (10-20, 33) ADD`	add families 10 to 20 and 33 to environment BB

3.8.16 FSE

Purpose:

To call the Full Screen Editor for creating or modifying a text member in an SL (source language) library.

Syntax:

```
FSE
```

Parameters:

None

Constraints:

FSE can only be used on an SL library member; to use it on a compiled procedure in a BIN library:

- first convert the procedure to source text through the DECOMPILE command
- store it as an SL library member; the SL library must be assigned through the SLLIB command before FSE can be used
- after the FSE session, recompile the edited text through the COMPILE command.

FSE is described in the *Full Screen Editor User's Guide*.

Examples:

```
BINLIB BLIB        assign binary library BLIB

SLLIB SLIB         assign source language library SLIB

DOMAIN D-2         current domain is D-2

DECOMPILE MYPROC   decompile MYPROC of domain D-2 and store text in SLIB

FSE                call Full Screen Editor
   .         }
   .         }     sequence of Full Screen Editor requests
   .         }

COMPILE MYPROC     compile MYPROC from SLIB, store it in domain D-2
```

3.8.17 LEDIT (LED)

Purpose:

To activate the Line Editor to operate on lines in the workspace:

- Insert (or replace) lines
- Delete lines
- Print lines
- Substitute strings in lines.

Syntax:

```
{ LEDIT }
{       }
{ LED   }
```

Parameters:

None

Constraints:

Insert

To insert a line, enter the line number, a colon and the command to be inserted. If a line of that number already exists, it is replaced by the new line. Each line is checked as it is entered (same as in CREATE).

Example:

```
100: GOTO ERROR;
```

Delete

To delete a line, enter the line number and the letter D. To delete more than one line, enter a list of line numbers, separated by commas, then D. To delete a range of lines, enter the first and last line numbers, separated by a dash, then D.

Example:

`10D`	delete line 10
`20,50,80D`	delete lines 20, 50, and 80
`20-80D`	delete all lines from 20 to 80

Print

To print a line, enter the line number and the letter P. To display more than one line, enter a list of line numbers, separated by commas, then P. To display a range of lines, enter the first and last line numbers, separated by a dash, then P.

Example:

`40P`	print line 40
`20,50,180P`	print lines 20, 50, and 180
`20-180P`	print all lines from 20 to 180

Substitute

To replace an existing character string with a new character string, use the substitute request of format: `S/old-string/new-string/`

The S request must be preceded by the number(s) of the lines(s) to be substituted. Every occurrence of the old string is replaced by the new string.

Example:

`10S/ABCDE/xyz/`	replace `xyz` with `ABCDE` in line 10
`70,150,220S/AB/xy(z)/`	replace `xy(z)` with `AB` in lines 70, 150, and 220
`10-80S/;/22;/`	replace `22`; with character ; in lines 10 through 80

If `P` is entered at the end of the request, the line(s) involved in the substitution are displayed after the substitution has been done.

Example:

```
190-400S/CALL A/CALL BB/P
```

Slash (/) is the delimiter in all of the examples above, however the following characters also serve as delimiters:

```
/ ! # ' % & * = [ ] [ . + - $ ; > < ( )
```

To quit the Line Editor enter / (or & or [F).

Examples:

```
DOMAIN DMN1              current domain is DMN1

BINLIB BL                assign binary library BL

LOAD MYPROC              load procedure into workspace

LED                      call Line Editor
I:...               }
I:...               }    sequence of Line Editor requests
I:...               }
I:/                      quit Line Editor

RESAVE                   save edited procedure
```

3.8.18 LIST (LS)

Purpose:

To display the names of all the procedures of a given domain in the current BIN library dates of their most recent modification and the number of their commands.

Syntax:

```
{ LIST }
{      }
{ LS   }

   { PROCEDURES } { *                    }
   {            }={ star62               }
   { PROC       } {( name31 [ name31 ]...)}

   [{ DOMAIN | DOMAINS }=star62 ]

   [{ ENVT | ENVIRONMENT }=name12 ]

   [ OPACC=( dec1 [ dec1 ]...)]

   [ OPHID=( dec1 [ dec1 ]...)]

   [ ACCESS=( dec3 [ dec3 ]...)]

   [ HIDE=( dec3 [ dec3 ]...)]

   [ OWNER={ 0 | bool }]
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names. *Default:* All procedures in the domain.
DOMAINS	Name(s) of the domain(s). *Default:* the current domain.
ENVIRONMENT	Restricts list to the procedures accessible under the specified environment.
OPACC	Restricts list to the procedures having one of the specified operator rights.
OPHID	Restricts list to the procedures hidden under one of the specified operator rights.

ACCESS	Restricts list to the procedures belonging to one of the specified families.
HIDE	Restricts list to the procedures hidden under one of the specified families.
OWNER	If the owner of the procedure defined with LOCK is to appear:
`=1`	If the command submitter is SYSADMIN and the CONFIG statement SADMOPT specifies: `GCLKSADM=YES`: the name of the user of the procedure is displayed. If GCLKPROJ is specified, the user is a Project and is prefixed with * in the list. `GCLKSADM=NO`: OWNER is ignored. If the submitter is an IOF user with access to the procedure source, the owner appears in front of the procedure name.
`=0`	*Default:* No owner name displayed.

Constraints:

The BIN library must be assigned through the BINLIB command before LIST can be used.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`DOMAIN D`	current domain is D
`LS`	list all procedures in the current domain D
`LS PR* PAYROLL`	list procedures of PAYROLL with names beginning with PR

3.8.19 LIST_ACCESS (LSA)

Purpose:

To list the access rights and priorities of the procedures of a given domain stored in the current BIN library.

Syntax:

```
{ LIST_ACCESS }
{             }
{ LSA         }

   [{ PROCEDURES } { *                     }]
   [{            }={ star62                }]
   [{ PROC       } {( name31 [ name31 ]...)}]

   [{ DOMAIN | DOMAINS }=star62 ]
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names. *Default:* all procedures in the domain.
DOMAINS	Name(s) of the domain(s). *Default:* the current domain.

Constraints:

- Access rights are listed in the form of two arrays and a status character:
  `"+"` means that the procedure is accessible
  `"-"` means that it is accessible but hidden
  `"`*space*`"` means that the procedure is not accessible.
- The BIN library must be assigned through the BINLIB command before LIST_ACCESS can be used.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`DOMAIN D`	current domain is D
`LSA`	list access rights and priorities of all procedures in domain D
`LSA ^E* D2`	list access rights and priorities of all procedures whose names do not begin with E, for domain D2

3.8.20 LIST_ENVT (LSENVT)

Purpose:

To display the list of the families to which the specified environment(s) have access.

Syntax:

```
{ LIST_ENVT }
{           }
{ LSENVT    }

   ENVIRONMENTS=star24
```

Description of Parameters:

ENVIRONMENTS	Single environment or a star-name for a set of environments to be displayed with the associated families.

Constraints:

None

Examples:

`LSENVT ENV1`	list all families for environment ENV1
`LSENVT E*`	list all families for environments whose names begin with E

3.8.21 LIST_PROJ (LSPROJ)

Purpose:

To list the environments to which the specified project(s) have access.

Syntax:

```
{ LIST_PROJ }
{           }
{ LSPROJ    }

   PROJECTS=star24
```

Description of Parameter:

PROJECTS	Single project or a star-name for a set of projects.

Constraints:

None

Examples:

`LSPROJ PT1`	list all environments for project PT1
`LSPROJ PT*`	list all environments for projects whose names begin with PT

3.8.22 LOAD (LD)

Purpose:

To load a procedure from the BIN library into the workspace.

Syntax:

```
{ LOAD }
{      }
{ LD   }

{ PROC | PROCEDURE }=name31
[ DOMAIN=name31 ]
```

Description of Parameters:

PROC	Name of the procedure.
DOMAIN	Name of the domain. *Default:* current domain defined by DOMAIN command.

Constraints:

- Restrictions on loading:
  - a procedure compiled with `SOURCE=0` in the COMPILE, SAVE or RESAVE commands cannot be loaded
  - a locked procedure can be loaded by:
    only its owner (User or Project)
    or a user under SYSADMIN project if `GCLKSADM=YES` appears in SADMOPT.
- Restrictions on LOAD:
  - the procedure must exist in the specified or the current domain
  - the BIN library must be assigned through BINLIB command before LOAD is used
  - LOAD cannot be used on procedures compiled in standard domains since their source code is not saved in SYS.HBINLIB. See Section 10 of the *GCOS 7-V6 System Administrator's Manual* for the list of standard domains.

Examples:

`BINLIB BLIB1`		assign binary library BLIB1
`DOMAIN MYDMN`		specify domain MYDMN
`LD MYPROC`		load procedure MYPROC into workspace
`LEDIT`		call the Line Editor
`.`	}	
`.`	}	sequence of Line Editor requests
`.`	}	
`/`		leave the Line Editor
`RESAVE`		save new version of MYPROC in BIN library BLIB1 of domain MYDMN

3.8.23 MERGE

Purpose:

To merge two existing domains into a single one by selectively copying the procedures of the *input domain* into the *output domain*:

- *input domain* procedures absent in the *output domain* are copied into it
- *input domain* procedures replace those of identical names in the *output domain*.

Syntax:

```
MERGE

   OUTDOM=name31
   INDOM=name31
   [{ INLIB | IL }=lib78 ]
- - - - - - - - - - - - - - - - - - -
   [ OLDVERS={ 0 | bool }]
```

Description of Parameters:

OUTDOM	Name of the *output domain*.
INDOM	Name of the *input domain*.
INLIB	Library in which the *input domain* is searched for. *Default:* both *input* and *output domains* are in the same library specified by the BINLIB command.
OLDVERS	Maximum number of procedures or aliases for different GCOS releases:
=1	Maximum limited to 510 and format of domain subfile is the same as that for previous V3 and V5 releases.
=0	*Default:* Maximum can exceed 510 fixed by GCL, in which case, format of domain subfile differs from that of previous releases, and will not be accepted by previous versions of GCL and MAINTAIN_COMMAND.

Constraints:

Restrictions on MERGE:

- the output library must be assigned through the command BINLIB before MERGE can be used
- the *output domain* must exist in the output library
- when INLIB is omitted, INDOM and OUTDOM must be different.

Examples:

`BINLIB BLIB1`	assign binary library BLIB1
`MERGE IOF IOF_TEMP1`	copy all procedures of domain IOF_TEMP1 into domain IOF
`MERGE IOF IOF_TEMP1 .INLIB1`	same as above; IOF_TEMP1 is in library .INLIB1

3.8.24 MODIFY_ACCESS (MDA)

Purpose:

Reserved for the System Administrator: To modify the access rights or priorities of procedures of the current BIN library.

Syntax:

```
{ MODIFY_ACCESS }
{               }
{ MDA           }

   { PROCEDURES } { star62                }
   {            }={                       }
   { PROC       } {( name31 [ name31 ]...)}

   [ DOMAIN=name31 ]
   [ PRTY=dec3 ]
   [ ACCESS=( nnn [ -nnn ]...[ nnn [ -nnn ]])]
   [ ACCESS_MODE={ DL | ADD | RPL }]
   [ HIDE=( nnn [ -nnn ]...[ nnn [ -nnn ]])]
   [ HIDE_MODE={ ADD | DL | RPL }]
   [ OPACC=( dec1 [ dec1 ]...)]
   [ OPACC_MODE={ DL | ADD | RPL }]
   [ OPHID=( dec1 [ dec1 ]...)]
   [ OPHID_MODE={ ADD | DL | RPL }]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   [ NO_ACCESS=( nnn [ -nnn ]...[ nnn [ -nnn ]])]
   [ NO_OPACC=( dec1 [ dec1 ]...)]
   [ LIMITED_ACCESS=bool ]
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names.
DOMAIN	Name of the domain. *Default:* current domain defined in last DOMAIN command
PRTY	New priority for the procedure.

ACCESS	List of up to 32 family numbers (nnn) or ranges of family numbers (nnn-nnn): – to be added to or deleted from the procedure – or to replace the current list in the procedure, depending on the value specified by ACCESS_MODE.
ACCESS_MODE	Specifies how families *nnn* or family ranges *nnn-nnn* in ACCESS are used:
`=ADD`	Add to current list of the procedure.
`=RPL`	Replace current list of the procedure.
`=DL`	*Default:* Delete from list of the procedure.
HIDE	List of up to 32 families *nnn* or family ranges *nnn-nnn:* – from which the procedure is hidden or no longer hidden – or which is to replace the current list in the procedure, depending on the value specified by HIDE_MODE.
HIDE_MODE	Specifies how the list of families or family ranges given in HIDE is to be processed:
`=DL`	Procedure is no longer to be hidden.
`=RPL`	Replace current list in the procedure by list given in HIDE.
`=ADD`	*Default:* Procedure is hidden from the list given in HIDE.
OPACC	List of up to 8 operator rights: – to be added to or deleted from the procedure – or which are to replace the current list in the procedure, depending on the value specified by OPACC_MODE.
OPACC_MODE	Specifies how operator rights given in OPACC are used:
`=ADD`	Add operator rights to the procedure.
`=RPL`	Replace operator rights in the current list in the procedure.
`=DL`	*Default:* Delete operator rights from the procedure.
OPHID	List of up to 8 operator rights: – from which the procedure is hidden or no longer hidden – or which is to replace the current list in the procedure, depending on the value specified by OPHIDE_MODE.

OPHIDE_MODE	Specifies how operator rights given in OPHID are used:
`=DL`	Procedure is no longer to be hidden for the operator rights.
`=RPL`	Replace the current list in the procedure.
`=ADD`	*Default:* Procedure to be hidden for the listed operator rights.
NO_ACCESS	List of up to 32 family numbers *nnn* or ranges of family numbers *nnn-nnn* to be suppressed in the procedure.
HIDE	List of up to 32 family numbers *nnn* or ranges of family numbers *nnn-nnn* for which the procedure is to be hidden.
OPHID	List of up to 8 operator rights from which the procedure is to be hidden.
NO_OPACC	List of up to 8 operator rights to be suppressed for the procedure.
LIMITED_ACCESS	Determines how the procedure is called. It applies only to users who do not belong to SYSADMIN project. There is no link between attribute LIMITED_ACCESS and environments. A command having this attribute is not directly accessible for all projects, except SYSADMIN. Values:
`=0`	Procedure may be accessed directly
`=1`	Procedure must be called by another procedure.

Constraints:

- Restrictions on MODIFY_ACCESS:
  - the BIN library must be assigned through the BINLIB command before MODIFY_ACCESS can be used.
  - star-name applies only to procedure names not to aliases.
- The following parameters are mutually exclusive:
  - ACCESS and NO_ACCESS
  - OPACC and NO_OPACC.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`DOMAIN D`	specify current domain D
`MODIFY_ACCESS MYPROC NO_ACCESS=(5,6) HIDE=(1,2)`	modify access for MYPROC procedure, suppressing access rights for families 5 and 6, and hiding the procedure for families 1 and 2
`MDA P PRTY=200`	modify priority of procedure P
`MDA E1 NO_OPACC=4`	suppress operator right 4 for procedure E1
`MDA E2 OPHID=4`	hide operator right 4 for procedure E2
`MDA P1 NO_ACCESS=(1 3 10-12) HIDE=(5 20-22)`	suppress access for families 1, 3, 10, 11, and 12; hide for families 5, 20, 21, and 22

3.8.25 MODIFY_LOCK (MDLK)

Purpose:

To lock or unlock procedures of a given domain in the current BIN library.

Syntax:

```
{ MODIFY_LOCK }
{             }
{ MDLK        }

   { PROCEDURES } { star62                }
   {            }={                       }
   { PROC       } {( name31 [ name31 ]...)}

   [ DOMAIN=name31 ]
     NEWVALUE={ LOCK | UNLOCK }
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names.
DOMAIN	Name of the domain. *Default:* current domain defined in last DOMAIN command.
NEWVALUE	sets or resets *locking* of the specified procedure(s)
=LOCK	lock
=UNLOCK	unlock.

Constraints:

- The BIN library must be assigned through the BINLIB command MODIFY_LOCK can be used.
- A locked procedure can be unlocked by:
  - only the User or Project that set the lock
  - or a user of SYSADMIN project if `GCLKSADM=YES` appears in the SADMOPT.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`MODIFY_LOCK (CBL FOR77) DOMAIN=IOF LOCK`	
	lock procedures CBL and FOR77 of domain
IOF in	
	BIN library BLIB

3.8.26 ON_ERROR

Purpose:

To specify the action of MAINTAIN_COMMAND if an error of Severity 3 occurs during the execution of a command.

Syntax:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ON_ERROR

   [ ACTION={ DEFAULT | ABORT | CONTINUE | RESET }]
```

Description of Parameters:

ACTION	the action requested when a Sev 3 error occurs:
`=ABORT`	MAINTAIN_COMMAND is terminated.
`=CONTINUE`	execution of commands continues; the next statement, whether of the current record (line) or of the following record (line), is executed; error is ignored.
`=RESET`	the Sev 3 error is released and execution of commands continues.
`=DEFAULT`	*Default:* Execution of commands continues: – the error is ignored – the statements of the current record (line) are skipped – the statements of the following record (line) are executed.

Constraints:

None

Examples:

ON_ERROR ACTION=ABORT	if a Sev 3 error occurs, abort MAINTAIN_COMMAND
ON_ERROR RESET	if a Sev 3 error occurs, release the error indicator and continue execution of commands

3.8.27 PRINT (PR)

Purpose:

To print one or more procedures of a given domain stored in the current BIN library. PRINT operates only on compiled procedures stored in the current BIN library. To print all or part of the workspace, use the P request of the Line Editor (LEDIT) command.

Syntax:

```
{ PRINT }
{       }
{ PR    }

 { PROCEDURES } { star62                }
 {            }={                       }
 { PROC       } {( name31 [ name31 ]...)}

 [ DOMAIN=name31 ]
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names.
DOMAIN	Name of the Domain. *Default:* current domain defined in last DOMAIN command.

Constraints:

- Restrictions on printing:
  - a procedure compiled with `SOURCE=0` in the COMPILE, SAVE or RESAVE commands cannot be printed
  - a locked procedure can be printed by:
    only its owner (User or Project)
    or a user of SYSADMIN project if `GCLKSADM=YES` appears in SADMOPT.
- Restrictions on PRINT:
  - the BIN library must be assigned through BINLIB command before PRINT is used
  - PRINT cannot be used on procedures compiled in standard domains since their source code is not saved in SYS.HBINLIB. See Section 10 of the *GCOS 7-V6 System Administrator's Manual* for the list of standard domains.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`DOMAIN D`	specify current domain D
`PR P1`	print procedure P1, current domain D
`PR A1 AA`	print procedure A1, domain AA
`PR * DMN1`	print all procedures in domain DMN1

3.8.28 PROJ

Purpose:

Reserved for the System Administrator: To specify the environment(s) to which a project has access.

Syntax:

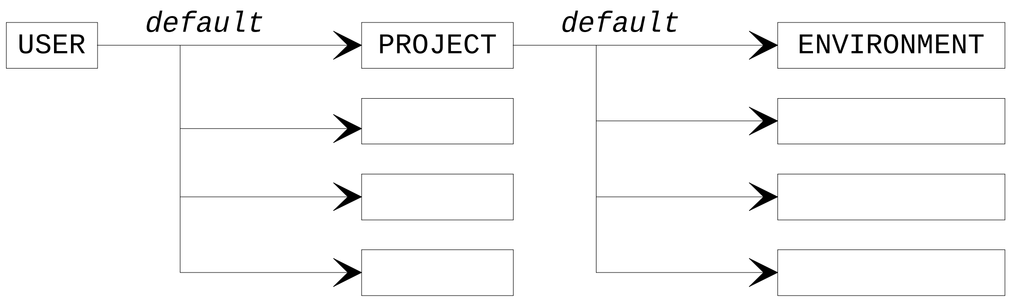
```
PROJ    PROJECT=name12

        ENVIRONMENTS=( name12 [ name12 ]...)

        [ MODE={ CR | ADD | DL }]
```

Description of Parameters:

PROJECT	Name of project accessing the environment(s).
ENVIRONMENTS	List of up to 32 names of environments to which the project is to have access or no longer have access if `MODE=DL`. The first name in the list is the default environment for the project.
	`ENVIRONMENTS=0`: *Not allowed if `MODE=ADD` or `DL`*, specifies an environment which allows access to all families.
MODE	Defines how access rights are to be processed:
`=ADD`	Adds new access rights to the project.
`=DL`	Suppresses access rights attached to the project.
`=CR`	*Default:* Defines or redefines the project's access rights.

Constraints:

The environment(s) must exist before they can be specified in PROJ. To create an environment, use ENVT. The general project structure is as follows:



Examples:

`PROJ PR1 (ENV1, ENV2, ENV3)`	project PR1 has access to ENV1 (default), ENV2, ENV3
`PROJ AA E1`	project AA has access to environment E1
`PROJ P3 0`	project P3 has access to all families

3.8.29 QUIT (Q)

Purpose:

To leave the Command Management processor.

Syntax:

```
{ QUIT }
{      }
{ Q    }
```

Parameters:

None

Constraints:

If the contents of the workspace have been modified since loaded, and no SAVE or RESAVE has been made, the system asks whether the user wants to save what has been edited. Reply YES or NO as appropriate.

Example:

`QUIT or Q`	leave MAINTAIN_COMMAND

3.8.30 RESAVE (RSV)

Purpose:

To replace an existing procedure in the BIN library with the contents of the workspace:

- the new procedure is created with the same name as the old one
- if no procedure of that name exists, RESAVE creates a new one.

Syntax:

```
{ RESAVE }
{        }
{ RSV    }

   [ FORCE={ 0 | bool }]

   [ DOMAIN=name31 ]

   [ SOURCE={ 1 | bool }]

- - - - - - - - - - - - - - - - - - - -

   [ OLDVERS={ 0 | bool }]
```

Description of Parameters:

FORCE	*An incomplete or erroneous procedure saved using FORCE is not executable; it can only be further edited and resaved.*
=1	Save the procedure even if it is incomplete or contains errors
=0	Discard the procedure.
DOMAIN	*Default:* current domain defined by last DOMAIN command.

SOURCE	If source of procedures is stored in domain subfile of BINLIB:
`=0`	Not stored: so, PRINT, DECOMPILE and LOAD commands cannot be used. *Procedures of standard domains delivered in the library SYS.HBINLIB are compiled with `SOURCE=0`.*
`=1`	*Default:* Source lines are stored.
OLDVERS	Maximum number of procedures or aliases for different GCOS releases:
`=1`	Maximum limited to 510 and format of domain subfile is the same as that for previous V3 and V5 releases.
`=0`	*Default:* Maximum can exceed 510 fixed by GCL, in which case, format of domain subfile differs from that of previous releases, and will not be accepted by previous versions of GCL and MAINTAIN_COMMAND.

Constraints:

- Restrictions on RESAVE:
  - if FORCE is omitted, incomplete or erroneous procedures are discarded
  - the BIN library must be assigned through the BINLIB command before RESAVE can be used.
- RESAVE creates a new procedure as part of:
  - the specified domain if DOMAIN parameter is specified
  - the current domain if DOMAIN is omitted.
- The owner of a procedure defined with LOCK depends on SADMOPT options:
  - `GCLKPROJ=NO:` the **user** who compiled the procedure
  - `GCLKPROJ=YES:` the **project** of the user who compiled the procedure.
- Storage of the source code:
  - the source code for compiled procedures with its executable code is stored in SYS.HBINLIB
  - the source code for procedures compiled in standard domains is not saved in the binary library. See Section 10 of the *GCOS 7-V6 System Administrator's Manual* for the list of standard domains.

Examples:

`BINLIB BL1`	assign binary library BL1
`DOMAIN DMN`	current domain is DMN
`LD P1`	load procedure P1 into workspace
`LEDIT`	call the Line Editor
`.` }	
`.` }	sequence of Line Editor requests
`.` }	
`/`	leave the Line Editor
`RSV`	resave procedure P1 in BIN library BL1

3.8.31 RESEQUENCE (RSQ)

Purpose:

To renumber the lines in the workspace.

Syntax:

```
{ RESEQUENCE }
{            }
{ RSQ        }

   [ INIT={ 10 | dec6 }]

   [ STEP={ 10 | dec6 }]
```

Description of Parameters:

INIT	Number to be assigned to the first line. *Default:* 10
STEP	Increment to be used for numbering each new line. *Default:* 10

Constraints:

None

Examples:

`RSQ`	lines are numbered in tens
`RSQ 100`	lines are numbered in tens from 100
`RSQ 100 20`	lines are numbered in twenties from 100

3.8.32 RESET

Purpose:

Reserved for the System Administrator: To reset the original access rights and priorities of procedures in the current domain stored in the current BIN library.

Syntax:

```
RESET

   { PROCEDURES } { star62               }
   {            }={                      }
   { PROC       } {( name31 [ name31 ]...)}

   [ DOMAIN=name31 ]
```

Description of Parameters:

PROCEDURES	Star-name or list of up to eight procedure names.
DOMAIN	Name of the domain. *Default:* current domain defined by DOMAIN command.

Constraints:

- The BIN library must be assigned through the BINLIB command RESET can be used
- Star-name applies only to procedure names not to aliases.

Examples:

`BINLIB BLIB`	assign binary library BLIB
`DOMAIN D`	specify the current domain D
`RESET DC*`	reset rights and priorities for procedures whose names begin with DC
`RESET *`	reset access rights and priorities for all procedures

3.8.33 RESTORE_ACCESS (RSTA)

Purpose:

Reserved for the System Administrator and restricted to users of the SYSADMIN project: To restore the access rights previously saved in an SL library member by SAVE_ACCESS, of all procedures of a domain.

Syntax:

```
{ RESTORE_ACCESS }
{                }
{ RSTA           }

           { star62               }
   DOMAINS={                      }
           {( name31 [ name31 ]...)}

   [ PREFIX=name6 ]
   [ LIST={ NEW | ALL | NO }]
```

Description of Parameters:

DOMAINS	Star-name or list of up to eight domain names.
PREFIX	Prefix used when creating the SL library members.
LIST	Specifies the information to be listed:
=ALL	List all procedures of the domain and indicate for each procedure if access rights have been restored.
=NO	No list produced.
=NEW	*Default:* List all new procedures in the domain that did not exist on last SAVE_ACCESS.

Constraints:

BIN and SL libraries must both be assigned through BINLIB and SLLIB commands before RESTORE_ACCESS can be used.

Examples:

`BINLIB BL1`	assign binary library BL1
`SLLIB SL1`	assign source library SL1

```
RESTORE_ACCESS (IOF,H_NOCTX) PREFIX=JUNE24
```

restore access rights of domains IOF and H_NOCTX saved with command SAVE_ACCESS under the names JUNE24IOF and JUNE24H_NOCTX

3.8.34 SAVE (SV)

Purpose:

To store the contents of the workspace as a compiled procedure in the current BIN library. The compiled procedure is given the procedure name specified in the PROC command which heads the procedure.

Syntax:

```
{ SAVE }
{      }
{ SV   }

   [ FORCE={ 0 | bool }]

   [ DOMAIN=name31 ]

   [ SOURCE={ 1 | bool }]

- - - - - - - - - - - - - - - - - - -

   [ OLDVERS={ 0 | bool }]
```

Description of Parameters:

FORCE	An incomplete or erroneous procedure saved using FORCE is not executable; it can only be further edited and resaved.
=1	Save the procedure even if it is incomplete or contains errors.
=0	Discard the procedure.
DOMAIN	*Default:* current domain defined by last DOMAIN command.

SOURCE	If source of procedures is stored in domain subfile of BINLIB:
`=0`	Not stored: so, PRINT, DECOMPILE and LOAD commands cannot be used. *Procedures of standard domains delivered in the library SYS.HBINLIB are compiled with `SOURCE=0`.*
`=1`	*Default:* Source lines are stored.
OLDVERS	Maximum number of procedures or aliases for different GCOS releases:
`=1`	Maximum limited to 510 and format of domain subfile is the same as that for previous V3 and V5 releases.
`=0`	*Default:* Maximum can exceed 510 fixed by GCL, in which case, format of domain subfile differs from that of previous releases, and will not be accepted by previous versions of GCL and MAINTAIN_COMMAND.

Constraints:

See RESAVE.

Examples:

```
BINLIB BL1               assign binary library BL1

DOMAIN DMN               current domain is DMN

CREATE                   create new procedure
   10:PROC P1 ...
   20:...
   30:...
   40:/

SV                       save procedure P1 in domain DMN of BIN library BL1
```

3.8.35 SAVE_ACCESS (SVA)

Purpose:

Restricted to users of the SYSADMIN project: To save the access rights of all procedures of a domain in a member of an SL library

- defined by parameters of the basic GCL command PROC:

ACCESS	LIMITED_ACCESS	OPACC
HIDE	LOCK	PRIORITY

- and possibly modified by the commands MODIFY_ACCESS or MODIFY_LOCK.

Syntax:

```
{ SAVE_ACCESS }
{             }
{ SVA         }

           { star62                }
   DOMAINS={                       }
           {( name31 [ name31 ]...)}

   [ PREFIX=name6 ]
   [ REPLACE={ 0 | bool }]
```

Description of Parameters:

DOMAINS	Star-name or as list of up to eight domain names.
PREFIX	Prefix used when creating the SL library members. When omitted, names of members are those of domains.
REPLACE	Specifies if members being created overwrite those with the identical names in the SL library:
=1	Overwrite
=0 (*default*)	No overwrite.

Constraints:

- Restrictions on SAVE_ACCESS:
  - BIN and SL libraries must both be assigned through BINLIB and SLLIB commands before SAVE_ACCESS is used.
  - if a member of the same name exists in the SL library and is not type CMD, an error is returned and no member is created
  - if the member is type CMD, it is overwritten if `REPLACE=1`; otherwise a message is issued and no member is created.
- SAVE_ACCESS creates new SL members:
  - in the SL library specified by the SLLIB command
  - with names of the corresponding domain headed by *`prefix`* and truncated, if necessary, to 31 characters:

Examples:

`BINLIB BL1`	assign binary library BL1
`SLLIB SL1`	assign source library SL1
`SAVE_ACCESS (IOF,H_NOCTX) PREFIX=JUNE24`	
	save access rights of all procedures of domains IOF and H_NOCTX; create SL members whose names are JUNE24IOF and JUNE24H_NOCTX

3.8.36 SLLIB

Purpose:

To assign an SL library:

- if an SL library is already assigned, it is replaced by the library specified in SLLIB
- if the SLLIB command is specified without a library name, the current SL library is deassigned.

Syntax:

```
SLLIB

   [ LIBRARY=lib78 ]
```

Description of Parameter:

LIBRARY	Name of an SL library. *Default:* the current SL library.

Constraints:

If no SL library is assigned, any attempt to perform an operation which involves an SL library such as FSE will result in an error.

Examples:

`SLLIB SLIB1`	assign SL library SLIB1
`SLLIB SL2:K152:MS/D500`	assign uncataloged SL library
`SLLIB`	deassign current SL library

3.8.37 STATUS (ST)

Purpose:

To display information on the current context: assigned BIN and SL libraries, the name of the current domain, the number of lines in the workspace, and the current value of ON_ERROR.

Syntax:

```
{ STATUS }
{        }
{ ST     }
```

Parameters:

None

Constraints:

None

Example:

```
STATUS or ST                display current context
  C: ST
     SLLIB   : LINT.CUR.SLLIB$CAT (BVU630:MS/B10)
     BINLIB  : LINT.TEST.TBINLIB$CAT (BVU630:MS/B10)
     WORKSPACE IS EMPTY
     CURRENT DOMAIN : CMDMGT
     ON_ERROR ACTION = DEFAULT
```

❑

4. Access to GCOS Files through GCL

The set of *GCOS File Access* commands allows:

- executing GCL directives
- and accessing GCOS 7
  - UFAS sequential, relative and indexed disk files
  - libraries
  - and tape files.

See Paragraph *"GCOS File Access Commands"* for details on these GCL commands.

4.1 Files

Each file to be accessed must have been:

- previously declared by the DECLARE_FILE command which associates:
  - the *efn* (External File Name) of the file
  - with its *sfn* (Symbolic File Name) given by the user
- and opened by the OPEN_FILE command which specifies whether the file is to be read from or written to.

An *sfn* once assigned, cannot be assigned to another file until a RELEASE_FILE command is executed. Further access to the file is made by simply specifying its *sfn*. All other commands except EXIST_FILE, use the *sfn* to refer to the file.

GCL associates:

- the *sfn* of the file
- with its *ifn* (Internal File Name) which serves as the interface between GCL and the access method.

Up to 800 files may be simultaneously declared. The file remains open:

- until a CLOSE_FILE or RELEASE_FILE command is executed
- or until job termination.

4.2 Command Parameters

Unless otherwise stated, input parameters may receive either a literal value or the contents of a GCL variable. GCL variable names must be different from the keyword names of GCL procedures.

4.3 Completion Codes

The completion codes of all command functions (except LIST_DECLARED_FILE) are returned to the variable specified in the STATUS parameter. This variable must be declared with TYPE=DEC and LENGTH=3, except for the EXIST_FILE command where the declaration is TYPE=CHAR and LENGTH=32.

Parameter STATUS is mandatory in batch mode. In interactive mode, if STATUS is omitted, abnormal completion codes will be displayed on the user's terminal.

4.4 Address Format

For UFAS sequential and relative disk files, the address format specified in the ADDR parameter must be the logical record number. For UFAS indexed disk files, in direct access mode `ACCMODE=D`, only the key may be used. For libraries and tape files, direct access is not authorized. The first logical record is record 1.

4.5 Temporary Files

Since most files to be accessed already exist, their organization is defined and will be automatically retrieved. Temporary files or sequential tape files will be created as UFAS sequential files with RECSIZE=255, CISIZE=2048, and RECFORM=FB. Subfiles of SL libraries will be created. Dummy files are not supported.

4.6 Types of Access

The files to be processed depend on the type of access:

SEQUENTIAL	For access to: – UFAS sequential, indexed and relative disk files – libraries – and tape files.
DIRECT	For UFAS sequential, indexed and relative disk files.

4.7 Access Requirements

UFAS disk files, libraries, and tape files have specific access requirements:

UFAS Sequential and Relative Disk Files

Records can be accessed sequentially or directly by using the record address. The record address is the logical record number of the record in the file. However, for UFAS sequential disk files, direct access mode is not authorized for files with RECFORM=VB.

UFAS Indexed Disk Files

Records can be accessed sequentially or directly by using the key. The key is part of the record, and its location and length are characteristics of the file. No two records can have the same key. Both primary and secondary keys are supported. A record may have up to 32767 characters. The key up to 251 characters.

When a UFAS indexed file with secondary keys is created with OUTPUT processing mode, it must first be closed and sorted with the command "SRTIDX filename", before being opened in UPDATE processing mode.

Libraries

The access mode must be sequential.

The commands MODIFY_RECORD and DELETE_RECORD are not authorized. In input mode, SSF format is not returned to the user. In output mode, no SSF is created.

For non-SL libraries, the access mode must be READ, SPREAD, or ALLREAD. Only read operations are authorized.

Tape Files

The access mode must be sequential.

The commands POINT_RECORD, DELETE_RECORD, and MODIFY_RECORD are not authorized.

4.8 Break Processing

GCOS file access commands can be used to create GCL procedures that can be executed in batch mode or in interactive mode. The Break key or equivalent can be used, in interactive mode to interrupt these procedures, provided the procedures can manage the interrupts. The capability to do so is given to the GCL procedures when they are created if required.

Two system variables are available:

#BRKPMODE (BReaK Processing MODE)

Type=char
Length=1

The #BRKPMODE variable may be set to 0 or 1 by using the following commands:

```
LET #BRKPMODE={ 0 | 1 }
```

or

```
MP BRKPMODE={ 0 | 1 }
```

The setting of PRKPMODE determines which can and does manage the Break interrupts:
- if 1 is specified, GCL procedures
- if 0 is specified, the GCL processor.

#BRK (BReaK)

Type=char
Length=1

#BRK may be set to 0 or 1. The value 0 indicates that no interrupt request has been processed.

In the following example:
- the user sets and resets the #BRK variable to 0 with "LET #BRK 0."
- #BRK is set to 1 when the following series of actions occur:
  1) variable #BRKPMODE has been set to 1
  2) the Break key or equivalent has generated an interrupt request
  3) the "???" message has been answered by "/" or "IT".

4.9 Example of Procedure Using Break Processing

The PRINT_SOURCE procedure accesses GCOS files through GCL. It prints the contents of a sequential file in batch or interactive mode at either the "S:" or "C:" level. It also manages Break interrupts: the variable #BRKPMODE is set to 1 at the start of the procedure. If a Break interrupt occurs (if #BRK=1), PRINT_SOURCE stops reading the file, closes then releases it, sets #BRKPMODE to 0, and ends. If a GCOS file access command terminates abnormally, the EDFERR command is used to print the error message.

```
COMM '**************************************************';
COMM '    PROCEDURE PRINT_SOURCE                        ';
COMM 'PRINT SEQUENTIAL FILES IN SOURCE FORMAT OR        ';
COMM 'LIBRARY MEMBERS.  RECORDS GREATER THAN 255        ';
COMM 'BYTES WILL BE TRUNCATED.                          ';
COMM '**************************************************';
PROC NAME=(PRINT_SOURCE,PRS)
       PROMPT='PRINT A SOURCE FILE';
KWD  NAME=FILE_NAME
       TYPE=FILE
       MSSG='ERRONEOUS VALUE FOR FILE NAME'
       PROMPT='FILE TO BE PRINTED';
LOCAL ST DEC 5;
LOCAL WAZ CHAR 255;
LOCAL LG DEC 5;
LOCAL NM CHAR 15;
LET #BRKPMODE 1;
LET #BRK 0;
LET NM '';
LET # #CAT('FILE:',%FILE_NAME);
LET # '';
DCLF FIL,FILE=%FILE_NAME,ACCESS=READ,STATUS=ST;
IF #NE(%ST,0);
       EDFERR SFN=FIL,COMMAND='DCLF',ERROR=%ST
       GOTO RESET_BREAK;
ENDIF;
OPENF FIL,PMD=IN,ACCMODE=S,STATUS=ST;
IF #NE(%ST,0);
       EDFERR SFN=FIL,COMMAND='OPENF',ERROR=%ST;
       GOTO RELEASE_FILE;
ENDIF;
LABEL LOOP;
IF #EQ(#BRK,1);
       GOTO CLOSE_FILE;
```

```
ENDIF;
RDREC FIL,WA=WAZ,LENGTH=LG,STATUS=ST;
IF #OR(#EQ(%ST,0) #EQ(%ST,1));
       LET # #SUBSTR(%WAZ,1,%LG);
       GOTO LOOP;
ENDIF;
IF #NE(%ST,2);
       EDFERR SFN=FIL,COMMAND='RDREC',ERROR=%ST;
ENDIF;
LABEL CLOSE_FILE;
CLOSEF FIL,STATUS=ST;
LABEL RELEASE_FILE;
RLSF FIL,STATUS=ST;
LABEL RESET_BREAK;
LET #BRKPMODE 0;
LET #BRK 0;
ENDPROC;
```

4.10 GCOS File Access Commands

GCOS File Access commands are:

• BUILD_RECORD	move a GCL variable into the buffer attached to a file.
• CLOSE_FILE	close an opened file referenced by its *sfn*.
• DECLARE_FILE	declare a file by associating an *sfn* with a file description.
• DELETE_RECORD	delete a record in UFAS disk files.
• EDIT_FILE_ERROR	display information about the last error on a file.
• EXIST_FILE	determine whether a file exists.
• LIST_DECLARED_FILE	display information about the user's declared files.
• MODIFY_RECORD	overwrite a record in a UFAS disk file.
• OPEN_FILE	open disk files, tape files, and libraries in the specified mode.
• POINT_RECORD	point to a specific record in a disk file.
• READ_RECORD	read a record from a declared and opened file, and store it in a buffer or GCL variable.
• RELEASE_FILE	release a file referenced by its *sfn*.
• RETURN_DECLARED_FILE	retrieve the description of a declared file.
• SPLIT_RECORD	extract a character string from an input buffer and store it in a GCL variable.
• WRITE_RECORD	get a record from a buffer or GCL variable and write it to a file.

4.10.1 BUILD_RECORD (BREC)

Purpose:

Moves a GCL variable into the buffer attached to a UFAS sequential, relative, or indexed disk file, library, or tape file.

Syntax:

```
{ BUILD_RECORD }
{              }
{ BREC         }

          SFN = name16

          [ WA = name31 ]

          INDEX = dec5

          [ LENGTH = dec5 ]

          [ STATUS = name31 ]
```

Parameters:

SFN	the symbolic file name of the file.
WA	the working area. The working area specified is the name of a global or local GCL variable. The record is retrieved from the variable and stored in the buffer attached to the file. WA is mandatory if INDEX is not equal to 0.
INDEX	the rank (position) in the buffer where the first character of the retrieved character string will be stored. INDEX values must be in the range 0 to 32767 (the maximum record length). The first character is character number 1. If INDEX=0, the buffer is initialized with blanks; in this case, WA and LENGTH are not used.
LENGTH	the length of the string to be moved to the buffer. If LENGTH is not specified and WA is specified, LENGTH defaults to the length of the working area. If INDEX is equal to 0, LENGTH defaults to the length of the buffer.

STATUS	the name of a GCL variable that will receive the completion code for BUILD_RECORD. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for BUILD_RECORD are: Normal: 0: Normal execution of command. Abnormal: 256: Wrong symbolic file name. 260: File not opened. 264: Wrong processing mode. 265: System error: argument error. 266: System error: file structure not available. 270: System error: buffer pointer not available. 278: Overflow.

Constraints:

- If LENGTH is greater than the data length in WA, the buffer is padded with blanks.
- If INDEX + LENGTH is greater than the size of the buffer, completion code 278 ("Overflow") is returned and the command is not executed.
- STATUS must be used in Batch Mode.

Examples:

```
BREC SFN=MYFILE0002 WA=BRECWAVAR INDEX=50 LENGTH=250
     STATUS=BRECVAR
          {move string in GCL variable BRECWAVAR to buffer
           attached to MYFILE0002; string is 250 characters
           long; first character to be written to rank
           (position) 50 of buffer; GCL variable BRECVAR will
           receive completion code}

BREC SFN=MYFILE0018 INDEX=0 STATUS=BRECVAR
          {the buffer attached to MYFILE0018 will be initialized
           with blanks; GCL variable BRECVAR will receive the
           completion code}

BREC SFN=MYFILE0018 WA=BRECWAVAR INDEX=0 LENGTH=4000
     STATUS=BRECVAR
          {same as above; WA and LENGTH are ignored}
```

4.10.2 CLOSE_FILE (CLOSEF)

Purpose:

Closes an opened file referenced by its symbolic file name (SFN).

Syntax:

```
{ CLOSE_FILE }
{            }
{ CLOSEF     }

          { ( name16 [ name16 ] ... ) }
    SFN = {                           }
          { star36                    }

    [ STATUS = name31 ]
```

Parameters:

SFN	a symbolic file name, a list of up to 6 SFNs, or a name using the star convention.
STATUS	the name of a GCL variable that will receive the completion code for CLOSE_FILE. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for CLOSE_FILE are: Normal: 0: Normal execution of command. Abnormal: 256: Wrong symbolic file name. 257: No declared file. 260: File not opened. 265: System error: argument error. 266: System error: file structure not available. 270: System error: buffer pointer not available. 300: Abnormal return code from system primitive.

Constraints:

STATUS must be used in Batch Mode.

Examples:

```
CLOSEF SFN=(MYFILE0008,MYFILE0020) STATUS=CLOSEFVAR
          {close the files identified by SFNs MYFILE0008 and
           MYFILE0020; GCL variable CLOSEFVAR will receive the
           completion code}

CLOSEF SFN=MYFILE* STATUS=CLOSEFVAR
          {as above, but for all SFNs beginning with MYFILE}

CLOSEF * STATUS=CLOSEFVAR
          {as above, but for all user SFNs}
```

4.10.3 DECLARE_FILE (DCLF)

Purpose:

Declares UFAS sequential, relative, or indexed disk files; libraries; or sequential tape files by associating a symbolic file name (SFN) with the file description. Does not check whether the file exists and does not assign it.

Syntax:

```
{ DECLARE_FILE }
{              }
{ DCLF         }

          SFN = name16

          FILE = file78

          ACCESS = { WRITE|READ|SPREAD|SPWRITE|ALLREAD|RECOVERY }

          [ SHARE = { NORMAL|ONEWRITE|FREE|DIR|MONITOR } ]

          [ STATUS = name31 ]
```

Parameters:

SFN	symbolic file name to be associated with the file. The other commands will specify this declared SFN to access the file.
FILE	formal description of the file, according to the following syntax:

```
external_file_name[..subfile_name][:media:dvc][$attributes]
```

ACCESS	the access mode for the program. The allowed values are:
= WRITE	allows input, output, append, and update modes. This is the default value (see Constraints) .
= READ	allows input mode only.
= SPREAD	same as READ, but additionally the current program has exclusive file control, regardless of the SHARE value.

	`= SPWRITE`	same as WRITE, but additionally the current program has exclusive file control, regardless of the SHARE value.
	`= ALLREAD`	no output allowed; only simultaneous input
	`= RECOVERY`	applies only to cataloged files. The program has exclusive access for file recovery purposes.
SHARE		the maximum allowable level of concurrent file access:
	`= NORMAL`	one writer or several readers.
	`= ONEWRITE`	one writer and several readers.
	`= FREE`	no restriction on access.
	`= DIR`	applies only to libraries. Each member can be concurrently accessed by several readers or one writer.
	`= MONITOR`	several readers. The accesses to the files are checked and synchronized when necessary by GAC (General Access Control)

For a library file, cataloged or uncataloged, when the subfile name is specified, the default value for SHARE is DIR. If the file is not a library file, the default value for SHARE is NORMAL (see Constraints).

The sharing of a file becomes effective once the file is assigned. A file processed by GCL is:

- assigned by the command OPEN_FILE.
- deassigned explicitly by commands CLOSE_FILE and RELEASE_FILE, or implicitly at the end of a step or at the end of a job. When a file is deassigned at the end of a step, it remains deassigned until the next command that accesses the file (READ_RECORD, WRITE_RECORD, etc.) is executed.

Example:

```
S: DCLF F1...           +----------------------------+
S: .....                | F1 not assigned            |
S: OPENF F1...          +----------------------------+
S: .....                |                            |
S: .....                | F1 assigned                |
S: .....                |                            |
S: CLOSEF F1...         +----------------------------+
S: .....                | F1 deassigned              |
S: OPENF F1...          +----------------------------+
S: .....                | F1 assigned                |
S: MNLIB SL...          +----------------------------|
C: .....                | F1 deassigned              |
C: RDREC F1...          +----------------------------|
C: .....                | F1 assigned                |
C: /                    +----------------------------+
S: .....                | F1 deassigned              |
S: .....                +----------------------------+
```

STATUS	the name of a GCL variable that will receive the completion code for DECLARE_FILE. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for DECLARE_FILE are:

Normal:
0: Normal execution of command.

Abnormal:
256: Wrong symbolic file name.
265: System error: argument error.
266: System error: file structure not available.
267: Declared files table overflow.
272: System error: semaphore for file not available.
282: Functionality not available now.
300: Abnormal return code from system primitive.

Constraints:

- For cataloged files, the default values for parameters ACCESS and SHARE can be specified in the catalog. When this is the case, both the default values for DECLARE_FILE and those values given explicitly are either ignored or interpreted in a particular way, according to the method of access. The effective values of ACCESS and SHARE are discussed in the *Data Management Utilities User's Guide*.
- SHARE=MONITOR may be used only if ACCESS=READ or SPREAD or ALLREAD.
- STATUS must be used in Batch Mode.

Examples:

```
DCLF SFN=MYFILE0003 FILE=TESTFILE$RES ACCESS=READ
     SHARE=NORMAL STATUS=DCLFVAR
          {declare resident file TESTFILE with SFN MYFILE0003;
           access input only; maximum level concurrent access is
           one writer or several readers; DCLFVAR will receive
           completion code}

DCLF SFN=MYFILE0014 FILE=CHECKFILE:VOL5:MS/D500 SHARE=ONEWRITE
     STATUS=DCLFVAR
          {declare file CHECKFILE with SFN MYFILE0014; ACCESS
           defaults to WRITE; SHARE specifies one writer and
           several readers; DCLFVAR will receive the completion
           code}
```

4.10.4 DELETE_RECORD (DLREC)

Purpose:

Deletes a record in UFAS disk files.

Syntax:

```
{ DELETE_RECORD }
{               }
{ DLREC         }

          SFN = name16

          [ KEY = char251 ]

          [ ADDR = dec10 ]

          [ STATUS = name31 ]

          [ HEXA_KEY = char252 ]
```

Parameters:

SFN	the symbolic file name of the file.
KEY	key value of the record to be deleted. If no record exists with the given key value, an error occurs. KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).
ADDR	for UFAS relative disk files, the address of the record to be deleted. The file must be opened in direct access mode (ACCMODE=D).

STATUS	the name of a GCL variable that will receive the completion code for DELETE_RECORD. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for DELETE_RECORD are: Normal: 0: Normal execution of command. Abnormal: 256: Wrong symbolic file name. 260: File not opened. 261: No current record. 264: Wrong processing mode. 265: System error: argument error. 266: System error: file structure not available. 270: System error: buffer pointer not available. 273: Wrong parameter (FIRST, KEY, HEXA_KEY or ADDR) for this organization. 279: Unauthorized command for this type of file. 280: Wrong access mode. 281: ADDR or FIRST unauthorized. 300: Abnormal return code from system primitive.
HEXA_KEY	key value in hexadecimal of the record to be deleted. If no record exists with the given key value, an error occurs. HEXA_KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).

Constraints:

- The file must be opened in UP (update) processing mode.
- KEY, HEXA_KEY and ADDR are mutually exclusive.
- UFAS Sequential Disk Files

  The last issued command must have been a successful READ_RECORD. The record retrieved by READ_RECORD is deleted. KEY, HEXA_KEY and ADDR are not allowed.
- UFAS Relative Disk Files

  If ADDR is specified, DELETE_RECORD deletes the record whose direct address is given. If ADDR is not specified, the last issued command must have been a successful READ_RECORD; the record retrieved by READ_RECORD is deleted; the current record pointer is unchanged. (There is no new current record.)
- UFAS Indexed Disk Files

  If KEY or HEXA_KEY is specified, DELETE_RECORD deletes the record whose key is given. Only the primary key may be used, even though the file may have been declared as indexed with a secondary key. If neither KEY nor HEXA_KEY is specified, the last issued command must have been a successful READ_RECORD; the record retrieved by READ_RECORD is deleted; the current record pointer is unchanged (no new current record).

  Maximum key length when using HEXA_KEY is 126 bytes.
- Libraries and Tape Files

  DELETE_RECORD is not authorized.
- STATUS must be used in Batch Mode.

Examples:

```
DLREC SFN=MYFILE0002 ADDR=874 STATUS=DLRECVAR
          {delete record at address 874 in UFAS relative disk
           file identified by SFN MYFILE0002; GCL variable
           DLRECVAR will receive the completion code}

DLREC SFN=MYFILE0020 KEY=4007832215 STATUS=DLRECVAR
          {delete the record whose key is 4007832215 in UFAS
           indexed disk file identified by SFN MYFILE0020; GCL
           variable DLRECVAR will receive the completion code}
```

4.10.5 EDIT_FILE_ERROR (EDFERR)

Purpose:

Returns information on the last error that has occurred on a file. The information is displayed on the active output device (user's terminal or SYSOUT), or, if the WA parameter is specified, is placed in the working area. This command cannot be used after the commands EXIST_FILE, LIST_DECLARED_FILE, RETURN_DECLARED_FILE, and EDIT_FILE_ERROR.

Syntax:

```
{ EDIT_FILE_ERROR }
{                 }
{ EDFERR          }

          [ SFN = name16 ]

          [ COMMAND = char10 ]

          [ ERROR = dec3 ]

          [ WA = name31 ]

          [ STATUS = name31 ]
```

Parameters:

SFN	the symbolic file name of the file. If SFN is specified, EDFERR will return the symbolic file name. If both SFN and ERROR are specified, the SFN and the edited form of the given ERROR value are displayed on the active output device (user's terminal or SYSOUT). If SFN is specified and ERROR is not specified, EDFERR returns the SFN and the completion code (in edited format) associated with the last command applied to the file.
COMMAND	specifies a message provided by the user. This message will be displayed on the active output device.

ERROR	specifies a completion code. When ERROR is present, EDFERR will return the literal corresponding to the given completion code. If, for example, ERROR=300 and SFN are specified, EDFERR will return the SFN and the G4 value in edited format. The following G4 value is in edited format: RC=98051813 -> GCL 5, CDERR
WA	the working area. WA is the name of a local or global GCL variable, declared with TYPE=CHAR and LENGTH=126. If specified, WA will contain a character string that is the concatenation of all strings returned to the working area. This character string stored in WA will have the following format, with blanks present when information is not supplied:

```
  SFN         COMMAND         ERROR                G4
                        (edited format)     (edited format)
  ---        ---------    ---------------     ---------------
16 chars      10 chars        70 chars            30 chars
```

	If WA is not specified, the information will be displayed on the active output device (user's terminal or SYSOUT).
STATUS	the name of a GCL variable that will receive the completion code for EDIT_FILE_ERROR. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for EDIT_FILE_ERROR are: Normal: 0: Normal execution of command. Abnormal: 256: Wrong symbolic file name. 265: System error: argument error. 266: System error: file structure not available.

Constraints:

EDFERR cannot be used after commands EXIST_FILE, LIST_DECLARED_FILE, RETURN_DECLARED_FILE, and EDIT_FILE_ERROR.

STATUS must be used in Batch Mode.

Examples:

```
EDFERR SFN=MYFILE0008 COMMAND=WRREC_ERR ERROR=300
       STATUS=EDFERRVAR
          {SFN, G4 value in edited format, and "WRREC_ERR"
           message will be displayed on the active output
           device; GCL variable EDFERRVAR will receive the
           completion code for EDFERR}

EDFERR ERROR=256 WA=EDFERRWAVAR STATUS=EDFERRVAR
          {literal corresponding to completion code 256 will be
           written to the working area EDFERRWAVAR; variable
           EDFERRVAR will receive the completion code for EDFERR}
```

4.10.6 EXIST_FILE (EXISTF)

Purpose:

Determines whether a file exists.

Syntax:

```
{ EXIST_FILE }
{            }
{ EXISTF     }

          FILE = file78

          [ STATUS = name31 ]
```

Parameters:

FILE the file description of the file, using the GCL syntax:

```
  external_file_name[..subfile_name][:media:dvc][$attributes]
```

STATUS the name of a GCL variable that will receive the result of the command. The variable must be declared with TYPE=CHAR and LENGTH=32. The result of EXIST_FILE is:

```
    Byte 1    Bytes 2-32             State
    ------    ----------             -----
      0       Spaces                 The file does not exist.

      1       Spaces                 The file exists.

      2       Abnormal return code   Unknown state.
              in edited format.
```

Constraints:

STATUS must be used in Batch Mode.

Examples:

```
EXISTF FILE=TESTFILE$RES STATUS=EXISTFVAR
          {EXISTF determines whether resident file TESTFILE
           exists, and writes the result to GCL variable
           EXISTFVAR}

EXISTF FILE=TESTFILE..SUBTEST2:VOL1:MS/D500 STATUS=EXISTFVAR
          {EXISTF determines whether subfile SUBTEST2 of file
           TESTFILE exists, and writes the result to EXISTFVAR}
```

4.10.7 LIST_DECLARED_FILE (LSDCLF)

Purpose:

Displays information about declared user files. This information is displayed on the active output device, either the user's terminal or SYSOUT.

Syntax:

```
{ LIST_DECLARED_FILE }
{                    }
{ LSDCLF             }

                  { ( name16 [ name16 ] ...) }
          [ SFN = { star36                   } ]
                  { *                        }

          [ FILE = bool ]

          [ IFN = bool ]

          [ STATE = bool ]

          [ ORG = bool ]

          [ PMD = bool ]

          [ ACCMODE = bool ]
```

Parameters:

SFN	a symbolic file name, a list of up to 6 SFNs, a name using the star convention, or a " * " (default) for all SFNs.
FILE	the file description to be displayed. If FILE=1, the description is displayed; if FILE=0, it is not displayed.
IFN	specifies whether the Internal File Name is to be displayed. If IFN=1, the IFN is displayed; if IFN=0, it is not displayed.
STATE	specifies whether the file state (OPENED, CLOSED, or UNKNOWN) is to be displayed. If STATE=1, the file state is displayed; if STATE=0, it is not displayed.

ORG	specifies whether the file organization (SEQUENTIAL, INDEXED, RELATIVE, QUEUED, or NONE) is to be displayed. (QUEUED is for libraries.) If ORG=1, the organization is displayed; if ORG=0, it is not displayed.
PMD	specifies whether the processing mode (INPUT, OUTPUT, APPEND, UPDATE, or NONE) is to be displayed. If PMD=1, the processing mode is to be displayed; if PMD=0, it is not displayed.
ACCMODE	specifies whether the access mode (SEQUENTIAL, DIRECT, or NONE) is to be displayed. If ACCMODE=1, the access mode is displayed; if ACCMODE=0, it is not displayed.

Constraints:

- If only SFN is specified, then all information about the files is displayed.
- If SFN and at least one other parameter are specified, all other unspecified parameters are ignored.
- Completion codes for LIST_DECLARED_FILE are the following:

Normal:
0: Normal execution of command.

Abnormal:
257: No declared file.
265: System error: argument error.
266: System error: file structure not available.

Examples:

```
LSDCLF SFN=(MYFILE0002,MYFILE0009) IFN=1 ORG=1 ACCMODE=1
          {displays for the files identified by SFNs MYFILE0002
           and MYFILE0009 the IFNs, file organizations, and
           access modes}

LSDCLF SFN=MYFILE* FILE STATE PMD
          {displays for all files whose SFNs begin with MYFILE
           the file descriptions, file states, and processing
           modes}

LSDCLF    {the SFN parameter defaults to all SFNs; since no
           optional parameter is specified, all information
           for all files is displayed}
```

4.10.8 MODIFY_RECORD (MDREC)

Purpose:

Modifies (overwrites) a record in a UFAS disk file. The file must be opened in UP (update) processing mode. This command is not authorized for libraries or tape files.

Syntax:

```
{ MODIFY_RECORD }
{               }
{ MDREC         }

          SFN = name16

          [ KEY = char251 ]

          [ ADDR = dec10 ]

          [ WA = name31 ]

          [ LENGTH = dec5 ]

          [ STATUS = name31 ]

          [ HEXA_KEY = char252 ]
```

Parameters:

SFN	the symbolic file name of the file.
KEY	the key value of the record to be modified. If no record exists with the given key value, an error occurs. KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).
ADDR	the record address of the record to be modified for UFAS relative disk files. The file must be opened in direct access mode (ACCMODE=D).

WA	the working area. The working area specified is the name of a global or local GCL variable. If WA is specified, the record is taken from the variable and written to the file. If WA is not specified, the record is taken from the buffer attached to the file; in this case, the contents of the buffer must have been previously initialized, using the BUILD_RECORD command.
LENGTH	the length of the record to be modified. The length of the new record must be the same as the length of the old one. The default value is the maximum record size of the file.
STATUS	the name of a GCL variable that will receive the completion code for MODIFY_RECORD. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for MODIFY_RECORD are: Normal: 0: Normal execution of command. 4: Synonym. Abnormal: 256: Wrong symbolic file name. 260: File not opened. 261: No current record. 262: Data length error. 264: Wrong processing mode. 265: System error: argument error. 266: System error: file structure not available. 270: System error: buffer pointer not available. 273: Wrong parameter (FIRST, KEY, HEXA_KEY, or ADDR) for this organization. 275: Attempt to change existing record key. 276: Duplicate secondary key. 279: Unauthorized command for this type of file. 280: Wrong access mode. 281: ADDR or FIRST unauthorized. 300: Abnormal return code from system primitive.
HEXA_KEY	key value in hexadecimal of the record to be modified. If no record exists with the given key value, an error occurs. HEXA_KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).

Constraints:

- The file must be opened in UP (update) processing mode.
- The length of the new record must be the same as the length of the old one.
- KEY, HEXA_KEY and ADDR are mutually exclusive.
- UFAS Sequential Disk Files.

  The last issued command must have been a successful READ_RECORD. The record retrieved by READ_RECORD is replaced by the new record. KEY, HEXA_KEY and ADDR are not authorized.
- UFAS Relative Disk Files

  If ADDR is specified (and ACCMODE=D), MDREC modifies and replaces the record whose direct record address is given by ADDR. The new record becomes the current record. If ADDR is not specified (and ACCMODE=S or ACCMODE=D), the last issued command must have been a successful READ_RECORD; the record retrieved by READ_RECORD is replaced by the new record; the new record becomes the current record.
- UFAS Indexed Disk Files

  If KEY or HEXA_KEY is specified, MDREC modifies and replaces the record whose key is given. The key value of the new record must be the same as the key value of the old record. If neither KEY nor HEXA_KEY is specified, the last command must have been a successful READ_RECORD; the record retrieved by READ_RECORD is replaced by the new record; the new record becomes the current record. Again, the key value of the new record must be the same as the key value of the old record.

  The secondary key within the record being written can be different from that of the record being replaced. If an invalid duplicate value of a modified secondary key is found, a DUPLICATE SECONDARY KEY completion code is sent and the record is not replaced. If the duplicate value is allowed, a SYNONYM completion code is sent and the record is replaced. The order of sequential retrieval of such a record, using a modified secondary key as a reference, is the order in which it was delivered to UFAS. The new record becomes the current record.
- Libraries and Tape Files

  MODIFY_RECORD is not authorized.
- STATUS must be used in Batch Mode.

Examples:

```
MDREC SFN=MYFILE0009 ADDR=874 WA=MDRECWAVAR LENGTH=250
      STATUS=MDRECVAR
          {modify the record at address 874 in UFAS relative
           file identified by SFN MYFILE0009; new record is
           retrieved from GCL variable MDRECWAVAR; write record
           of 250 characters to file; GCL variable MDRECVAR will
           receive the completion code}

MDREC SFN=MYFILE0002 KEY=4007832215 WA=MDRECWAVAR LENGTH=4000
      STATUS=MDRECVAR
          {modify record whose KEY is 4007832215 in UFAS indexed
           file identified by SFN MYFILE0002; new record is
           retrieved from GCL variable MDRECWAVAR; write record
           of 4000 characters to the file; completion code to
           MDRECVAR}
```

4.10.9 OPEN_FILE (OPENF)

Purpose:

Opens sequential, relative, or indexed disk files, and sequential library and tape files in the specified mode. An opened file can be read, written to, or modified (except for tapes and libraries, which cannot be modified) until it is closed by a CLOSE_FILE or RELEASE_FILE command, or the job terminates.

Syntax:

```
{ OPEN_FILE }
{           }
{ OPENF     }

          SFN = name16

          PMD = { IN | OU | AP | UP }

          ACCMODE = { S | D }

          [ STATUS = name31 ]

          [ BPB = dec3 ]
```

Parameters:

SFN	the symbolic file name of the file.
PMD	the processing mode. The following types of access are allowed:
= IN	Read access.
= OU	Write access.
= AP	Append access: append (write) to the end of file. AP may not be used with UFAS relative and indexed files.
= UP	Read and write access. Modification of a record. UP may not be used with libraries and tape files.

The following series of tables shows the authorized GCL commands according to PMD value:

UFAS Sequential Disk Files

	Processing Mode (PMD)			
	IN	OU	AP	UP
READ_RECORD	X			X
WRITE_RECORD		X	X	
MODIFY_RECORD				X
DELETE_RECORD				X
POINT_RECORD	X (*)			X

* (Allowed only if RECFORM=F or FB.)

UFAS Relative Disk Files

	Processing Mode (PMD)			
	IN	OU	AP	UP
READ_RECORD	X			X
WRITE_RECORD		X		X (*)
MODIFY_RECORD				X
DELETE_RECORD				X
POINT_RECORD	X			X

* (Allowed in UP only if ACCMODE=D was specified in OPENF.)

UFAS Indexed Disk Files

	Processing Mode (PMD)			
	IN	OU	AP	UP
READ_RECORD	X			X
WRITE_RECORD		X		X (*)
MODIFY_RECORD				X
DELETE_RECORD				X
POINT_RECORD	X			X

* (Allowed in UP only if ACCMODE=D was specified in OPENF.)

Libraries

	Processing Mode (PMD)			
	IN	OU	AP	UP
READ_RECORD	X			
WRITE_RECORD		X	X	
POINT_RECORD	X			

Tape Files

	Processing Mode (PMD)			
	IN	OU	AP	UP
READ_RECORD	X			
WRITE_RECORD		X	X	

ACCMODE	the access mode. Two access modes are allowed:
= S (Sequential)	Records are accessed sequentially. In a sequential file, access is according to physical sequence. In a relative file, access is according to record number sequence. In an indexed file, access is by key sequence.
= D (Direct)	Records are accessed directly, by key or by address.
STATUS	the name of a GCL variable that will receive the completion code for OPEN_FILE. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for OPEN_FILE are:

Normal:
0: Normal execution of command.

Abnormal:
256: Wrong symbolic file name.
258: Unknown external file name or subfile name.
259: File or subfile already opened.
263: File is in an unstable state.
264: Wrong processing mode.
265: System error: argument error.
266: System error: file structure not available.
268: Wrong organization.
280: Wrong access mode.
283: File busy.
300: Abnormal return code from system primitive.

BPB	(Blocks Per Buffer) meaningful for UFAS files. Indicates the number of CIs to be read in one I/O operation.

Constraints:

- Libraries and tape files must be opened in sequential access mode.
- When files are opened with IN or UP processing mode, the current record will be the first record of the file. If the file is empty, the next READ_RECORD will return a DATALIM completion code.
- UP processing mode may not be used with libraries or tape files.
- When the value for SHARE is MONITOR, the only authorized processing mode is IN.
- STATUS must be used in Batch Mode.

Examples:

```
OPENF SFN=MYFILE0002 PMD=IN ACCMODE=S STATUS=OPENFVAR
          {open file identified by SFN MYFILE0002 in input
           processing mode and sequential access mode; GCL
           variable OPENFVAR will receive the completion
           code}

OPENF SFN=MYFILE0018 PMD=UP ACCMODE=D STATUS=OPENFVAR
          {open file identified by SFN MYFILE0018 in update
           processing mode and direct access mode; GCL
           variable OPENFVAR will receive the completion
           code}
```

4.10.10 POINT_RECORD (PTREC)

Purpose:

Points to a record in a disk file. This command cannot be used with tape files.

Syntax:

```
{ POINT_RECORD }
{              }
{ PTREC        }

          SFN = name16

          [ ADDR = dec10 ]

          [ KEY = char251 ]

          [ KEYLOC = dec5 ]

          [ KEYSIZE = dec3 ]

          [ FIRST = bool ]

          [ COND = { EQ | GE | GT } ]

          [ STATUS = name31 ]

          [ HEXA_KEY = char252 ]
```

Parameters:

SFN	the symbolic file name of the file.
ADDR	the address of a record in the file. This direct address and the comparison condition supplied by the COND parameter are used to locate the record to be pointed to. The address of the first record is 1. ADDR cannot be used with libraries and UFAS indexed disk files.

KEY		the key value of the record to be pointed to. The key value and the comparison condition supplied by the COND parameter are used to locate the record to be pointed to. This record is the first record that satisfies the comparison condition. KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).
KEYLOC		the key offset from the beginning of the record. The first byte is byte 0. When KEYLOC is omitted, the reference key is the primary key. This parameter is used with UFAS indexed disk files and only has meaning when KEY or HEXA_KEY is present. KEYLOCK can take a value in the range 0 to 32766.
KEYSIZE		the length in bytes of the partial key. The partial key is the leftmost part of the complete key specified by the KEY parameter. When KEYSIZE is present, the partial key is used in the comparison defined by the COND parameter. The default value is the length of the complete key. KEYSIZE is used with UFAS indexed disk files and only has meaning when KEY or HEXA_KEY is present. KEYSIZE can take a value in the range 1 to 251.
FIRST		identifies the record pointed to as the first record of the file. FIRST cannot be used with UFAS indexed disk files. Libraries must be opened in sequential access mode (ACCMODE=S); only the FIRST parameter may be specified.
COND		the comparison conditions for sequential, relative, and indexed disk files. These conditions are the following:
	`= EQ`	the key of the current record is equal to KEY or HEXA_KEY, or the direct record address is equal to ADDR. This is the default value.
	`= GE`	the key of the current record is greater than or equal to KEY or HEXA_KEY, or the direct record address is greater than or equal to ADDR.
	`= GT`	the key of the current record is greater than KEY or HEXA_KEY, or the direct record address is greater than ADDR

STATUS	the name of a GCL variable that will receive the completion code for POINT_RECORD. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for POINT_RECORD are: Normal: 0: Normal execution of command. Abnormal: 256: Wrong symbolic file name. 260: File not opened. 261: No current record. 264: Wrong processing mode. 265: System error: argument error. 266: System error: file structure not available. 273: Wrong parameter (FIRST, KEY, HEXA_KEY or ADDR) for this organization 277: Key length error. 279: Unauthorized command for this type of file. 280: Wrong access mode. 281: ADDR or FIRST unauthorized. 300: Abnormal return code from system primitive.
HEXA_KEY	the key value in hexadecimal of the record to be pointed to. The key value and the comparison condition supplied by the COND parameter are used to locate the record to be pointed to. This record is the first record that satisfies the comparison condition. If no record exists with the given key value, an error occurs. HEXA_KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).

Constraints:

- ADDR, KEY, HEXA_KEY and FIRST are mutually exclusive. One of them must be specified.
- After an unsuccessful PTREC, the current record pointer is undefined.
- A file must be opened either in IN (input) or UP (update) processing mode; however, libraries must always be opened in IN mode and in sequential access mode (ACCMODE=S).
- For libraries, only the FIRST and STATUS parameters may be specified.
- For UFAS sequential and relative files, the record pointed to is the logical record at the given address. If no such record is found, an abnormal completion code is sent and there is no current record.
- For UFAS indexed files, the record pointed to is the first record whose key satisfies the conditions of comparison. FIRST and ADDR are not authorized.
- PTREC may not be used with tape files and sequential files with RECFORM=V or VB.
- KEYLOCK and KEYSIZE must be used with KEY.
- COND must be used with KEY or ADDR.
- STATUS must be used in Batch Mode.

Examples:

```
PTREC SFN=MYFILE0009 ADDR=874 COND=GT STATUS=PTRECVAR
          {PTREC will get a successful comparison at the first
           direct record address greater than 874, and point to
           that record in the file identified by SFN MYFILE0009;
           GCL variable PTRECVAR will receive the completion code}

PTREC SFN=MYFILE0020 KEY=4007832215 KEYLOC=250 KEYSIZE=6
      COND=EQ STATUS=PTRECVAR
          {PTREC will point to the first record whose leftmost
           6 bytes are equal to 400783; key begins at byte 250 of
           the record; file is identified by SFN MYFILE0020; GCL
           variable PTRECVAR will receive the completion code}
```

4.10.11 READ_RECORD (RDREC)

Purpose:

Reads a record from a previously declared and opened file, and stores it in a buffer or a GCL variable (the working area WA).

Syntax:

```
{ READ_RECORD }
{             }
{ RDREC       }

          SFN = name16

          [ ADDR = dec10 ]

          [ KEY = char251 ]

          [ KEYLOC = dec5 ]

          [ FIRST = bool ]

          [ WA = name31 ]

          [ LENGTH = name31 ]

          [ STATUS = name31 ]

          [ HEXA_KEY = char252 ]
```

Parameters:

SFN	the symbolic file name of the file.
ADDR	the address of the record to be read for UFAS sequential and relative disk files. ADDR is not allowed for libraries or UFAS indexed disk files. The file must be opened in direct access mode (ACCMODE=D). The address of the first record is 1.
KEY	the key value of the record to be read. If no record exists with the given key value, an error occurs. KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).

KEYLOC	the key offset from the beginning of the record for UFAS indexed disk files. The first byte is byte 0. KEYLOC only has meaning when KEY or HEXA_KEY is present. If KEYLOC is omitted, the reference key is the primary key.
FIRST	specifies that the read record is the first record of the file. FIRST is not authorized with UFAS indexed disk files.
WA	the working area. The working area specified is the name of a local or global GCL variable. The read record is stored in this variable. Records longer than the length of the variable are truncated. If WA is not specified, the read record is stored in a buffer attached to the file. In this case, the SPLIT_RECORD command can be used to move the record to a GCL variable.
LENGTH	the name of a GCL variable, declared with TYPE=DEC and LENGTH=5, that will receive the length of the read record (except when the completion code is abnormal).
STATUS	the name of a GCL variable that will receive the completion code for READ_RECORD. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for READ_RECORD are: Normal: 0: Normal execution of command. 1: Truncation of record. 2: Datalim. 4: Synonym. Abnormal: 256: Wrong symbolic file name. 260: File not opened. 261: No current record. 264: Wrong processing mode. 265: System error: argument error. 266: System error: file structure not available. 270: System error: buffer pointer not available. 273: Wrong parameter (FIRST, KEY, HEXA_KEY or ADDR) for this organization. 280: Wrong access mode. 300: Abnormal return code from system primitive.

HEXA_KEY	key value in hexadecimal of the record to be read. If no record exists with the given key value, an error occurs. HEXA_KEY is used with UFAS indexed disk files only; these files must be opened in direct access mode (ACCMODE=D).

Constraints:

- KEY, HEXA_KEY, ADDR, and FIRST are mutually exclusive.
- KEYLOCK must be used with KEY or HEXA_KEY.
- Files must be opened in IN (input) or UP (update) processing mode; however, libraries and tape files must be opened in IN mode only.
- For all types of files, the record read by READ_RECORD done either in sequential access mode (ACCMODE=S) or in direct access mode (ACCMODE=D) with records read sequentially is determined as follows:
  - if the previous command was POINT_RECORD or OPEN_FILE, the current record is retrieved.
  - if the record searched for has been deleted, the current record is the next record in the file and it is retrieved.
  - if the previous command was READ_RECORD, the current record is the next record in the file and it is retrieved.
  - if there is no current record in the file, READ_RECORD is unsuccessful.
  - if there is no next record in the file, the DATALIM completion code is returned and READ_RECORD is unsuccessful.
- After an unsuccessful READ_RECORD, a sequential READ_RECORD must not be issued until one of the following has occurred:
  - a successful CLOSE_FILE, followed by a successful OPEN_FILE for that file.
  - a successful POINT_RECORD for that file.
  - a successful direct READ_RECORD for that file.
- For UFAS disk files, the record read by READ_RECORD done in direct access mode (ACCMODE=D) with records read directly is determined as follows:
  - the current record is the one whose direct record address is given by the ADDR, KEY or HEXA_KEY parameter; this record is read.
  - if the file contains no such record, a NO CURRENT RECORD completion code is returned and READ_RECORD is unsuccessful.
- STATUS must be used in Batch Mode.

Examples:

```
RDREC SFN=MYFILE0020 KEY=4007832215 KEYLOC=250
      LENGTH=RDRECLNGVAR STATUS=RDRECVAR
          {read record from file identified by SFN MYFILE0020
           and store in buffer attached to file; GCL variable
           RDRECLNGVAR will receive length of record; record key
           is 4007832215 and key offset is 250; completion code
           will be written to RDRECVAR}

RDREC SFN=MYFILE0007 FIRST WA=RDRECWAVAR LENGTH=RDRECLNGVAR
      STATUS=RDRECVAR
          {read record from file identified by SFN MYFILE0007;
           record is first record of file; store record in
           RDRECWAVA; store its length in RDRECLNGVAR; completion
           code is written to RDRECVAR}

RDREC SFN=MYFILE0008 ADDR=874 WA=RDRECWAVAR
      LENGTH=RDRECLNGVAR STATUS=RDRECVAR
          {read record from file identified by SFN MYFILE0008
           and store it in GCL variable RDRECWAVAR, the working
           area; record address is 874; GCL variable RDRECLNGVAR
           will receive length of record; GCL variable RDRECVAR
           will receive the completion code}
```

4.10.12 RELEASE_FILE (RLSF)

Purpose:

Releases a file referenced by its symbolic file name (SFN).

Syntax:

```
{ RELEASE_FILE }
{              }
{ RLSF         }

          { ( name16 [ name16 ] ... ) }
    SFN = {                           }
          { star36                    }

    [ STATUS = name31 ]
```

Parameters:

SFN	a symbolic file name, a list of up to 6 SFNs, or a name using the star convention.
STATUS	the name of a GCL variable that will receive the completion code for RELEASE_FILE. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for RELEASE_FILE are: Normal: 0: Normal execution of command. Abnormal: 256: Wrong symbolic file name. 257: No declared file. 265: System error: argument error. 266: System error: file structure not available. 272: System error: semaphore for file not available. 300: Abnormal return code from system primitive.

Constraints:

- If RELEASE_FILE is unsuccessful, the file remains declared and the completion code can be accessed through the EDIT_FILE_ERROR command.
- If RELEASE_FILE is successful, the file is deassigned and the SFN can be used to declare another file.
- STATUS must be used in Batch Mode.

Examples:

```
RLSF SFN=MYFILE0002 STATUS=RLSFVAR
          {release the file identified by SFN MYFILE0002; GCL
           variable RLSFVAR will receive the completion code}

RLSF SFN=(MYFILE0002,MYFILE0020,MYFILE0021) STATUS=RLSFVAR
          {as above for files identified by SFNs MYFILE0002,
           MYFILE0020, and MYFILE0021}

RLSF * STATUS=RLSFVAR
          {as above, but for all user SFNs}

RLSF SFN=MYFILE* STATUS=RLSFVAR
          {as above, but for all SFNs beginning with MYFILE}
```

4.10.13 RETURN_DECLARED_FILE (RTDCLF)

Purpose:

Retrieves the description of a declared file.

Syntax:

```
{ RETURN_DECLARED_FILE }
{                      }
{ RTDCLF               }

          SFN = name16

          [ FILE = name31 ]

          [ SHARE = name31 ]

          [ ACCESS = name31 ]

          [ STATE = name31 ]

          [ STATUS = name31 ]
```

Parameters:

SFN	the symbolic file name of the file.
FILE	the name of a local or global GCL variable, declared with TYPE=CHAR and LENGTH=78, that will receive the external file description.
SHARE	the name of a local or global GCL variable, declared with TYPE=CHAR and LENGTH=8, that will receive a value identifying the maximum allowable level of concurrent access. Possible values are:
= NORMAL	one writer or several readers.
= ONEWRITE	one writer and several readers.
= FREE	no restriction on access.
= DIR	applies only to libraries. Each member can be concurrently accessed by several readers or one writer.

`= MONITOR`	several readers. The accesses to the files are checked and synchronized when necessary by GAC (General Access Control)
ACCESS	the name of a local or global GCL variable, declared with TYPE=CHAR and LENGTH=8, that will receive a value identifying the access mode. Possible values are:
`= WRITE`	allows input, output, append, and update modes.
`= READ`	allows input mode only.
`= SPREAD`	same as READ, but additionally the current program has exclusive file control, regardless of the SHARE value.
`= SPWRITE`	same as WRITE, but additionally the current program has exclusive file control, regardless of the SHARE value.
`= ALLREAD`	no output allowed; only simultaneous input
`= RECOVERY`	applies only to cataloged files. The program has exclusive access for file recovery purposes.
STATE	the name of a local or GCL variable, declared with TYPE=CHAR and LENGTH=7, that will receive a value identifying the state of the file: OPENED, CLOSED, or UNKNOWN.
STATUS	the name of a GCL variable that will receive the completion code for RETURN_DECLARED_FILE. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for RETURN_DECLARED_FILE are:
	Normal: 0: Normal execution of command.
	Abnormal: 256: Wrong symbolic file name. 265: System error: argument error. 266: System error: file structure not available.

Constraints:

At least one of the parameters, FILE, SHARE, ACCESS, or STATE, must be specified.

STATUS must be used in Batch Mode.

Examples:

```
RTDCLF SFN=MYFILE0002 FILE=RTDCLFFILVAR STATUS=RTDCLFVAR
          {external file description of the file identified
           by SFN MYFILE0002 will be written to GCL variable
           RTDCLFFILVAR; GCL variable RTDCLFVAR will receive
           the completion code}

RTDCLF SFN=MYFILE0020 SHARE=RTDCLFSHVAR STATUS=RTDCLFVAR
          {a value giving the maximum allowable level of
           concurrent file access will be written to GCL
           variable RTDCLFSHVAR; file is identified by SFN
           MYFILE0020; GCL variable RTDCLFVAR will receive
           the completion code}

RTDCLF SFN=MYFILE0009 FILE=RTDCLFFILVAR ACCESS=RTDCLFACCVAR
       STATE=RTDCLFSTVAR STATUS=RTDCLFVAR
          {FILE and STATUS as above, but for MYFILE0009;
           access mode will be written to GCL variable
           RTDCLFACCVAR; file state will be written to GCL
           variable RTDCLFSTVAR; completion code to
           RTDCLFVAR}
```

4.10.14 SPLIT_RECORD (SPREC)

Purpose:

Takes a character string from the input buffer of a file and stores it in a GCL variable (the working area WA). The file must be opened in IN (input) or UP (update) processing mode.

Syntax:

```
{ SPLIT_RECORD }
{              }
{ SPREC        }

          SFN = name16

          WA = name31

          INDEX = dec5

          LENGTH = dec3

          [ STATUS = name31 ]
```

Parameters:

SFN	the symbolic file name of the file.
WA	the working area. The working area specified is the name of a local or global GCL variable. The character string taken from the input buffer is stored in this variable.
INDEX	the rank (position) in the input buffer of the first character of the character string to be moved to the GCL variable WA. The value supplied by INDEX must be in the range 1 to 32767, which is the maximum record length.
LENGTH	the length of the character string to be taken from the input buffer. The value must be in the range 1 to 255, which is the maximum length of a GCL variable.

STATUS	the name of a GCL variable that will receive the completion code for SPLIT_RECORD. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for SPLIT_RECORD are: Normal: 0: Normal execution of command. 1: Truncation of record. Abnormal: 256: Wrong symbolic file name. 260: File not opened. 264: Wrong processing mode. 265: System error: argument error. 266: System error: file structure not available. 270: System error: buffer pointer not available.

Constraints:

If INDEX + LENGTH is greater than the length of the input buffer, the receiving area will be padded with blanks.

STATUS must be used in Batch Mode.

Examples:

```
SPREC SFN=MYFILE0002 WA=SPRECWAVAR INDEX=50 LENGTH=40
      STATUS=SPRECVAR
          {character string is taken from the input buffer of
           the file identified by SFN MYFILE0002 and stored in
           GCL variable SPRECWAVAR; string is 40 characters
           long; first character of string is at rank 50 in
           buffer; GCL variable SPRECVAR will receive the
           completion code}

SPREC SFN=MYFILE0020 WA=SPRECWAVAR INDEX=900 LENGTH=200
      STATUS=SPRECVAR
          {as above, with SFN MYFILE0020; character string
           200 characters long; first character of string
           at rank 900 in the buffer; completion code to GCL
           variable SPRECVAR}
```

4.10.15 WRITE_RECORD (WRREC)

Purpose:

Retrieves a record from a buffer or GCL variable, and writes it to a file.

Syntax:

```
{ WRITE_RECORD }
{              }
{ WRREC        }

          SFN = name16

          [ ADDR = dec10 ]

          [ WA = name31 ]

          [ LENGTH = dec5 ]

          [ STATUS = name31 ]
```

Parameters:

SFN	the symbolic file name of the file.
ADDR	the address of the record to be written for UFAS relative files. The record is written to this address. The file must be opened in direct access mode (ACCMODE=D).
WA	the working area. The working area specified is the name of a local or global GCL variable. If WA is specified, the record is taken from the named variable and written to the file. If WA is not specified, the record is taken from the buffer attached to the file; in this case, the contents of the buffer must have been previously initialized, using the BUILD_RECORD command.

LENGTH	the length of the record to be written to the file. When LENGTH is not specified, the length of the record defaults to the length of WA, the working area. When both LENGTH and WA are not specified, the length defaults to the length of the record in the buffer attached to the file.
STATUS	the name of a GCL variable that will receive the completion code for WRITE_RECORD. The variable must be declared with TYPE=DEC and LENGTH=3. Completion codes for WRITE_RECORD are:

Normal:
0: Normal execution of command.
4: Synonym.

Abnormal:
256: Wrong symbolic file name.
260: File not opened.
262: Data length error.
264: Wrong processing mode.
265: System error: argument error.
266: System error: file structure not available.
269: Duplicate record.
270: System error: buffer pointer not available.
273: Wrong parameter (FIRST, KEY, or ADDR)
for this organization.
274: Record key is lower than that of previous record.
278: Overflow.
280: Wrong access mode.
281: ADDR or FIRST unauthorized.
300: Abnormal return code from system primitive.

Constraints:

- The current record pointer is not modified by execution of WRREC.
- UFAS Sequential Disk Files

  must be opened in OU (output) or AP (append) processing mode with ACCMODE=S.
  ADDR is not authorized.
- UFAS Relative Disk Files

  if ADDR is used, file must be opened in OU or UP processing mode with ACCMODE=D. If ADDR is not used, file must be opened in OU mode with ACCMODE=S.

  when the file is opened in OU mode, records are written to the file as follows:
  – if ACCMODE=S, the first record written will have address 1; subsequent records will have addresses 2, 3, 4, etc.
  – if ACCMODE=D, the ADDR parameter must provide a direct record address. The record is written to this address.

  when the file is opened in UP mode and ACCMODE=D, ADDR must provide a direct record address. The record is written to this address, which must not already contain an active record.
- UFAS Indexed Disk Files

  for sequential access, the file must be opened in OU processing mode with ACCMODE=S.

  for direct access, the file must be opened in OU or UP mode with ACCMODE=D.

  if OU mode is specified, records must be written in ascending key value order.

  if UP mode is specified, records may be written in any order.

  ADDR is not authorized.
- Libraries

  WRREC is authorized for SL libraries only.

  the library must be opened in OU or AP processing mode with ACCMODE=S.

  SSF format is not handled.

  ADDR is not authorized.
- Tape Files

  the file must be opened in OU or AP processing mode with ACCMODE=S.

  ADDR is not authorized.
- STATUS must be used in Batch Mode.

Examples:

```
WRREC SFN=MYFILE0002 ADDR=874 WA=WRRECWAVAR LENGTH=250
      STATUS=WRRECVAR
          {record is retrieved from GCL variable WRRECWAVAR;
           written to file identified by SFN MYFILE0002;
           record length is 250 characters; record address
           is 874; GCL variable WRRECVAR will receive the
           completion code}

WRREC SFN=MYFILE0009 ADDR=11449 STATUS=WRRECVAR
          {record is retrieved from a buffer attached to the
           file and written to the file identified by SFN
           MYFILE0009; buffer has already been initialized by
           BUILD_RECORD; record length defaults to length of
           record in buffer; completion code to WRRECVAR}

WRREC SFN=MYFILE0020 WA=WRRECWAVAR LENGTH=1800
      STATUS=WRRECVAR
          {record is retrieved from GCL variable WRRECWAVAR
           and written to the file identified by SFN
           MYFILE0020; record length is 1800 characters;
           completion code to WRRECVAR; if a UFAS indexed
           file is assumed, ADDR cannot be used}
```

5. GCL Batch Job

5.1 Overview

5.1.1 GCL Job Statements

A GCL batch job description consists of statements and data records.

The statements that may be used are:

- Input Reader statements
- and GCL statements being
  - System Level commands
  - Directives
  - and GCL basic commands.

Input Reader statements begin by a '$' character and end by a semi colon character. The '$' character must be the first character (other than blank) of the line. These statements may be split into more than one source line. More than one such statement cannot be entered on the same line.

System Level commands, directives and GCL basic commands may be split into more than one source line. Each statement must be terminated by a semicolon. More than one command can be entered on the same line; in this case, each command is separated from the preceding one by a semicolon as follows:

```
command-1; beginning-of-command-2

remainder-of_command-2;command-3;
```

Data records contain either data or commands specific to a processor. These records are enclosed between $INPUT and $ENDINPUT statements.

EXAMPLE OF A GCL BATCH JOB:

```
$JOB LKSM13,CLASS=L,JOBLANG=GCL;
$OPTIONS PRTFILE=GCL.PRTLIB..H_SM13_JOBOUT;
MO #CAT (X,#RON) CLASS=A;
COMM '*****************************************************';
COMM '*                                                   *';
COMM '*              THIS JOB LINKS H_GCL                 *';
COMM '*                                                   *';
COMM '*****************************************************';
KWD NAME=CULIB TYPE=LIB DEFAULT=GCL.CULIB;
KWD NAME=SMLIB TYPE=LIB DEFAULT=GCL.SMLIB;
KWD NAME=LISTING TYPE=LIB DEFAULT=GCL.PRTLIB;
KWD NAME=SYSTEM TYPE=LIB DEFAULT=GCL.SYSTEM;
KWD NAME=MOVE TYPE=BOOL DEFAULT=1;
COMM '*****************************************************';
COMM '*                                                   *';
COMM '*                   DELETE MAPS                     *';
COMM '*                                                   *';
COMM '*****************************************************';
ON_ERROR CONTINUE;
MNLIB SL LIB=%LISTING,
         PRTFILE=DUMMY,
         COMFILE=*L1;
$INPUT L1;
 .
 . Commands of MNLIB
 .
$ENDINPUT;
COMM '*****************************************************';
COMM '*                                                   *';
COMM '*                    LINK H_GCL                     *';
COMM '*                                                   *';
COMM '*****************************************************';
LINK_PG H_GCL,
        LIB=%SMLIB,
        PRTFILE=#CAT (%LISTING,'..H_SM13_MAP'),
        COMFILE=*L2;
$INPUT L2;
 .
```

```
  . Commands of LINK_PG
  .
$ENDINPUT;
IF #GE (#SEV,3);
    GOTO PRINT;
ENDIF;
SET_VALUES SML=%SMLIB;
IF %MOVE;
   MNSYS COMFILE=*L3,OS=%SYSTEM,
   PRTFILE=#CAT (%LISTING,'..H_SM13_MOVE_LKU');
ENDIF;
$INPUT L3,JVALUES;
SM;
IL1 &SML;
MOVE H_SM13 IL1 UNIT=H_GCL;
$ENDINPUT;
LABEL PRINT;
MNLIB SL COMMAND='PRINT
(h_sm13_jobout,h_sm13_map),ASIS;',lib=%listing;
$ENDJOB;
```

❑

5.1.2 Job Submission

The end user can submit a GCL job to GCOS 7 from a file by using:

- the EJR directive
- the system-level command RUN_JOB
- and the programmatic interfaces.

An executing GCL batch job can submit another GCL job through EJR or RUN_JOB.

The job submitted must be specified as containing GCL statements at the following levels:

• $JOB Statement	using the parameter JOBLANG=GCL
• EJR or RUN_JOB	using the parameter JOBLANG=GCL
• Program Interface	using the field JOBLANG
• User level	the variable JOBLANG in the User Profile specifying the default value of the parameter JOBLANG of the EJR directive and RUN_JOB system-level command.

5.1.3 Job Translation and Execution

After introduction by the Input Reader, the job is scheduled and execution starts.

The execution phase begins with the step H_BATCH which:

- displays startups
- compiles and displays GCL statements, the binary code resulting from the compilation remaining in memory
- executes startups
- and executes the GCL statements, being the binary code previously produced.

The GCL statements, if requested, and error messages are displayed in the Job Sysout.

5.2 Input Reader Statements

The Input Reader statements are

- $JOB
- $ENDJOB
- $INPUT
- $ENDINPUT
- $SWINPUT
- $SENDCONS
- and $OPTIONS.

The Input Reader statements are not parameterizable (neither values, nor GCL expressions).

Only the commands and parameters usable in GCL batch are described.

5.2.1 $JOB

Purpose:

$JOB, if used, must be the first in a job description to mark the beginning of a job enclosure to identify the job. $JOB and $ENDJOB are not needed if only one job is submitted. The statement is recognized by the Stream Reader which stores the job description statements in the backing store.

If a valid site catalog exists, values specified in $JOB for USER, PROJECT and BILLING must correspond exactly with the entries in the site catalog.

Syntax:

```
$JOB       job-name

   [ USER=user-name ]
   [ PROJECT=project-name ]
   [ BILLING=billing-name ]
   [ NSTARTUP ]

   [       { SOURCE }]
   [ LIST={ NO     }]
   [       { ALL    }]

   [      { NORMAL }]
   [ JOR={ ABORT  }]
   [      { NO     }]

   [ CLASS=identifier2 ]
   [ HOLD=digits2 ]
   [ HOLDOUT ]
   [ PRIORITY=digit1 ]
   [ REPEAT ]
   [ HOST=name4 ]
   [ JOBLANG={ JCL | GCL }]
```

Parameters:

job-name	***Mandatory:*** Up to 8 alphanumeric, hyphen and underscore characters, the first being alphanumeric. Job-name must appear in the same record as the $JOB statement. The job is known to the system by its RON. If there is no $JOB, the job-name is the member-name or external-file-name. If the name exceeds 8 characters only the first 8 characters are used.
USER	Up to twelve alphanumeric, hyphen and underscore characters, the first being alphanumeric. If Access Rights have been implemented, USER values may be restricted depending on the submitter: – for the main or station operator, any USER value is valid – for a batch or IOF user, USER must be the submitter.
PROJECT	User's project up to 12 alphanumeric, hyphen and underscore characters, the first being alphanumeric. *PROJECT is* ***mandatory*** *in $JOB if there no default is specified in the SITE.CATALOG.*
BILLING	Used for accounting and control of the current PROJECT, up to 12 alphanumeric, hyphen and underscore characters, the first being alphanumeric. If BILLING is omitted and no default in the SITE.CATALOG is associated with the current project, PROJECT appears for BILLING in the JOR banner. *BILLING is* ***mandatory*** *in $JOB if there no default is specified in the SITE.CATALOG.*

NSTARTUP	Startup sequences are logically inserted after $JOB if they exist in the startup source library SITE.STARTUP. They may contain all system level commands and directives executable in batch mode. In Batch, both a mandatory and an optional start-up sequence may be attached to a project in the catalog. For GCL batch jobs, available startups are: a. SITE_GCL_B b. *project*_GCL_B c. *project_user*_GCL_B Rules for Implementing Startup Sequences are: *1*. *a*, *b* and *c* can be mandatory. *2*. Only *b* and *c* are optional. *3*. Mandatory startup executes before the optional one. *4*. Optional startup can be inhibited by specifying: - NSTARTUP in $JOB - STARTUP=0 in EJR or RUN_JOB - or STARTUP of the input programmatic structure. *5*. If PROJECT=SYSADMIN and NSTARTUP are specified, ***all*** startups are inhibited.
LIST	Lists the type of information on the GCL in the JOR. *If Access Rights have been implemented and the user does not have the right to READ the file(s) to be printed, LIST is ignored.*
`=ALL`	- the source GCL – expansions of GCL sequences entered on $SWINPUT – startup sequences and error messages.
`=No`	stream reader statements and comments and error messages.
`=SOURCE`	*Default:* – the source GCL and inserted Stream Reader statements – records inserted using $SWINPUT with CONSOLE – and error messages.

JOR	Determines when the JOR is to be produced:
`=ABORT`	Prints the JOR only if the job aborts.
`=NO`	The JOR is not printed.
`=NORMAL`	*Default:* The JOR is printed.

CLASS

Job class defining:

- the default scheduling
- the execution priority
- and the maximum multiprogramming level.

The job is attached to one of the following job classes:

- 16 job classes defined by single letter from A through P
- 416job classes of two letters from AA through PZ.

The operator can suspend and reactivate a given class for flexible and efficient use of machine resources. Job class selects jobs and manages the serial execution of jobs.

Up to 26 classes attached to a project. The PROJECT in the SITE.CATALOG may restrict the class of a job by its:

- batch default class
- and IOF default class.

A job submitted with a job class that either does not exist or is not accessible to the project is launched in class P.

HOLD

Prevents the job from being scheduled for execution until the submitter or operator:

- issues a RELEASE_JOB (RJ) command
- or suppresses HOLD in the RJ command.

A value specified with HOLD becomes the *hold* count for the job and decrements by 1 each time RJ is issued. The job is then available for scheduling once the *hold* count reaches zero. HOLD may be forced to 0 if `RJ STRONG` is issued.

HOLDOUT

Retains output files produced by the job until released by a RELEASE_OUTPUT (RO) command.

PRIORITY	Scheduling priority between 0 (highest) and 7 (lowest). Jobs are scheduled according to this priority. The scheduling priority of a job may be restricted by its PROJECT in the SITE.CATALOG. If the specified priority exceeds the maximum value allowed for its PROJECT, the maximum is used instead and a warning is issued. Within a specified priority, jobs are scheduled on a first-in-first-out (FIFO) basis. *A job will not be scheduled if the maximum number of jobs in the same class is already scheduled.* *Default:* depends on CLASS
REPEAT	Specifies that the job may be repeated from its beginning after a warm restart. If REPEAT is omitted, the entire job cannot be repeated.

HOST	Up to 4 alphanumeric characters identifying a DPS 7/7000 site on which the job is to run, if different from the site where the GCL is entered or stored.
	
	Example:
	
	`$JOBDMJOB,USER=DJM,PROJECT=MECTP,`
	`HOST=BP3C;`
	
	`DMJOB` will be transferred to and executed on `BP3C`.

JOBLANG	Identifies the Command Language used for the JOB description:
`=GCL`	GCOS Command Language. If not loaded, the default JCL is used.
`=JCL`	*Default:* JOB Control Language If GCL, it is recommended to specify the JOBLANG in $JOB because the JOBLANG parameter of the EJR command is ignored in case of INFILE=remote-file or if HOST is specified in the EJR command.
EXPVAL	*Meaningful only with `JOBLANG=JCL`:* Specifies that the values in the JCL are to be expanded in the JOR.

5.2.2 $ENDJOB

Purpose:

$ENDJOB must appear as the last statement of a job which starts with $JOB to terminate the job enclosure. The Stream Reader recognizes the statement to closes the file containing the source GCL statements (see $JOB).

Jobs submitted with the directive EJR do not necessarily require $JOB. If $JOB is omitted, $ENDJOB must also be omitted.

If $ENDJOB is required but omitted, the job aborts.

Syntax:

```
$ENDJOB;
```

Parameters:

None.

5.2.3 $INPUT

Purpose:

Opens an input enclosure and names the SYSIN subfile into which the records of the enclosure are stored.

Syntax:

```
$INPUT      input-enclosure-name

   [      { DATA    }]
   [      { DATASSF }]
   [      { COBOL   }]
   [ TYPE={ COBOLX  }]
   [      { FORTRAN }]
   [      { GCL     }]
   [      { GPL     }]
   [      { JCL     }]

   [ FORMAT=( digit3,digit3,digit3,digit3 )]

   [     { ENDINPUT  }]
   [ END={ DOLLAR    }]
   [     { MATCH     }]
   [     { 'string8' }]

   [ ENDCHAR='char1' ]
   [ CONTCHAR='char1' ]
   [ PRINT ]
   [ CKSTAT ]
   [ JVALUES ]
```

Parameters:

input-enclosure-name	***Mandatory**:* Name of SYSIN subfile *unique within a job* containing the records of the input enclosure. Up to 16 alphanumeric, hyphen and underscore characters.
TYPE	Type defining format of input enclosure records. FORTRAN, JCL, COBOLX, GPL and GCL (and their implicit formats) are treated in the *Library Maintenance Reference Manual.*
`=DATA`	*Default:* if TYPE and FORMAT are omitted. Enter data without modifying the input format and without adding a header to each input record. Used for COBOL or FORTRAN source statements.
`=COBOL`	Add an SSF header to each input record for automatic processing by the system. Used for COBOL source program.
`=DATASSF`	*Default:* if FORMAT specified but not TYPE. Add an SSF header to each input record for automatic processing by the system. Each data record is numbered in the SSF header from 10 and incremented by 10's. An SSF control record is placed at the beginning of the SYSIN subfile. This value applies when: – data input to a COBOL object program is ACCEPTED from SYSIN – and FORTRAN source statements are entered.
FORMAT	Format of input records: – if FORMAT is specified, TYPE must be other than DATA for output records – if FORMAT is omitted, the default is the format corresponding to TYPE. See the *Library Maintenance Reference Manual.*

	The 4 values `digit3` in FORMAT denote the respective locations in the input record of: ***1*** the first digit of the line number, where the first location is 1 and leading blanks are considered zeros ***2*** the last digit of the line number For ***1*** and ***2***, if 0 is specified, no line numbers are given. ***3*** the first text character ***4*** the last text character.
END	Indicates to the input reader how the input enclosure is terminated:
`=DOLLAR`	Next input record with dollar sign ($) in column 1. The record is analyzed as a record outside an input enclosure.
`=MATCH`	First $ENDINPUT statement declaring *input-enclosure-name*. A complete input enclosure is contained between this $INPUT and its matching $ENDINPUT.
`=`*`string8`*	First record starting with this *`string`* which must be EBCDIC. This terminator record is not treated as one of the input enclosure data records.
`=ENDINPUT`	*Default:* Next $ENDINPUT.
ENDCHAR	*Applicable only to input enclosure records of type DATA or DATASSF.* Used where each record contains a specific character as its last non-blank character to define this character. May be enclosed in single quotes for protection. *A space as a last character must be protected.* The *endchar* character and all trailing blanks are removed from the input enclosure records.

If ENDCHAR is specified, the last record must contain the specified character:

- records without the last character are concatenated to the next record up to a maximum of 256 characters
- until a record with the specified character in the last non-blank position is encountered signifying the end of record.

Examples of ENDCHAR:

In the examples, the letter ~~b~~ represents a blank character.

Example 1:

ENDCHAR=Q

record input (80 characters):
1234Q5678Q~~bb~~9bQ~~bbb~~............b

record stored (14 characters):
1234Q5678Q~~bb~~9~~b~~

Example 2:

ENDCHAR='~~b~~'

record input (80 characters):
~~b~~VENI~~b~~VIDI~~b~~VICI~~bbb~~............b

record stored (15 characters):
~~b~~VENI~~b~~VIDI~~b~~VICI

Example 3:

ENDCHAR=/

records input (80 characters):
AAA~~bbb~~.................~~b~~
BBB/~~bbb~~................~~b~~
CCC~~bbb~~.................~~b~~
DDD/~~bbb~~................~~b~~
EEE/~~bbb~~................~~b~~

records stored:
AAA~~bbb~~..................~~b~~BBB (83 characters)
CCC~~bbb~~..................~~b~~DDD (83 characters)
EEE . (3 characters)

CONTCHAR

Applicable only to records of type DATA or DATASSF type: The continuation-character optionally protected by single quotes. A record formed by CONTCHAR must not exceed 255 characters.

The specified character in the last non-blank character position in each input record, is deleted together with all trailing blanks, and the record is concatenated to the one following.

Examples of CONTCHAR:

In the example, the letter ~~b~~ represents a blank character.

`CONTCHAR=-`

records input:
ABCDE-FFF-~~bbb~~AXZ-~~bbb~~........~~b~~
12345-678-~~bbb~~999~~bbb~~...........~~b~~
-XYZ~~bbb~~...............~~b~~

records stored:
ABCDE-FFF-~~bbb~~AXZ12345-678-
~~bbb~~999~~bbb~~.......~~b~~
-XYZ~~bbb~~....................~~b~~

Examples of both ENDCHAR and CONTCHAR:

ENDCHAR and CONTCHAR may be combined in the same $INPUT statement, provided a different character is specified for each one.

In the examples, the ~~b~~ represents a blank character.

Example 1:

`CONTCHAR=-, ENDCHAR=9`

records input:
ABCD9~~b~~-~~b~~XZ-~~bbb~~........~~b~~
XZZZ9~~b~~-~~b~~AB-9~~bbb~~..........~~b~~

record stored:
ABCD9~~b~~-~~b~~XZXZZZ9~~b~~-~~b~~AB-

Example 2*:*

Consider the input enclosure of 80 characters per record:

```
$INPUT XXXX, CONTCHAR=+, ENDCHAR=/;
AAA~~bbb~~..............~~b~~
BBB/~~bbb~~.............~~b~~
CCC~~bbb~~..............~~b~~
DDD+~~bbb~~.............~~b~~
EEE/~~bbb~~.............~~b~~
FFF~~bbb~~..............~~b~~
GGG+~~bbb~~.............~~b~~
HHH/~~bbb~~.............~~b~~
$ENDINPUT;
```

The following records are stored:

```
AAA~~bbb~~................~~b~~BBB      (83 characters)
CCC~~bbb~~................~~b~~DDDEEE (86 characters)
FFF~~bbb~~................~~b~~GGGHHH (86 characters)
```

If the result of using ENDCHAR and/or CONTCHAR is a record of length zero, the record is ignored.

PRINT

Prints the contents of the input enclosure in the translation part of the JOR when the job is introduced. Up to 200 input records per job.

CKSTAT

Requests the Stream Reader:

- to check each input enclosure record for a $SWINPUT
- and if found, to interpret and execute it.

JVALUES

Applies only to input records of type DATA or DATASSF: Specifies that:

- the input enclosure contains parameter references
- and at execution time, the system uses a version updated by the substituting parameter values applicable at job level.

Example of JVALUES:

```
$JOB MYJOB, USER=SMITH,
PROJECT=Q141;

SET_VALUES IOF, MYPROC_IOF;
MNCMD MYBINLIB COMFILE=*MYFILE;

$INPUT MYFILE JVALUES;
```

list of commands of MAINTAIN_COMMAND processor

```
   ...
   DOMAIN &1;
   COMPILE &2;
   ...

$ENDINPUT;

$ENDJOB;
```

In the example, the input enclosure `MYFILE` contains a set of commands for the `MAINTAIN_COMMAND` processor and ends with a `$ENDINPUT` statement.

Constraints:

Function of Statement

$INPUT with its corresponding $ENDINPUT are boundary statements of an input enclosure containing a job description which can be:

- either data input to a user program or system utility
- or a source program of processor-specific commands to be submitted for compilation, see TYPE.

The Stream Reader recognizes $INPUT to open an input enclosure and to name the SYSIN subfile into which the records of the enclosure are to be stored.

Data read in is stored in a temporary subfile of the SYSIN system file. Each SYSIN subfile is identified by its *input-enclosure-name* specified in $INPUT.

An input enclosure may be placed anywhere other than in a step enclosure and may be made available to any number of steps in a job.

Comments and Restrictions

An *Input-enclosure-name* can be referenced in two ways in GCL batch mode:

- with the EXECUTE_GCL directive
- using the COMFILE parameter of the system processor's commands

EXAMPLE 1:

```
$JOB IE-5 JOBLANG = GCL;
COMM ' Test an Input Enclosure used by   ';
COMM ' EXECUTE_GCL with a parameter defined by the ';
COMM ' parameter VALUES of EXECUTE_GCL     ';
$OPTIONS PRTFILE = LINT.BATCH.TESTRES..T_IE_5;
EXECUTE_GCL INFILE = *IE_5 VALUES = (SFN = T_IE_5) LIST;
$INPUT IE_5;
KWD SFN NAME;
MNLIB SL
  LIB = LINT.BATCH.TESTSL
  PRTFILE = #CAT ('LINT.BATCH.TESTRES..', %SFN)
  COMMAND = #CAT ('PRINT', %SFN) ;
$ENDINPUT ;
$ENDJOB ;
```

❑

EXAMPLE 2:

```
$JOB MYJOB, USER = PILLET, PROJECT = FUEL ;
SET_VALUES MEMBER1, MEMBER2 ;
MNLIB SL MYSLLIB COMFILE = *ENCLOSE ;
$INPUT ENCLOSE JVALUES ;
.....
  List of commands of MAINTAIN_LIBRARY processor ;
.....
  COMPARE &1 &2 ;
.....
$ENDINPUT ;
$ENDJOB ;
```

❑

Input enclosure Parameter Substitution

Input enclosure parameter substitution allows generating several input enclosures from a single definition comprising data records between $INPUT and $ENDINPUT.

An input enclosure can contain parameter references in the form "&string" defined in the current job description within one or more SET_VALUES or MODIFY_VALUES. Substitution occurs ***if and only*** if JVALUES is specified in the associated $INPUT.

Each time the input enclosure is referenced during GCL execution, a new version of the input enclosure is created where each parameter referenced is replaced by the value associated with it.

Restrictions

Parameters are substituted only within input enclosures of type DATA or DATASSF.

Parameters cannot be substituted within input enclosures referenced by the EXECUTE_GCL directive.

Protection from Parameter Substitution

A contiguous sequence of records in an input enclosure can be *protected* from substitution as follows:

- a record containing the five characters `//BOD` (starting in column 1) must precede the first record to be protected
- followed by a record containing the five characters `//EOD` in the first 5 columns must follow the last record to be protected.

Each time a version of the input enclosure is created, these two delimiting records are removed from the sequence and no parameter substitution occurs.

This function is useful if an input enclosure contains MAINTAIN_LIBRARY editing statements, where & characters are not to be interpreted as parameter references. For details of the MAINTAIN_LIBRARY editing facilities, see the *Text Editor User's Guide*.

Order of Application of Parameters

The parameters of $INPUT are applied in the order in the following order:

1) CKSTAT

If CKSTAT is specified, the $SWINPUT function is applied first, except where following have been specified:
1) END=DOLLAR
2) END='$SWINPUT' (END='$SWI')

in which case a $SWINPUT encountered in the input enclosure is interpreted as ***the end of the input enclosure*** and not as a $SWINPUT statement.

In ***2***, $SWINPUT ($SWI) can be applied first if it and its parameters appear in the same record.

Example:

```
$INPUT XXXX END='$SWINPUT';
$SWINPUT FILE-A;
```
is treated as a $SWINPUT statement and not as the end of the input enclosure.

However,
```
$INPUT XXXX END='$SWINPUT';
$SWINPUT
FILE-A;
```
is treated as the end of the input enclosure and not as a $SWINPUT statement.

2) CONTCHAR *and/or* ENDCHAR

3) FORMAT

And ***only lastly*** by:

4) parameters of the *input-enclosure*.

5.2.4 $ENDINPUT

Purpose:

$ENDINPUT marks the end of an input enclosure introduced by $INPUT. When the statement is encountered by the stream reader, the SYSIN subfile containing the input enclosure records is closed.

Syntax:

```
$ENDINPUT [ input-enclosure-name ];
```

Parameter:

input-enclosure-name	*Meaningful only if `END=MATCH` was declared in the `$INPUT` statement corresponding to this `$ENDINPUT` statement:* If both input-enclosure-names match, the $ENDINPUT is considered to be the closing statement of the input enclosure. $ENDINPUT is treated as data contained in the current input enclosure if its input-enclosure-name does not match that specified in a previous $INPUT statement.

5.2.5 $SWINPUT

Purpose:

$SWINPUT names the file to which the input stream is to switch from the current stream or file. The effect is equivalent to:

- removing the $SWINPUT statement from the stream of the Stream Reader
- and replacing it with the contents of the file which it references.

Instead of referencing a file, a $SWINPUT may refer to CONSOLE input in which case the console becomes the submitter.

$SWINPUT can appear anywhere within a job-enclosure or an input-enclosure. If a $SWINPUT appears within an input-enclosure, CKSTAT must be specified in $INPUT. When the file to which the input has been switched reaches its *end-of-file*, the Stream Reader reverts to the stream containing the $SWINPUT.

$SWINPUT may also be embedded in a GCL statement.

The restrictions on using this statement are:

- *$SWINPUT must not be embedded in a Stream Reader statement, namely:*
  - *$JOB and its associated $ENDJOB*
  - *$INPUT and its associated $ENDINPUT*
  - *and another $SWINPUT.*
- *The file referred to by $SWINPUT must not contain a $ENDJOB statement.*
- *Although any number of $SWINPUT statements may appear in a stream, the level of nesting cannot exceed 3.*

Syntax:

```
$SWINPUT

   {[INFILE=] sequential-input-file-description                 }
   {                                                            }
   { member-name [INLIB=] input-library-description             }
   {                                                            }
   {         {'string105'[ANSWERS=('string105'['string105']...)]}}
   { CONSOLE={                                                  }}
   {         {'string105'['string105'] END='string8'            }}
```

Description of Parameters:

The only mandatory parameter is that which declares the file to which the input stream is to be switched:

sequential-input-file-description
: Optionally prefixed by `INFILE=` to reference a sequential file in the GCL format, whose contents replace the `$SWINPUT` statement in the input stream.

input-library-description
: Optionally prefixed by `INLIB=` to reference an input-library in the GCL format from which a member will be selected. The contents of *`member-name`* replaces the `$SWINPUT` statement in the input stream.

CONSOLE
: Switch input to the console in the one of the following ways:

`ANSWERS=`
: *`'string 105'`* preceding `ANSWERS` specifies the prompt sent to the console to solicit the desired input. `ANSWERS=`*`'string 105'`* defines the possible valid inputs. After three unsuccessful attempts to enter a valid reply, the job aborts. The first valid reply constitutes the only switched input and is inserted in the input stream in place of `$SWINPUT`. If ANSWERS and the valid replies are not specified, any reply up to one record length is accepted.

`END=`	Several lines of input are expected from the console: – the first *`'string 105'`* is the initial prompt sent to the console to solicit the first line of input – the second *`'string 105'`* is the next prompt sent to the console to solicit the next and subsequent lines of input. If the second string is missing, it defaults to the first string. Both are positional parameters. `END=`*`'string 105'`* indicates the end of the input. Replies are not restricted to valid responses as for `ANSWERS` and form the switched input data replacing the `$SWINPUT` in the original stream.

5.2.6 $SENDCONS

Purpose:

`$SENDCONS` displays a message on the operator's console when the Job is introduced.

Syntax:

```
$SENDCONS 'string105';
```

Parameter:

string105	Message up to 105 characters.

5.2.7 $OPTIONS

Purpose:

$OPTIONS is used only with GCL Batch Jobs to specify permanent options of the H_BATCH processor. The option PRTFILE specified at job submission time (in commands EJR or RUN_JOB) overrides the option PRTFILE specified in $OPTIONS.

Syntax:

```
$OPTIONS

     [ PRTFILE=file-78 ]
     [ REPEAT ]


     [          { NONE   }]
     [ JOURNAL={ AFTER  }]
     [          { BEFORE }]
     [          { BOTH   }]

     [       { NO    }]
     [       { ALL   }]
     [ DUMP={ DATA  }]
     [       { PALL  }]
     [       { PDATA }]
```

Parameters:

PRTFILE	Name of the report file of H_BATCH processor. When omitted: – the default is that specified at job submission time – the report is displayed in the SYS.OUT file.
REPEAT	Determines how the step H_BATCH is to proceed on step abort or after a system crash:
=1	the step H_BATCH is repeated.
=0	no repeat.

JOURNAL	Applicable to the files accessed at system level using the GCL commands. See Section 4. Must be specified when at least one of the cataloged files processed through GCL commands has other than `JOURNAL=NO` in the catalog:
`=BEFORE`	if the files have `JOURNAL=BEFORE` or `JOURNAL=NO`
`=AFTER`	if the files have `JOURNAL=AFTER` or `JOURNAL=NO`
`=BOTH`	if the files have `JOURNAL=BOTH`, `=AFTER` and `=BEFORE`
`=NO`	*Default:* if all files have `JOURNAL=NO`.

Refer to the *File Recovery Facilities User's Guide* for recovery of files.

DUMP	Specifies if a dump listing is to be produced on abnormal program termination, and the form of its contents:
`=ALL`	all program segments
`=DATA`	only data segments
`=PALL`	only private segments
`=PDATA`	only private data segments
`=NO`	*Default:* no dump listing.

5.3 System Level Commands

The System-level commands may be executed in batch mode.

For details of syntax, refer to *IOF Terminal User's Reference Manual Part 2*, Sections 3 through 7.

5.4 Directives

The directives that can be used in a GCL job, may be executed in batch mode.

For details of syntax, refer to *IOF Terminal User's Reference Manual Part 3*, Section 2.

5.5 GCL Basic Commands

See Section 2.

5.6 Parameterization

5.6.1 Parameterization of GCL Statements

It is done by the KEYWORDs of the GCL sequence. The value of each keyword parameter can be passed to the GCL sequence in the VALUES parameter of EJR or RUN_JOB, or in the field VALUES_DESCRIPTION if programmatic interface is used.

EXAMPLE:

```
$JOB TEST JOBLANG=GCL;
   KWD X DEC 3;
   KWD Y DEC 3;
     LET # #PLUS (%X, %Y);
$ENDJOB;
```

This job can be submitted by command:

```
EJR INFILE=MY_JOB VALUES=(X=2, Y=5);
```

or

```
EJR INFILE=MY_JOB VALUES=(2,5);
```

It is the only way to parameterize the GCL statements.

❑

5.6.2 Parameterization of Input Enclosures

The Input Enclosures may contain:

- data or processor specific commands:

  In this case the Input Enclosures can be parameterized with values set by SET_VALUES and MODIFY_VALUES statements.

  Values are referenced symbolically preceded by an '&' sign in the SET_VALUES statement by:
  – a positional parameter
  – or by the keyword specified.

  The parameterization by values is allowed only in Input Enclosures.

  The values are modified each time a command SET_VALUES or MODIFY_VALUES is executed.

  The parameterization by values is done at execution time each time an Input Enclosure is referenced by a processor.

  The parameterization of processor-specific commands may also be done with Global Variables. The parameterization with Global Variables is done when the command is executed.

- GCL statements to be executed using the directive EXECUTE_GCL.

  In this case the parameterization of Input Enclosures with SET_VALUES and MODIFY_VALUES is not allowed. Only parameterization by Global Variables may be used.

5.6.3 Example of Parameterization

The parameterization of the Input Enclosure `IE` is done by SET_VALUES and by Global Variables:

```
$JOB TEST_IE JOBLANG=GCL;
$OPTIONS PRTFILE=GCL.PRTLIB..TEST_IE;
KWD MEMBER_NAME NAME 31;
SET_VALUES SF1=%MEMBER_NAME;
GLOBAL SF2 NAME 31;
LET SF2 %MEMBER_NAME;
MNLIB SL COMFILE=*IE
  LIB=GCL.TESTSL
  PRTFILE=GCL.PRTLIB..TEST_IE;
$INPUT IE JVALUES;
  PRINT &SF1;
  PRINT %SF2;
$ENDINPUT;
$ENDJOB;
```

5.7 Chaining of Commands

GCL commands are normally executed in sequence. Several GCL basic commands such as IF, WHILE, CASEOF and GOTO allow modifying sequential chaining. In the case of Severity 3 or 4 error occurring during the execution of a processor or of a directive between steps, the chaining of commands can be controlled through the ON_ERROR command.

The user can, for example, decide if a Severity 3 or 4 error should occur, for the Job:

- either to terminate by specifying:
  `ON_ERROR ACTION=ABORT;`
- or to continue by specifying:
  ON_ERROR ACTION=CONTINUE;

In the latter case, the user can nevertheless control the chaining of the commands to execute by testing the severity of the error by using the IF command, for example:

```
IF #GE(#SEV,3);
GOTO label;
ENDIF;
```

Note that the severity is not reset to 0 when the step H_BATCH is executed. Therefore, if the user's step is aborted with a severity 3, H_BATCH will execute normally but will terminate with an abort severity 3, showing the last return codes.

5.8 Recoveries

Checkpoint and Restart facilities are available for jobs executed in batch mode.

Checkpoint is activated by the REPEAT parameter of:

- the $JOB command for the whole job or at the submission time
- the commands EXEC_PG or STEP for user programs
- the $OPTIONS statement for the system processor H_BATCH.

Checkpoint and Restart are complemented by the Journal files. BEFORE and/or AFTER Journals are used for files processed:

- by user programs
- and at System level by the GCL commands described in the *GCOS File Access Commands* of *Section 4*. In this case the BEFORE and AFTER Journal are activated by the parameter JOURNAL of the $OPTIONS statement.

The rules are the same as for JCL mode. For a complete discussion, please refer to the manual *GCOS 7 File Recovery Facilities User's Guide*.

When the whole job is restarted, all the private data attached to the job such as Global Variables and Values, and parameterized Input Enclosures are deleted. The System Variables are reset to their initial value.

When the job is restarted at the beginning of a step or at a Checkpoint, the private data attached to the job, namely, Global Variables, System Variables and Values, are set to the value saved by Checkpoint at the beginning of the step or at Checkpoint. The parameterized Input Enclosures are still valid.

GCL Batch Jobs may be restarted after a System Restart WARM without RESTORE if:

- the Sharable Module H_SM13 has not been loaded through the LOADSM command
- the system files SYS.SPOOLi have not been re-initialized through the SPOOL command.

5.9 Reports

5.9.1 Job Occurrence Report (JOR)

A GCL batch job report is not unlike any other job report except when H_BATCH executes in the termination phase of a user step. In this case, the user step report contains information on both executions: the user step execution and H_BATCH execution.

In the following example, the job report contains:

- directives
- a user step (MAINTAIN_LIBRARY) and its directives.

EXAMPLE:

```
JOBID=1STEP  USER=MARGULIS PROJECT=LINT BILLING=LINT--V6 RON=X1279
------------------------------------------------------------------
14:22:07 JOB INTRODUCED FROM                          DEC 14, 1993
         1STEP LINT.GB.SL BFU033
------------------------------------------------------------------
13:22:09 START OF TRANSLATION
         $JOB 1STEP JOBLANG=GCL LIST=ALL;
         $ENDJOB;
         RECORD COUNT: 7
13:22:09 END OF TRANSLATION
------------------------------------------------------------------
13:22:09 JOB EXECUTION LISTING                        DEC 14, 1993
         STEP 1
         LOAD MODULE=H_BATCH (18:03 OCT 28, 1993) PREINITIALIZED
         LIBRARY=SYS.HLMLIB
13:22:10 STEP STARTED XPRTY=9 (DEC 14, 1993)
         TASK MAIN PGID=000C PRID=00 COMPLETED
         SYSBKST ON S1F6B1: NB OF IO REQUESTS=0
         SYSPVMF ON S1F6B1: NB OF IO REQUESTS=29
         SYSLIB ON S1F6B1: NB OF IO REQUESTS=114
         SYSBKST* ON S1F6B1: NB OF IO REQUESTS=3
         H_GCL_ST ON S1F6B1: NB OF IO REQUESTS=4
         UFILE ON S1F6B1: NB OF IO REQUESTS=1
         BRD_BIN1 ON BVU0E5: NB OF IO REQUESTS=100
         H_GCL_B ON S1F6B1: NB OF IO REQUESTS=4
         CPU 0.070 PROG MISSING PAGES 102
         LINES 41 LIMIT NOLIM BACKING STORE 0
         LOCKED 143360
         CARDS 09 LIMIT NOLIM BUFFER SIZE 110592 CPSIZE 4096
```

```
13:22:24 STEP COMPLETED (DEC 14, 1993)
         JUMP CONTINUE
         STEP 2
         LOAD MODULE=H_LIBMAINT (18:02 MAR 19,1992) PREINITIALIZED
         LIBRARY=SYS.HLMLIB
13:22:25 STEP STARTED XPRTY=9 (DEC 14, 1993)
         TASK MAIN PGID=000C PRID=00 COMPLETED
13:22:30 USER STEP COMPLETED (DEC 14, 1993)
         TASK H_BATCH PGID=000C PRID=00 COMPLETED
         SYSBKST ON S1F6B1: NB OF IO REQUESTS=0
         SYSPVMF ON S1F6B1: NB OF IO REQUESTS=29
         SYSLIB ON S1F6B1: NB OF IO REQUESTS=16
         SYSBKST* ON S1F6B1: NB OF IO REQUESTS=0
         BRD_BIN1 ON BVU0E5: NB OF IO REQUESTS=7
         BRD_BSYS ON S1F6B1: NB OF IO REQUESTS=16
         LIB ON BVU0E5: NB OF REQUESTS=52
         H_PR ON S1F6B1: NB OF REQUESTS=4
         H_GCL_B ON S1F6B1: NB OF IO REQUESTS=4
         CPU 0.036 PROG MISSING PAGES 27 STACKOV 3
         ELAPSED 0.111 SYS MISSING PAGES 1
         LINES 51 LIMIT NOLIM BACKING STORE 0 LOCKED 155648
         CARDS 0 LIMIT NOLIM BUFFER SIZE 122880 CPSIZE 4096

13:22:32 STEP COMPLETED (DEC 14, 1993)

         START 13:22:09 (DEC 14, 1993) LINES 92
         STOP 13:22:33 (DEC 14, 1003) CARDS 0
         CPU 0.106
         ELAPSE 0.403
13:22:33 RESULT JOB COMPLETED
```

❑

In the Translation phase, only Input Reader statements and the contents of Input Enclosures, if requested, are listed.

The Step reports contains two steps: an H_BATCH step report and an H_LIBMAINT step report. The example is in fact the report of the user step and the report of the execution of the H_BATCH task, both tasks being executed in the *same* step.

The information on the number of IOs, CPU, elapsed time, lines and cards that appear between the messages USER STEP COMPLETED and STEP COMPLETED apply to *both* tasks.

5.9.2 H_BATCH Report

When an error occurs during the execution of a GCL command, two kinds of error messages can be printed in the H_BATCH report depending on the erroneous procedure:

- The procedure containing the error is not locked. In this case, the error message will be prefixed by the procedure name (see example 1).
- The procedure that containing the error is locked. In this case, the error message will be prefixed by the initial calling procedure in the job and its associated line number in the report (see example 2).

In both examples, the error occurs in the procedure P3.

EXAMPLE 1:

DESIGN:

```
$job job_doc joblang = GCL holdout;
mwinlib bin agtr.br.binlib;
let # 'begin job';
P1;
let # 'end job';
$endjob;

proc P1  lock = 0;  +-->proc  P2  lock = 1; +-->proc  P3 lock = 0;
                    |                       |
                    |                       |
call P2;------------+   call P3; -----------+   let # %G;

endproc;                endproc;                endproc;
```

REPORT:

```
*****************************************************************
**** GCOS 7                                                  ****
**** BATCH                                                   ****
****                                                         ****
****                      VERSION: 01.00 DATED: NOV 15, 1993 ****
*****************************************************************
15:35:32 START EXECUTION
   1: mwinlib bin agtr.br.binlib;
   2: let # 'begin job';
   3: P1;
   4: let # 'end job';
   begin job
***P3:  VARIABLE G HAS NOT BEEN DECLARED
*********************************************B*A*T*C*H************
```

❑

EXAMPLE 2:

DESIGN:

```
$job job_doc joblang = GCL holdout;
mwinlib bin agtr.br.binlib;
let # 'begin job';
P1;
let # 'end job';
$endjob;


proc P1  lock = 0;  +-->proc  P2  lock = 1; +-->proc  P3 lock = 1;
                    |                       |
                    |                       |
call P2;------------+   call P3; -----------+   let # %G;


endproc;                endproc;                endproc;
```

REPORT:

```
******************************************************************
**** GCOS 7                                                   ****
**** BATCH                                                    ****
****                                                          ****
****                       VERSION: 01.00 DATED: NOV 15, 1993 ****
******************************************************************

15:35:32 START EXECUTION

   1: mwinlib bin agtr.br.binlib;
   2: let # 'begin job';
   3: P1;
   4: let # 'end job';
   begin job
***P1/3:  VARIABLE G HAS NOT BEEN DECLARED
*********************************************B*A*T*C*H************
```

❑

6. Debugging

6.1 GCL Job Debugging

Like any other kind of program development, writing GCL procedures may require some form of debugging. For that purpose, two system variables are provided:

- #TRACE (boolean)
  when 1, execution of all standard supplied commands will be listed on the terminal.
- #DEBUG (boolean)
  when 1, each line executed within a procedure has its number displayed along with the name of the procedure before the line is executed.

The GCL debugging options TRACE and DEBUG (see MODIFY_PROFILE directive) are available in batch mode. When the option DEBUG is active, each executed line is printed after variables evaluation in the Job-Out subfile of the system file SYS.OUT.

These debugging options can be specified at job submission by parameters TRACE_GCL and DEBUG_GCL of the directive EJR or the system level command RUN_JOB.

Both system variables may be used simultaneously and may be set from outside or from inside the procedure being debugged. They are part of the user's profile and may be set by use of the MODIFY_PROFILE (MDP) directive:

```
MDP TRACE DEBUG
```

In order to ease debugging, each time an error is found during execution of a procedure, the diagnostics report the line number and name of the procedure where the error occurred. If the error is caused by some erroneous argument being supplied to a builtin function, the name of the builtin function is also reported.

The above will be sufficient in most debugging situations. If this proves insufficient, use the DUMP command at the suspected critical points in the procedure.

6.2 DUMP

The command DUMP may be used to display the value of variables specified by their name in the command. The variables that can be displayed are the local variables, the keywords and the global variables. The result of a GCL expression can also be displayed by the command DUMP.

The command DUMP is executed only when the system variable #TRACE of the User Profile is equal to 1. The command DUMP can also be conditionally executed according to the value of a boolean expression that can be specified by the parameter IF of the command.

6.3 Example of Debugging

```
S: MP TRACE DEBUG
S: EXGCL GCLTEST LIB=COMMON.SLLIB LIST
1:Proc TEST;
2:Local AUTO char 80;
3:Local D dec 3;
4:Let AUTO abcdefg;
5:Let AUTO #cat (%AUTO,'*',#date,#cat (123,x));
6:Dump AUTO;
7:If #eq (%AUTO,XYZ);
8:  Let # TRUE;
9:Else;
10: Let # FALSE;
11:Endif;
12:Let D 1;
13:While #lt (%D,3);
14: Let D #plus (%D,1);
15:Endwhile;
16:Endproc;
+++TEST/4 LET AUTO ABCDEFG
+++TEST/5 LET AUTO #CAT(ABCDEFG<%AUTO>,'*',#DATE,#CAT(123,X))
---TEST/6 DUMP VARIABLES=AUTO;
Variable AUTO:
ABCDEFG*93/12/14123X
+++
+++TEST/7 IF #EQ('ABCDEFG*93/12/14123X'<%AUTO>,XYZ)
+++TEST/10 LET # FALSE
FALSE
+++TEST/12 LET D 1
+++TEST/13 WHILE #LT(1<%D>,3)
+++TEST/14 LET D #PLUS(1<%D>,1)
+++TEST/13 WHILE #LT(2<%D>,3)
+++TEST/14 LET D #PLUS(2<%D>,1)
+++TEST/13 WHILE #LT(3<%D>,3)
+++TEST/16 RETURN
```

❑

7. Programmatic Interface

7.1 GCL Interface

The user accesses GCL facilities by activating the GCL translator from within the application. Primitives in both GPL and COBOL provide this interface by:

- reading commands from a terminal
- and activating the GCL procedures in a domain.

A domain is a group of commands or procedures which are either user or system defined. A command name is known in a domain if and only if a GCL procedure with that name has been compiled and stored in that domain.

EXAMPLE:

Command name	**GCL procedure**
COBOL	PROC COBOL

❑

Commands are read from the terminal. In screen mode, the user is asked to select a command from those available. For a serial printer, the user must explicitly enter the command name, which is rejected if the command is not in the current domain.

The command read from a terminal starts the associated procedure by:

- setting system variables such as printer width (#PW) and national language (#LANG)
- activating built-in functions
- then returning the text to the caller via the SYSTEM command.

The SYSTEM command passes a text string to the calling processor which may:

- either stop activity
- or issue another read primitive to resume executing the current procedure, before asking for a new command.

GCL returns the internal domain name to the calling processor. The domain is referenced by this internal domain name in subsequent calls to the primitives. A single processor can thus access several domains simultaneously.

7.1.1 Primitives

The five primitives *in the order of their function* are:

- GCLINIT
- GCLREAD
- GCLTERM
- GCLABORT
- GCLRETRY.

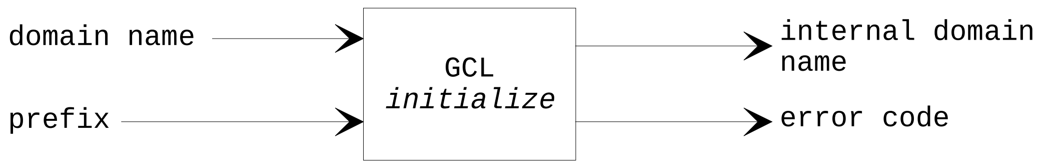
The return given by each primitive is described in terms of:

- its normal execution such as text output in read
- system status such as DATALIM when no further commands are to be processed
- and error codes where the primitive fails to complete.

7.1.1.1 GCLINIT

Purpose:

To initialize the GCL interpreter for a new domain to distinguish similarly named commands belonging to different processors (domains). A domain uniquely identifies the calling processor.



GCLINIT needs as input the name of the domain and a prefix to be associated with the domain. The prefix is a 1-character prompt used to request input from the terminal. In response to this prompt, the terminal user may enter any command of the domain concerned.

The primitive returns a code which denotes a normal or abnormal initialization phase and an internal identification for the domain. This identification must be supplied each time another primitive refers to this domain.

Syntax:

```
GPL:   $H_INITGCL DOMAIN=i_char31 INTDOM=o_ptr PREFIX=i_char1;

COBOL: CALL "CGCLINIT" USING DOMAIN,PREFIX,INTDOM,SUP,SUPP
                             ,ERROR-RC.

COBOL: CALL "CGCLINITE" USING DOMAIN,PREFIX,INTDOM,COMFILE
                               ,PRTFILE,ERROR-RC,ECHO.
```

Parameters:

DOMAIN	Name of the Domain:
	`GPL:   i_char31`
	`COBOL:` `PIC X(31)`

INTDOM	Pointer to Internal domain name:
	`GPL:   o_ptr`
	`COBOL: COMP-2`

PREFIX	Prompt to request input at the terminal:
	`GPL:   i_char1`
	`COBOL: PIC X(1)`

COMFILE	Internal File Name for COMFILE: `COBOL: COMP-2 (or zero's)`
PRTFILE	Internal File Name for PRTFILE: `COBOL: COMP-2 (or zero's)`
SUP	not currently used: `COBOL: PIC X(8)`
SUPP	not currently used: `COBOL: PIC X(8)`
ERROR-RC	Error Code: `COBOL: COMP-2`
ECHO	`COBOL: PIC X(1)`
="Y"	Display menu of command when error message is issued
="N"	No echo: only error messages appear at the terminal.

Return Codes:

Normal	`DONE`
Abnormal	`OBJUNKN:` domain unknown or empty `ARGERR:` argument error Plus other abnormal system return codes.

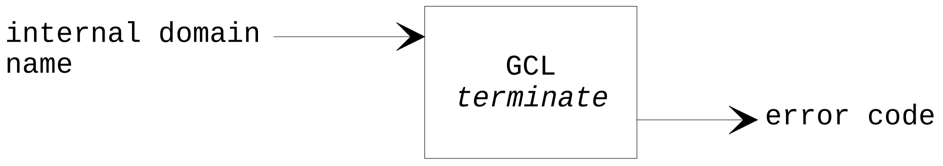
Error Codes (COBOL):

0	normal execution of primitive
259	domain unknown or empty
261	I/O error, details in GR4
262	other error, details in GR4
263	unbundling error
264	argument error
300	GCL error, details in GR4

7.1.1.2 GCLTERM

Purpose:

To terminate the activity of the GCL interpreter on the current domain.



When a calling processor has no further use for the GCL interpreter for a domain, it specifies GCLTERM for that domain. The internal domain name must be supplied. A code is returned to indicate the status of the termination.

Syntax:

GPL: `$H_TERMGCL INTDOM=i_ptr;`

COBOL: `CALL "CGCLTERM" USING INTDOM,ERROR-RC.`

Parameters:

INTDOM	Pointer to Internal domain name:
	`GPL:   i_ptr`
	`COBOL: COMP-2`

ERROR-RC	Error Code: `COBOL: COMP-2`

Return Codes:

Normal	`DONE`
Abnormal	`NOMATCH:` wrong internal domain name `ARGERR:` argument error

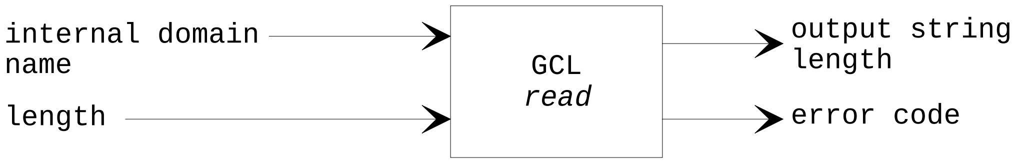
Error Codes (COBOL):

0	normal execution of primitive
256	wrong internal domain name
264	argument error

7.1.1.3 GCLREAD

Purpose:

To request translation of commands read in from the terminal or the COMFILE when specified, to activate the associated procedures.



GCLREAD will continue reading and translating commands until it encounters a SYSTEM command:

- and the value of the PROMPT parameter of the MODIFY_PROFILE command is 1 (default)
- or an error is detected and the value of the PROMPT parameter of the MODIFY_PROFILE command is 0.

If the user requests the end of the current domain activity (end of current action), the IGNORE return code is issued. The calling processor is then supposed to terminate its activity. The text returned to the caller is delivered into the area provided for output.

Syntax:

GPL: `$H_READGCL INTDOM=i_ptr OUTLEN=b_fb31 OUTAREA=o_charn;`

COBOL: `CALL "CGCLREAD" USING INTDOM,OUTLEN,OUTAREA,ERROR-RC.`

Parameters:

INTDOM	Pointer to Internal domain name:
	`GPL:   i_ptr`
	`COBOL: COMP-2`

OUTLEN	Max length of returned data in input; actual length in output:
	`GPL:   b_fb31`
	`COBOL: COMP-2`

OUTAREA	Area where the text produced by SYSTEM is returned:
	`GPL:   o_charn`
	`COBOL: PIC X(n)`

ERROR-RC	Error Code: `COBOL: COMP-2`

Return Codes:

Normal	`DONE`	
	`TRUNC:`	truncation
	`IGNORE:`	end of current action
Abnormal	`NOMATCH:`	wrong internal domain name
	`JCLERR:`	syntax error
	`CDERR:`	GCL procedure aborted
	`ARGERR:`	argument error

Plus other abnormal system return codes.

Error Codes (COBOL):

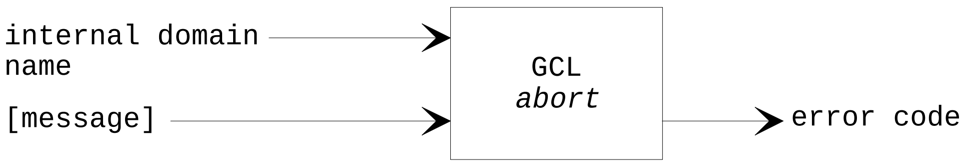
0	normal execution of primitive
1	truncation
2	ignore (end of current action)
256	wrong internal domain name
257	syntax error
260	GCL procedure aborted
261	I/O error, details in GR4
262	other error, details in GR4
264	argument error

7.1.1.4 GCLABORT

Purpose:

To force an external abort of a current procedure execution with an optional error diagnostic.

The next GCLREAD reads a new command line from the input stream and discards any pending SYSTEM and unprocessed commands on the current line.



The message, if present, is output on the terminal. The internal domain name must be provided in input. A status code is returned.

Syntax:

GPL: `$H_ABTGCL INTDOM=i_ptr [ MSG=i_char78 LENGTH=i_fb15 ];`

COBOL:
```
CALL "CGCLABORT" USING INTDOM,ADDRESS OF MSG,FLAG
                       ,ERROR-RC.
```

Parameters:

INTDOM	Pointer to Internal domain name:
	`GPL:   i_ptr`
	`COBOL: COMP-2`

MSG	Message text: `GPL:   MSG=i_char78`
LENGTH	Actual message length: `GPL:   LENGTH=i_fb15`

ADDRESS OF MSG	Length and Text of message:
	`COBOL: 01 MSG.`
	`       02 MSG_LENGTH COMP-1.`
	`       02 MSG_TEXT PIC X(78).`

FLAG	Type of action:
`=0`	abort
`=1`	retry
	`COBOL: COMP-1`
ERROR-RC	Error code:
	`COBOL: COMP-2`

Return Codes:

Normal	`DONE`
Abnormal	`NOMATCH:` wrong internal domain name
	`ARGERR:` argument error
	Plus other abnormal system return codes.

Error Codes (COBOL):

0	normal execution of primitive
256	wrong internal domain name
261	I/O error, details in GR4
264	argument error

NOTE:

The routine CGCLABORT will cause an abort (if FLAG=0) or a retry.

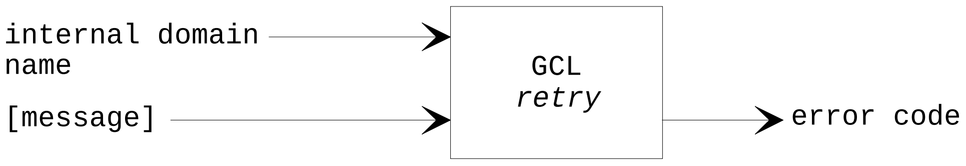
ADDRESS OF MSG is accepted only if the program is compiled with option LEVEL=NSTD in the COBOL command.

7.1.1.5 GCLRETRY

Purpose:

To cancel the execution of a command with an optional error diagnostic and give the user an opportunity to resupply or modify the currently processed command.

Retry is performed only when input is from a terminal and *menu* mode is set whereby keywords for which values are invalid, are prompted again. Otherwise the function is like GCLABORT.



Syntax:

GPL: `$H_RETRYGCL INTDOM=i_ptr [ MSG=i_char78 LENGTH=i_fb15 ];`

COBOL:
```
CALL "CGCLABORT" USING INTDOM,ADDRESS OF MSG,FLAG
                       ,ERROR-RC.
```

Parameters:

INTDOM	Pointer to Internal domain name:
	`GPL:   i_ptr`
	`COBOL: COMP-2`

MSG	Message text: `GPL:   MSG=i_char78`
LENGTH	Actual message length: `GPL:   LENGTH=i_fb15`

ADDRESS OF MSG	Length and Text of message:
	`COBOL: 01 MSG.`
	`       02 MSG_LENGTH COMP-1.`
	`       02 MSG_TEXT PIC X(78).`

FLAG	Type of action:
`=0`	abort
`=1`	retry
	`COBOL: COMP-1`
ERROR-RC	Error code: `COBOL: COMP-2`

Return Codes:

Normal	`DONE`
Abnormal	`NOMATCH:` wrong internal domain name
	`ARGERR:` argument error
	Plus other abnormal system return codes.

Error Codes (COBOL):

`0`	normal execution of primitive
`256`	wrong internal domain name
`261`	I/O error, details in GR4
`264`	argument error

NOTE:

The routine CGCLABORT will cause an abort or a retry (if FLAG=1).

ADDRESS OF MSG is accepted only if the program is compiled with option LEVEL=NSTD in the COBOL command.

7.1.2 Primitives in Schematic Program

The relationship among GCL primitives, terminal activity and GCL procedure execution can be summarized as follows:

```
GCL            GCL             GCL             Text returned
Primitives     Commands        Procedures      to Calling
                                               Processor

 GCLINIT
   |
   |
   |
+-->|
|   |
|GCLREAD------->|
|   |           |
+-->|           |COBOL------>|
   |           |            | PROC COBOL;
   |           |            |
   |           |            | SYSTEM 'C1,...C9';      C1,...C9
   |           |            |
   |           |            | ENDPROC;
   |           |            |
   |           |LINKGO---->|
   |           |            | PROC LINKGO;
   |           |            |
   |           |            | SYSTEM 'L1,...L8';      L1,...L8
   |           |            | SYSTEM 'S1,S2';         S1,S2
   |           |            |
   |           |            | ENDPROC;
   |
   |
 GCLTERM
```

A domain is initialized and named by GCLINIT.

When calling GCLREAD, supply the internal domain name and receive a return code and a text with its length.

GCLREAD accesses, analyzes and translates a command. In the above example, it leads to execution of the procedure `COBOL`.

The first `SYSTEM` encountered suspends execution and GCLREAD returns the text C1,..C9 to the caller.

Then at the next GCLREAD, processing is resumed.

Upon encountering the ENDPROC command:

- the next user-defined command LINKGO is read in
- and the procedure LINKGO is activated.

7.2 Interface Between Program and Procedure

7.2.1 Domain

The domain name specified in the GCLINIT primitive must be the name of the domain subfile contained in a binary library. This subfile is created when the first procedure of a domain is created using the DOMAIN, BINLIB and CREATE commands of MAINTAIN_COMMAND.

At execution, the domain subfile is sought in the binary search path. If the caller has not specified the binary library, then the GCLINIT primitive returns an error code, DOMAIN UNKNOWN.

7.2.2 SYSTEM Command

A SYSTEM command associates a GCL procedure of an initialized domain with a user program. Generally, a procedure will contain one SYSTEM command. Nothing prevents defining a procedure that contains none, or several SYSTEM commands. The next prompt appears after exiting from the procedure currently at the bottom of the GCL stack.

The argument of the SYSTEM command is a character string which is sent to the program in response to the GCLREAD primitive.

The user program makes no further checks:

- after the syntax of the command inside the procedure has been analyzed
- and if the string passed through the SYSTEM command is a coded string.

The same program can initialize several domains with different names, allowing domain nesting.

7.3 Programming Rules

The main operations for creating and testing an interactive program using GCL facilities are as follows:

1. Create the interactive program using READVAR and MODVAR primitives to pass parameters or to help debugging.
2. Create the GCL command EXEC_PG to invoke the program. This command procedure belongs to the IOF domain in the user or project private BIN library. Use the OPTIONS keyword of the EXEC_PG command or global variables to pass parameters to the procedure.
3. Create the commands of the processor domain. This domain must be stored in the private binary library. The QUIT command must be created. The entry "/" returns an IGNORE code to the calling GCLREAD primitive.
4. Place the binary library containing the procedures (created in Steps 2 and 3) in the binary search path, using the MWINLIB BIN command. The procedures are then available.
5. Invoke the processor with the command created in step 2.
6. Debugging is performed with:
   - the PCF facility for the user program
   - the DEBUG and TRACE system variables for the GCL procedures.
7. After debugging, delete the DEBUG option from the EXEC_PG command and remove trace functions in the program using variables. Lock the procedures if no further modification is desired.

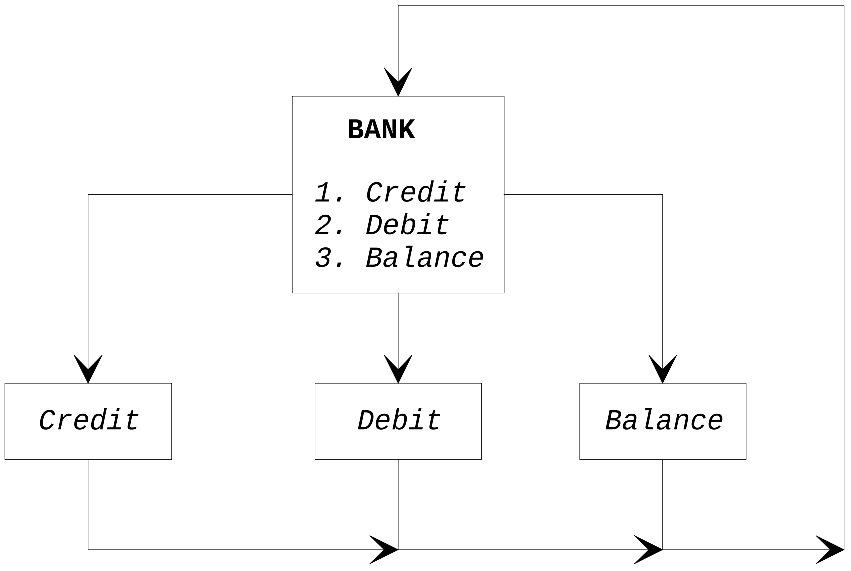
7.4 Example of Application

The following example shows how to create an interactive COBOL program which dialogs with the terminal using the GCL facility, the entities defined being:

- a COBOL program BANK
- a procedure BANK in the domain IOF that calls the preceding program
- a domain BANK containing three procedures activated by the program.

The user-defined domain BANK contains three commands; CREDIT, DEBIT and BALANCE. Associated with each command there is a section of the COBOL program to the processing necessary to handle the operation concerned.

The components of the application are shown as follows:



The following three types of operation are handled by the application:

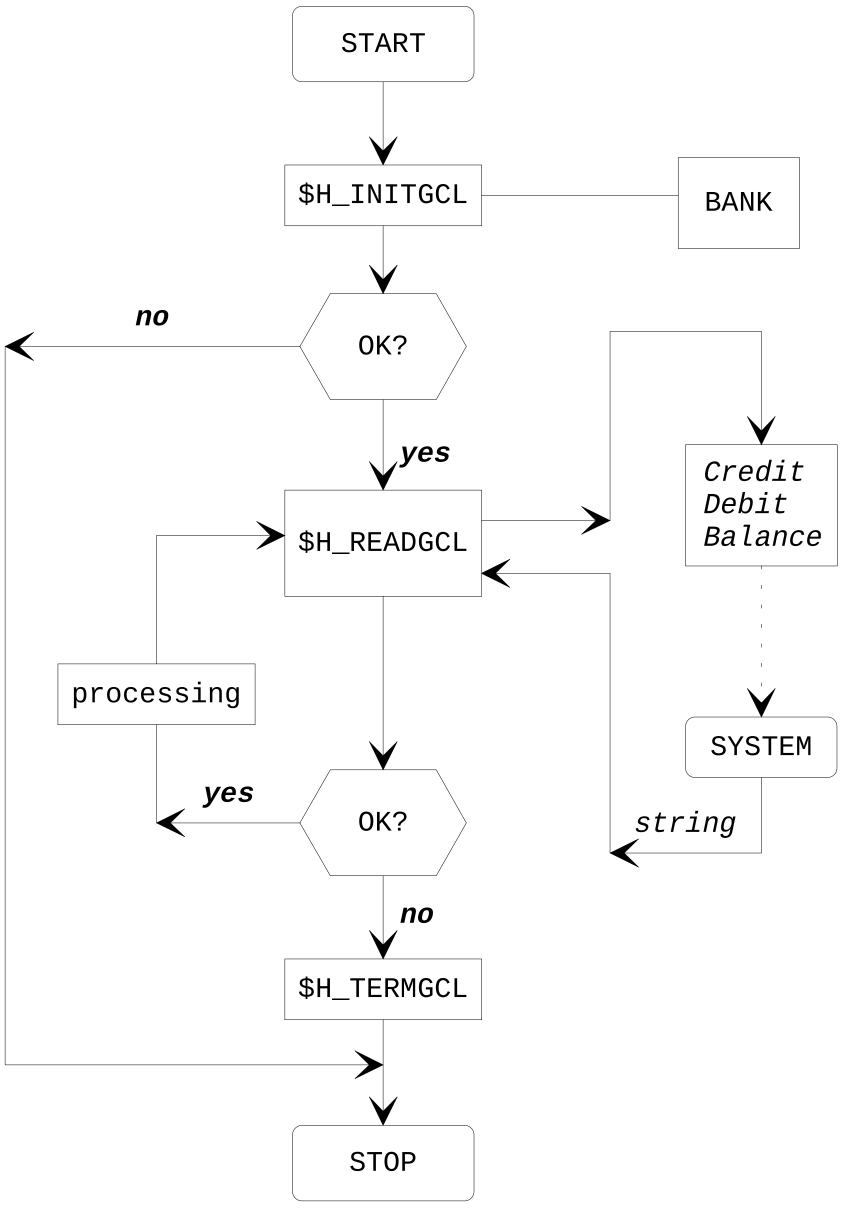
- CREDIT to credit an amount to an account.
- DEBIT to debit an amount to an account.
- BALANCE to display the balance of an account.

Detailed processing for each operation is done in the COBOL program is shown overleaf.

The user-defined GCL domain BANK contains a command for each operation. The COBOL program activates this domain (by means of the CGCLINIT command). The program reads the operation entered by the user by means of the CGCLREAD statement.

To select an operation, the user selects the appropriate command from the BANK domain level menu (menu mode) or enters the command's name (non-menu mode). The command chosen and its parameter value(s) are transmitted to the COBOL program.

The Flowchart of the Application is as follows:



7.4.1 Programming in COBOL

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.     BANK.
********************************************************
*  * THE MAIN AIM OF THIS PROGRAM IS TO ILLUSTRATE  *  *
*  * THE GCL PROGRAMMATIC INTERFACES USED FROM      *  *
*  *                A COBOL PROGRAM                 *  *
********************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  LEVEL-64.
 OBJECT-COMPUTER.  LEVEL-64.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
   SELECT BANK ASSIGN TO BANK.
 DATA DIVISION.
 FILE SECTION.
 FD  BANK LABEL RECORD STANDARD.
 01  BANK-REC.
   02  FILLER PIC X(200).
 WORKING-STORAGE SECTION.
 01  DOMAIN PIC X(31) VALUE "BANK".
 01  INT-DOM COMP-2.
 01  SUP PIC X(8).
 01  SUPP PIC X(8).
 01  PREFIX PIC X VALUE "C".
 01  ERROR-RC COMP-2.
 01  OUTLG COMP-2.
 01  BANK-FLD.
     02  BANK-CODE PIC X.
     02  CTVAR     PIC X(15).
 01  BANK-CREDIT.
     02  BANK-ID-C PIC X(3).
     02  AMOUNT-C  PIC 9(10).
 01  BANK-DEBIT.
     02  BANK-ID-D PIC X(3).
     02  AMOUNT-D  PIC 9(10).
 01  BANK-BALANCE PIC X(3).

 PROCEDURE DIVISION.
 PROC-BEGIN.
    OPEN I-O BANK.
    CALL "CGCLINIT" USING DOMAIN PREFIX INT-DOM SUP SUPP
        ERROR-RC.
    IF ERROR-RC NOT = 0 DISPLAY CONVERSION
       "INIT DOMAIN BANK MANAGEMENT NOT SUCCESSFUL ERROR-RC : "
       ERROR-RC UPON TERMINAL GO TO END-PRG.
 INPUT-BANK-MANAGEMENT.
    MOVE 16 TO OUTLG. MOVE SPACE TO BANK-FLD.
    CALL "CGCLREAD" USING INT-DOM OUTLG BANK-FLD ERROR-RC.
    IF ERROR-RC = 2 GO TO USER-END-PRG.
    IF ERROR-RC NOT = 0
           DISPLAY CONVERSION
              "READ DOMAIN BANK NOT SUCCESSFUL ERROR-RC : "
               ERROR-RC UPON TERMINAL GO TO READ-END-PRG.
    IF BANK-CODE =  "C" UNSTRING CTVAR DELIMITED BY "/"
        INTO BANK-ID-C AMOUNT-C
    DISPLAY "CREDIT THE ACCOUNT OF BANK: " BANK-ID-C UPON TERMINAL
*
*  **** SPECIFIC TREATMENT OF THE CREDIT ****
*       The statements needed to do this
*       are not shown as they are not
*       dependent on the fact that a GCL
*       interface is being used.
*
    GO TO INPUT-BANK-MANAGEMENT.
    IF BANK-CODE = "D" UNSTRING CTVAR DELIMITED BY "/"
        INTO BANK-ID-D AMOUNT-D
    DISPLAY "DEBIT THE ACCOUNT OF BANK: " BANK-ID-D UPON TERMINAL

*  **** SPECIFIC TREATMENT OF THE DEBIT ****
*       These statements are not shown.
*       see comments under CREDIT above.
*
    GO TO INPUT-BANK-MANAGEMENT.
    MOVE CTVAR TO BANK-BALANCE.
*
*  **** SPECIFIC TREATMENT OF THE BALANCE *****
*       These statements are not shown.
*       See comments under CREDIT above.
*
    GO TO INPUT-BANK-MANAGEMENT.
 USER-END-PRG.
    DISPLAY "END EXECUTION ASKED BY USER" UPON TERMINAL.
 READ-END-PRG.
    CALL "CGCLTERM" USING INT-DOM ERROR-RC.
    CLOSE BANK.
 END-PRG.
    STOP RUN.
```

The CALL "CGCLINIT" USING .... statement in the COBOL program initiates the user-defined domain named BANK.

The CALL "CGCLREAD" USING .... statement reads the command (and its parameters) entered by the user. The COBOL program tests to see which type of operation (CREDIT, DEBIT, or BALANCE) has been entered and a branch is made to the appropriate part of the program to do the processing. The statements to do this processing are not shown since they are not dependent on the fact that a GCL interface is being used.

While in the GCL part of the application, all the facilities of GCL are available. Thus for example, Help texts can be requested for each command (operation) and parameter (assuming that these have been written), controls on the values supplied can be made, error messages can be issued and retries can be done, as for any other GCL procedure. Thus the full power of GCL can be used in association with the COBOL program to create the application desired.

The loop on "CGCLREAD" continues until the user indicates that there are no more operations (by entering a /, instead of a command).

7.4.2 GCL Procedure BANK of IOF Domain

```
                         GCL COMMANDS

-> ACTIVATION OF THE BANK APPLICATION

IOF (MAY 7, 1986 13:20)
 BANK                                05/06/86  13:34  STMTS= 3

10:PROC (BANK B) PROMPT='BANK APPLICATION' HELP=BANK_HELP;
20:EXEC_PG BANK MYOWN.LMLIB FILE1=BANK MYOWN.BANK;
30:ENDPROC;
```

The three commands of the BANK domain are shown below.

7.4.3 GCL Procedures BALANCE, DEBIT and CREDIT

```
-> BANK APPLICATION

BANK (MAY 07, 1986 17:10)
 BALANCE                               05/06/86 15:44 STMTS= 4

10:PROC (BALANCE B) PROMPT=('BALANCE THE ACCOUNT');
20:KWD BANK TYPE=NAME NUMVAL=(1,1) LENGTH=3 VALUES=(BNP SG CA);
30:SYSTEM #CAT(B,%BANK);
40:ENDPROC;

 DEBIT                                 05/06/86 16:49 STMTS= 5

10:PROC (DEBIT D) PROMPT=('DEBIT THE ACCOUNT');
20:KWD BANK TYPE=NAME NUMVAL=(1,1) LENGTH=3 VALUES=(BNP SG CA);
30:KWD AMOUNT TYPE=DEC NUMVAL=(1,1) LENGTH=10 VALUES=>0 CONCEAL;
40:SYSTEM #CAT(D,%BANK,/,%AMOUNT,/);
50:ENDPROC;

 CREDIT                                05/06/86  16:54 STMTS= 5

10:PROC (CREDIT C) PROMPT=('CREDIT THE ACCOUNT');
20:KWD BANK TYPE=NAME NUMVAL=(1,1) LENGTH=3 VALUES=(BNP SG CA);
30:KWD AMOUNT TYPE=DEC NUMVAL=(1,1) LENGTH=10 VALUES=>0 CONCEAL;
40:SYSTEM #CAT(C,%BANK,/,%AMOUNT,/);
50:ENDPROC;
```

7.4.4 Equivalent Programming in GPL

```
BANK: PROC;

DCL      INTDOM                PTR;
DCL      TEXT                  CHAR(256);
DCL      TEXT_LG               FB31;

DCL 1    CMD     DEF TEXT,
   2   COMMAND               CHAR(1),
   2   BANK                  CHAR(3),
   2   AMOUNT                CHAR(10);

   $H_INITGCL DOMAIN="BANK" INTDOM=INTDOM PREFIX="B";
   IF $H_TESTRC ABNORMAL;
      THEN
      /* INIT ERROR */
      RETURN;

   DO FOREVER;
   TEXT_LG = MEASURE (TEXT);
   $H_READGCL INTDOM=INTDOM OUTLEN=TEXT_LG OUTAREA=TEXT;
   IF $H_TESTRC IGNORE; THEN DO;
              $H_TERMGCL INTDOM=INTDOM;
              IF $H_TESTRC ABNORMAL;
                 THEN
                 /* TERM ERROR */
              END;
              RETURN;
   IF $H_TESTRC NORMAL; THEN DO;
              SELECT (COMMAND);
                       WHEN ("C") DO;
                       /* PROCESSING OF CREDIT COMMAND */
                       END;
                       WHEN ("D") DO;
                       /* PROCESSING OF DEBIT COMMAND  */
                       END;
                       WHEN ("B") DO;
                       /* PROCESSING OF BALANCE COMMAND*/
                       END;
              END;
   END;
   ELSE
   /* READ ERROR */
   RETURN;
   END;
END BANK;
```

7.5 Help Text Handling

Together with the GCL interface described in Section 3 of this manual, a set of primitives is provided to permit communication between an interactive program and other GCL facilities.

Some of them, such as Help invocation or GCL variable management, can be used from inside a program which does not use the GCL interface. These primitives are given in COBOL, FORTRAN and GPL.

Others help the programmer in analyzing complex structures such as file literals, fileset literals, or star names. These features are normally used with the GCL interface, in order to decode strings returned in response to a GCLREAD request. These primitives are given in GPL.

7.5.1 Definition of a Help Text

A Help text is an explanatory text available on-line to help the user in using the system or in understanding a particular concept. Since the system knows the context in which a user is operating, the text displayed is as specific as possible to this context.

The system supports Help texts written in up to 10 languages. The version displayed is determined by the value of the system variable #LANG in the user's profile. If the text requested is not available in the national language, then the English version is displayed instead.

Help texts may be classified as falling in one of the following four categories:

1. Domain level.
2. Command level.
3. Parameter level.
4. Other.

A domain level Help text is associated with a domain of the system such as LIBMAINT and FORMGEN, and provides general information on the rules that pertain to that domain. The name of such a Help text is the name of the domain. A list of the standard domains delivered with the system is given in Section 5.

A command level Help text is associated with a command. It provides information on the use and purpose of the command. Its name is provided by the HELP parameter of the PROC statement in the GCL procedure which defines the command.

EXAMPLE:

```
PROC COPY
   HELP=EXPLAIN_COPY;
```

❑

The command level Help text is called EXPLAIN_COPY.

A parameter level Help text associated with a parameter of a command explains its use and purpose, and gives its possible values. Its name is provided by the HELP parameter of the KWD statement in the GCL procedure which defines the command.

EXAMPLE:

```
KWD ORDER
    TYPE=NAME
    VALUES=(ASC,DESC)
    DEFAULT=ASC,
    HELP=EXPL_SORTORDER;
```

❑

The parameter level Help text associated with ORDER is called EXPL_SORTORDER.

Help texts with user-defined names may be defined for other categories.

7.5.2 Requesting a Help Text

Requesting a Help text depends on the level of the Help text and on whether the terminal is serial or full screen.

A Domain level Help text can be requested only when the domain level menu is displayed on a screen terminal. Request the Help text either by entering a question mark (?) in the selection field, or by pressing a "Help" function key if available. The selection field is indicated by "-->: __" in the lower right-hand corner of the domain level menu. A domain level Help text provides an abridged user manual for the domain.

A Command level Help text is requested by entering a question mark (?):

- either immediately followed by a command name, for example, ?BUILD_FILE
- or in the control field when a command level menu appears, or by pressing a "Help" function key. The control field is indicated by "-->: __" in the top right-hand corner of the command level menu.

A command level Help text provides a brief description of the function(s) of the command, a list of its parameters (in most cases), and one or more examples.

A parameter level Help text is requested by entering a question mark (?) instead of a value for a parameter that is being prompted. It explains the purpose of the parameter, its relation to other parameters, and gives examples of possible values.

It is recommended that user-defined Help texts can also be requested by entering a question mark (?) in a specific field, or by pressing a "Help" function key.

7.5.3 Help Operations

A Help text is always presented as a series of screens. Once a screen is displayed, an action is requested from the user through a prompt (+++). Choose one of the following:

- Return to the place where the Help text was requested by:
  - typing a slash (/),
  - pressing a "Break" or "End-of-current-action" function key,
  - or asking for the next screen (see below) when the screen being displayed is the last one.
- Display the preceding screen, by typing a "<" character, or by pressing a "Backwards" function key.
- Display the following screen by entering one of the following:
  - a carriage return (serial terminals), or transmit (screen terminals)
  - a greater-than character (>)
  - a "Forwards" function key
  - any other string of characters that is not a single slash or a single less-than character.
- Ask for Help by entering a question mark (?).
- Call the bug/remark report mechanism by typing an asterisk (*).

7.5.4 Conventions

All Help texts are stored in two source language (SL) libraries:

- SYS.HELP for help texts associated to the standard domains.
- SITE.HELP for help texts created by users. All Help Texts have the same presentation.

Since the same Help text may exist in different national languages, the following convention defines the version corresponding to the user's declared national language:

- Names of subfiles in SYS.HELP and SITE.HELP are those of the Help texts suffixed by a digit from 0 through 9, 0 being English, 1 through 9 being alternate national languages available on the site.
- When help is requested, the national version declared in the system variable #LANG appears. If #LANG is omitted, English is the default.

Except when creating a Help text (see below), do not specify the suffix of a Help text. For example, the GCL procedure description header,

```
PROC COPY HELP=COPY;
```

implicitly refers to text subfile COPY0 for English, to COPY1 for the next language alternative and so forth.

7.5.5 CREATE_HELP_TEXT: CRHELP

This processor facilitates the creation of user-defined Help texts in the library SITE.HELP.

The source form of new Help texts are written using FSE or EDIT. The first line of the source text must contain the control HELP and the title of the text:

```
.HELP 'title of Help text';
'body of Help text'
```

This title will be repeated at the top of each screen of composed text.

For the desired layout of the composed text, insert formatting controls within the source text. These formatting controls are discussed below.

Compose the source text using the system command CREATE_HELP_TEXT (CRHELP). This command places the composed text in the SITE.HELP library.

Syntax:

```
CRHELP   MEMBERS = {member-name star-name}

        [LANG = language code]

        [INLIB = input-library-description];
```

Parameters:

MEMBERS	(list of) source texts to compose.
LANG	the language as defined in GCL. The default is 0, which means English. The languages corresponding to the other values are installation-dependent.
INLIB	library containing source texts. Default: Current value of system variable #SLIB.

7.5.6 Formatting Controls

Insert formatting controls (or commands) in the source Help texts for the desired layout. These controls start with a period (.) character in the first position of a line. Several controls can be placed on the same line, but in this case they must be separated by semicolons (;).

Some of the formatting controls are:

.help 'mine'	This specifies that the Help's title is 'mine'. It ensures that the title and page (or screen) number appears on each page (or screen). Enter the .help control on the first line of a source Help text.
.spb 1	This starts a new paragraph after skipping 1 line. 1 after .spb can be omitted (.spb is equivalent to .spb 1).
.spb2	This starts a new paragraph after skipping 2 lines.
.brp	This forces a page break. The text following will start on a new screen.
.inl 5	This indents the text 5 positions from the left. This indentation continues until canceled by another .inl.
.unl 3	This "un-indents" text 3 positions to the left. In contrast to .inl, .unl is effective only for the text line immediately following it.
.fif	This turns off fill mode. The text that follows is left as is until this control is canceled by a .fin control.
.fin	This cancels a preceding .fif control.

7.5.7 Examples of Help Texts

7.5.7.1 Source Text

The following is an example of a source Help text (that is, with the formatting controls still included).

```
.help 'a sample source text'
This is the first line.
.spb 1
This is a new paragraph.
The text of a single sentence can be split over several lines.
.spb
This is the same as specifying .spb 1.
.spb 2
Do not start a line with a space (or blank) character.
To indent the text from the left, use the .inl control.
.spb 3
Start a new paragraph after skipping 3 lines.
.brp
Start a new paragraph on a new page (screen).
.spb
.inl 8
.unl 3
Start indentation (from the left) at position 9.
So there will be 8 spaces at the start of each line.
However, the line immediately following the .unl is "un-indented" 3 positions to
the left.
.spb 1
.unl 6
This line in "un-indented" 6 positions.
.spb 1
The .inl is effective until canceled.
.spb 1
.inl 0
The .inl 0 resets the left hand margin, that is, it stops the indentation from the left.
.spb 1
.fif
The .fif means that the following
text is not composed, that is, it is
left as is.
.spb 2
.fin
```

Use the .fin control to cancel the .fif control.
.spb
Example:
.spb 1
.inl 25
.unl 20
SYNTAX ON THE LEFT The explanation can be placed on the right hand side.
If necessary, this explanation can continue on several lines.
The .inl and .unl controls ensure that the layout will be as desired.

7.5.7.2 Composed Text

Compose the source text as follows:

```
CREATE_HELP_TEXT MEMBERS=MYTEXT LANG=0 INLIB=MY.SL1;
```

`where MYTEXT is the name of the text.`

The composed Help text obtained from the source text presented above is shown below. Note that the line length of this manual is 68 characters whereas the line length of a Help text displayed on a screen terminal is usually 78 characters. The text presented below is based on a line length of 68 characters.

Typing in the above source text and displaying it on a screen terminal, gives a presentation slightly different due to the 10 character difference in line length.

The composed version of this Help text is given below.

1/3 A SAMPLE SOURCE TEXT

This is the first line.

This is a new paragraph. The text of a single sentence can be split over several lines.

This is the same as specifying .spb 1.

Do not start a line with a space (or blank) character. To indent the text from the left, use the .inl control.

Start a new paragraph after skipping 3 lines.

2/3 A SAMPLE SOURCE TEXT

Start a new paragraph on a new page (screen).

Start indentation (from the left) at position 9. So there will be 8 spaces at the start of each line. However, the line immediately following the .unl is "un-indented" 3 positions to the left.

This line in "un-indented" 6 positions.

The .inl is effective until canceled.

The .inl 0 resets the left hand margin, that is, it stops the indentation from the left.

The .fif means that the following text is not composed, that is, it is left as is.

3/3 A SAMPLE SOURCE TEXT

Use the .fin control to cancel the .fif control.

EXAMPLE:

SYNTAX ON THE LEFT	The explanation can be placed on the right hand side. If necessary, this explanation can continue on several lines. The .inl and .unl controls ensure that the layout will be as desired.

❑

Note the use of the .inl and .unl controls to place the syntax of an example on the left-hand side of the text and the associated explanatory text on the right-hand side.

The explanatory text can extend as long as needed.

The rest of this section consists of an example of a domain level Help text, followed by an example of a command level Help text, and ending with an example of a parameter level Help text.

7.5.7.3 Domain Level Help Text

The following is an example of a domain level Help text.

1/2 FULL SCREEN EDITOR

You are now in the domain of the Full Screen Editor (FSE); a powerful editor with which you can create and modify source texts in source libraries using predefined applications (defined by ADL). FSE has two modes of operation: an input mode and an edit mode.

In input mode you enter complete records that are used to build up a new text unit to be inserted in an existing member or used to replace existing lines. You may define the format of the records you enter, the way they are to be prompted on the screen, and the order in which they are to be transmitted to the calling program (selective sequence), in an application.

In edit mode you may specify the actions to be performed on an existing text. These actions are available through edit requests that you enter in a predefined field on the screen. The results of the requested actions appear immediately on the screen, where you may check to see if they have had the desired effect.

2/2 FULL SCREEN EDITOR

FSE can operate on up to four source libraries at a time; one is the output library, the others are input libraries. FSE commands that read members from the libraries operate on the input and the output libraries; whereas FSE commands that modify a library operate only on the output library.

You can assign the input libraries before entering FSE by the GCL command MWINLIB, or you can assign them within FSE by the commands INLIB1, INLIB2, and INLIB3. You can assign the output library before entering FSE by the GCL command MWLIB or by the keyword LIB of the FSE statement, or you can assign it within FSE by the LIB command.

Libraries are referred to in FSE by their symbolic names: LIB, INLIB1, INLIB2, and INLIB3. If you refer to a library that has not been assigned, an error message is displayed.

7.5.7.4 Command Level Help Text

The following is an example of a command level Help text.

1/3 LINK AN EXECUTABLE MODULE

The LINK PG (abbreviation LK) command is used to create an executable load module (or sharable module) from a set of one or more compile units.

The LM parameter specifies the name of the resulting load module (or sharable module).

The SM parameter indicates whether a load module or a sharable module is to be produced.

The INLIB parameter specifies the input library which contains the compile unit(s) to be linked.

The COMFILE parameter specifies the file which contains the commands to control the execution of LINK_PG.

2/3 LINK AN EXECUTABLE MODULE

The COMMAND parameter is used to specify directly a set of commands to control the execution of LINK_PG.

The ENTRY parameter specifies the entry point of the load module.

The LIB parameter specifies the library in which the load module or sharable module produced by LINK_PG is to be stored.

The PRTLIB parameter specifies the library which is to contain the listing(s).

For more details, see the LINKER User's Guide.

Examples are given on the next screen.

3/3 LINK AN EXECUTABLE MODULE

Examples:

```
LINK LM=LMOD1; Create the load module LMOD1.
```

```
LINK LM=PRTEST LIB=P1.LM3 COMFILE=P1.B1..CF;
```

Create the load module PRTEST, store it in the LM library P1.LM3. The command file is the member CF of the SL library P1.B1.

7.5.7.5 Parameter Level Help Text

The following is an example of a parameter level Help text.

1/1 EXPIRY DATE

The EXPDATE parameter specifies the expiry date of the file.

Specify it in one of the following forms:

```
yy/mm/dd     that is, year/month/day
yy/ddd       that is, year/day
ddd          that is, days (after today)
```

The default is that the expiry date is the same as today's date.

EXAMPLES:

```
89/12/31        31 December 1989

89/138          18 May 1989 (that is, the 138th day of
                1989).

138             138 days after today.
```

❑

7.5.8 HELP Primitive

Purpose:

To display a Help text.

Syntax:

GPL: `$H_HELP i_char32;`

COBOL: `CALL "CHELP" USING NAME, ERROR-RC.`

FORTRAN: `CALL FHELP (NAME,ERROR)`

Parameters:

NAME	Name of Help text, excluding the national language suffix: GPL: i_char32 COBOL: PIC X(32) FORTRAN: CHAR*32
ERROR or ERROR-RC	COBOL: COMP-2 FORTRAN: INTEGER

Return Codes:

Normal	DONE
Abnormal	SFNUNKN: subfile unknown: Help text does not exist Plus all return codes for the file containing Help texts and for the Terminal Access Method.

Error Codes (COBOL and FORTRAN):

0	function performed correctly
256	no Help text found
257	system error

EXAMPLE (GPL):

```
RECEV:
      CALL H_FRMRECV;
      IF $FRM FIELD 1; = "/" THEN GOTO ENDDELETE;
      IF $FRM FIELD 1; = "?" THEN DO;
         $H_HELP "H_FSE RENUMBER";
         $FRM SELECT NONE;
         $FRM SELECT 1;
         $FRM FIELD 1; = " ";
         GOTO RETRY;
END.
```

❑

7.6 Managing GCL Variables

7.6.1 Global Variables

GCL global variables are declared in the GLOBAL GCL command and remain accessible during a session, unless a DELETE GLOBAL command is executed for the variable.

Global variables are referred to by name. They are defined and used in GCL commands, or in external calls from programs written in one of these languages: COBOL, FORTRAN and GPL. GCL global variables thus allow communication between two GCL procedures, or between a GCL procedure and a program, or between two programs.

7.6.2 System Variables

System variable names are prefixed with a #. They determine the visibility that the programmer has of the system such as Printing Width (PW) and National Language (LANG). They are listed and presented in the *IOF Terminal User's Reference Manual*.

Any user may read or change these values. Setting a new value results in a new operating environment. For example, if LANG is modified, Help texts and error diagnostics will be displayed in the specified language instead of in English.

7.6.3 GCL Variable Primitives

Two functions are provided for using Global and System Variables:

- READVAR accesses a global or system variable, and processes the following builtins as system variables:

```
#BILLING      #LSYS         #RON         #USERID
#CPU          #MDAY         #TERMID      #WDAY
#DATE         #MODE         #TIME        #YDAY
#ELAPSED      #PROJECT      #TTYPE       #FW
#EXTDATE
```

- MODVAR modifies a global or system variable.

All global variables and system variables are lists. A scalar value is a particular case of a list with only one element. READVAR and MODVAR access only one element in the list which is selected through an index denoting its rank. Both these primitives require an index value as well as the name of the global or system variable to be accessed.

7.6.3.1 READVAR

Purpose:

To access a global or system variable.

Syntax:

```
GPL:     $H_READVAR i_char31 [,INDEX=i_fb31 ],
                    OUTLEN=b_fb31,OUTAREA=o_charn;

COBOL:   CALL "CREADVAR" USING NAME,INDEX-ID,OUTLEN,OUTAREA
                               ,ERROR-RC.

FORTRAN: CALL FREADVAR (NAME,INDEX,OUTLEN,OUTAREA,ERROR)
```

Parameters:

NAME	Type of variable:
	– system variable if it begins with # such as #MENU
	– otherwise, global variable such as MY_GLOBAL.
	`GPL:     i_char31`
	`COBOL:   PIC X(31)`
	`FORTRAN: CHARACTER*31`

INDEX or INDEX-ID	Index of element in list; scalar variable has only one element:
	`GPL:     i_fb31`
	`FORTRAN: INTEGER`
	`COBOL:   COMP-2`

OUTLEN	Length of output area in input or data read <= 255 in output:
	`GPL:     b_fb31`
	`COBOL:   COMP-2`
	`FORTRAN: INTEGER`

OUTAREA	Field to receive the value of the variable, left justified.
	`GPL:     o_charn`
	`COBOL:   PIC X(n)`
	`FORTRAN: CHARACTER*n`

ERROR or ERROR-RC	`COBOL:   COMP-2`
	`FORTRAN: INTEGER`

Return Codes:

Normal	`DONE:`	function performed correctly
	`TRUNC:`	truncation (output area too short)
Abnormal	`ARGERR:`	argument error: some argument is wrong
	`INDERR:`	invalid index or empty variable
	`LNERR:`	length error - length exceeds maximum allowed
	`NAMEERR:`	variable name unknown
	`RESNAV:`	result not available: global table not accessible
	`UNCNAV:`	function not available in batch mode

Error Codes (COBOL and FORTRAN):

0	function performed correctly
1	truncation due to insufficient output area
119	function not available in batch
256	variable name unknown
257	invalid index
258	empty variable
259	length erroneous
264	argument error
300	system error

EXAMPLE (GPL):

```
$H_READVAR '"#GCLFORM"', OUTLEN = 1,
OUTAREA = GCLFORM;
```

❑

7.6.3.2 MODVAR

Purpose:

To modify a global or system variable.

Syntax:

GPL:
```
$H_MODVAR i_char31 [,INDEX=i_fb31 ],
          INLEN=i_fb31,INAREA=i_charn;
```

COBOL:
```
CALL "CMODVAR" USING NAME,INDEX-ID,INLEN,INAREA
                     ,ERROR-RC.
```

FORTRAN:
```
CALL FMODVAR (NAME,INDEX,INLEN,INAREA,ERROR)
```

Parameters:

NAME	Type of variable:
	– system variable if it begins with # such as #MENU
	– otherwise, global variable such as MY_GLOBAL.
	`GPL:     i_char31`
	`COBOL:   PIC X(31)`
	`FORTRAN: CHARACTER*31`

INDEX or INDEX-ID	Index of element in list; Default: 1:
	`GPL:     i_fb31`
	`FORTRAN: INTEGER`
	`COBOL:   COMP-2`

INLEN	Length of the input area <= 255.
	`GPL:     i_fb31`
	`COBOL:   COMP-2`
	`FORTRAN: INTEGER`

INAREA	Field to be written in.
	`GPL:     i_charn`
	`COBOL:   PIC X(n)`
	`FORTRAN: CHARACTER*n`

ERROR or ERROR-RC	`COBOL:   COMP-2`
	`FORTRAN: INTEGER`

Return Codes:

Normal	`DONE:`	H_MODVAR performed correctly.
Abnormal	`ARGERR:`	argument error: some argument is wrong
	`FUNCNAV:`	function not available in batch mode
	`INDERR:`	invalid index
	`LNERR:`	invalid length
	`NAMEERR:`	variable name unknown
	`NOMATCH:`	invalid value
	`OBJUNKN:`	environment not allowed
	`RESNAV:`	result not available: global table not accessible
	`TYPEERR:`	invalid type

Error Codes (COBOL and FORTRAN):

`0`	function performed correctly
`119`	function not available
`256`	variable name unknown
`257`	invalid index
`259`	invalid length
`260`	invalid type
`261`	invalid value
`264`	argument error
`300`	system error

7.7 File Literal Analysis

Some primitives analyze the GCL constructs known as *file literals* and *volume literals*. These constructs are used to denote a file or a volume in a manner that is both compact and easy to use.

These primitives can be used in any context where a file literal may appear, in order to check whether it is correct and to prepare the structures required to dynamically assign the file ($H_DFLASG).

For details on how to specify files, refer to the *IOF Terminal User's Reference Manual*.

An example of the use of the H_DCANFILE and H_ANFILE macros is given in the *GPL System Primitives Reference Manual*.

NOTE:
Lowercase letters are converted to uppercase unless they are within protected strings.

7.7.1 File Literal

Syntax:

```
file-literal    ::= { local-file | remote-file }

                                  { :string-1              }
remote-file     ::= $site-name { :protected-string      }
                                  { { /|!|^|>|<|. }string-2 }
```

Definition of Elements:

```
• site-name        ::= name8

• string-1         ::= any combination of up to 255 characters
                       excluding , ; = # % ' " ( )* { } [ ] &
                       ? and space

• protected-string::= GCL protected string up to
                       255 characters

• string-2         ::= like string-1 but limited to
                       254 characters

local-file      ::= can be one of the following entries:
{ cataloged-file::=

                                   {[$CAT[i]][$VOLSET[:name6]]}
  path-name[/g-suffix][..subfile]{                          }
                                   {$NATIVE                  }

| temporary-file::=
                      {[:md[/md]...:dvc] }
  path-name[..subfile]{[$RES]            }$TEMPRY
                      {[$VOLSET[:name6]]}
| permanent-uncataloged-file::=
                      {$RES [$UNCAT]               }
                      {                            }
                      {               [{$UNCAT   }]}
  path-name[..subfile]{               [{$MFT[i|+]}]}
                      {:md[/md]...:dvc[{$NATIVE  }]}
                      {               [{$NONE    }]}
                      {               [{$NSTD    }]}
| { * | SYSIN .. } input-enclosure-name
| { DUMMY | SYS.OUT }
| * : md[/md]...:dvc [{ $UNCAT | $NATIVE }]
| [path-name]:[md]:TN[/tbd][$UNCAT]     }
```

Definition of Elements:

```
• full-path-nm     ::= simple name [.simple-name]...

• rel-asc-path     ::= <[<]... simple-name [.simple-name]...

• rel-desc-path    ::= .simple-name [.simple-name]...

• path-name        ::= {full-path-nm}
                       {rel-asc-path}
                       {rel-desc-path}

• g-suffix         ::= { { Gdigit4      }              }
                       { { G{+|-}digit4 }[{ Vdigit2 ] }
                       { { G{+|-}       }              }
                       { G name5                       }
                       { Vdigit2                       }

• subfile  ::= name31

• input-enclosure-name
             ::= name16
```

7.7.2 Syntax of a Volume Literal

```
                  [ $NATIVE ]
                  [ ------- ]
                  [ $COMPACT]
md [/ md] ... :dvc [         ]
                  [ $NONE   ]
                  [         ]
                  [ $NSTD   ]
```

7.7.3 Primitives

Two macro definitions are provided for analyzing a file literal string. The first one ($H_DCANFILE) declares a structure that is passed to the second ($H_ANFILE). This structure is used to convey the input parameters to the primitive and to obtain the resulting parameters.

The following is a very simple programming example that illustrates the manner in which these two primitives are to be used:

Example of Primitive Calls:

```
$H_DCANFILE  PREFIX=''       /* declare file analysis structure */

STRING_ADDRESS=ADDR (MYZONE) /* address of zone to be analyzed  */

FIRSTCHAR=1; LASTCHAR=LEN    /* index of 1st & last characters  */

SYNTAX_OPTION="1             /* OCL syntax is not allowed       */

SUBFILE_OPTION="1"           /* subfiles are allowed            */

EXPANDPATH="2"               /* expand the path name            */

SITE_OPTION="1"              /* site option is not allowed      */

$H_ANFILE ANFILE             /* call the analysis primitive     */

IF RETURNCODE=0 THEN DO      /* no error detected               */

...
END

ELSE DO                      /* some error detected             */
...
END
```

7.7.3.1 DCANFILE

Purpose:

To declare the file analysis structure.

Syntax:

```
$H_DCANFILE [ PREFIX=l_identifier16 ] [ ATTRIB=l_char ];
```

Parameters:

PREFIX	Specifies a character string prefixed to the name of the structure and to the name of each elementary item. Default: 'H_'
ATTRIB	Specifies the attributes of the structure. Default: none

For more information, see the GPL System Primitives Reference Manual.

7.7.3.2 ANFILE

To check the description of constructs denoting a file or volume literal, analyze the options, then return the information required to dynamically assign the file (H_ASSIGN).

Syntax:

```
$H_ANFILE b_structure;
```

Parameter:

name — Name of Structure: `b_structure`
- declared by the H_DCANFILE
- and contains input and output arguments for file analysis.

Return Codes:

Normal	`DONE:`	normal execution
Abnormal	`ARGERR:`	argument error: invalid argument

7.8 Fileset Literal Analysis

The primitives described here are devoted to the analysis of the GCL construct known as *fileset literal.*

Note that in the following, lowercase letters are treated as uppercase, unless they are within protected strings.

7.8.1 Fileset Literal

Syntax:

```
Fileset-literal   : :=
star-exp:md[/md...:dvc[$UNCAT]$MFT[+|i][$REF[:file-literal
                                                     [$REF:file-literal]...]]]
star-exp [{:md[/md]...:dvc}][$UNCAT]$REF[:file-literal[$REF:file-literal]
                                                     [{$RES}]...]]
star-exp  $VOLSET[:vset-6] [$REF[:file-literal[$REF:file-literal]...]]
star-exp [{ $CAT[i]          }][$ONLY:{*|md[/md...}:dvc][$REF[:file-literal
         [{ $CAT:catalog-name }]                     [$REF:file-literal]...]]]
star-exp [{ $CAT[i]          }][$ONLY:$VOLSET[:vset-6][/vset-6]....]
                                                     [$REF[:file-
         [{ $CAT:catalog-name }]      literal[$REF:file-literal]...]]
```

Definition of Elements:

```
star-exp : := [{.|<[<]...}] {sse[.sse]...[.**[.sse]...][/suffix]
                          | **[.sse]...[/suffix]  }
sse  : := {constant-string | [constant-string]*[constant-string]}
size max = char16
suffix : := { [{Gdigit4 }]            }
          { [{G+digit4}]            }
          { [{G-digit4}]  [{Vdigit2}]  }
          { [{G+      }]  [{V*     }]  }
          { [{G-      }]  [{V**    }]  }
          { [{G*      }]              }
          { [{G**     }]              }
          { G_alphanum5               }
```

catalog-name ::=[.]simple-name [.simple-name]... [.CATALOG]
simple-name ::= alphanum16
disk-pool-name ::= char6
vset ::= volset-name6

7.8.2 Primitives

Three macro definitions are provided for analyzing a fileset literal string:

- $H_DCANFST declares a structure that is passed to $H_ANFST
- $H_ANFST takes parameters from the generated structure, analyzes the fileset literal and issues new parameters to the structure
- $H_ANFLFST applies when a file literal is embedded within the fileset literal after $REF.

7.8.2.1 DCANFST

Purpose:

to declare and generate a Fileset structure to be used as the communication area between a program and the ANFST or ANFLFST primitive.

Syntax:

```
$H_DCANFST [ PREFIX=l_identifier16 ] [ ATTRIB=l_char ];
```

Parameters:

PREFIX	Specifies a character string to be prefixed to the name of the structure and to the name of each elementary item. Default: 'H_'
ATTRIB	Specifies the attributes of the structure. Default: none

For more information, see the *GPL System Primitives Reference Manual*.

7.8.2.2 ANFST

Purpose:

To analyze the fileset literal and to fill the structure generated by DCANFST.

Syntax:

```
$H_ANFST b_structure;
```

Parameter:

name	Name of Structure: `b_structure` – generated by H_DCANFST – and contains input and output arguments of H_ANFST.

Return Codes:

These are given by the RC in the H_DCANFST structure.

7.8.2.3 ANFLFST

Purpose:

To analyze a File Literal inside a Fileset literal.

Syntax:

```
$H_ANFLFST b_structure1 FILE_STRUCT=b_structure2;
```

Parameters:

name	Name of Structure: `b_structure1` – declared by H_DCANFST – filled by H_ANFST – and used by H_ANFLFST in input and output.
FILE_STRUCT	Name of Structure: `b_structure2` – declared by H_DCANFILE for use by H_ANFLFST in input – and initialized as required for H_ANFILE. The string address, beginning and end index are to be found in the substructure FLREF of H_DCANFST.

Return Codes:

These are given by the RC in the H_DCANFST structure.

If RC=-1, another $REF:*file_literal* has been found. Repeated calls are made to H_ANFLFST to analyze the remaining file literal, until RC is different from -1.

Output of $H_ANFLFST is to the $H_DCANFST structure.

7.9 Star Convention Analysis Primitives

The star convention is a syntactical device that may be used to denote a set of names in an abbreviated manner.

Syntax of the Star Convention:

```
      { single-name } {[$>lower-value] [$<upper-value]}
[^]   {             } {                              }
      { star-name   } {[$<upper-value] [$>lower-value]}
```

where:

- *single-name* is any combination of up to 31 characters from the set {A-Z 0-9 -}
- *star-name* is any combination of characters from the set {A-Z 0-9 - *}
- *upper-value* is a single-name
- *lower-value* is a single-name.

The star convention acts as a pattern in which the star * may match any occurrence (including none) of any characters. The upper and lower values delimit the inclusive range within which the matching names must fall.

7.9.1 Primitives

Three primitives are provided for analysis of the star convention:

- $H_DCANSTAR declares a data structure to be used as an argument for the $H_ANSTAR and $H_CHKSTAR
- $H_ANSTAR analyses whether a given string of characters is a valid construct for a star convention and fills in fields of the data structure for use by $H_CHKSTAR
- $H_CHKSTAR checks whether a given name matches the star convention of the data structure used by $H_ANSTAR.

7.9.1.1 DCANSTAR

Purpose:

To declare and generate a STAR convention structure to be is filled by H_ANSTAR and used by H_CHKSTAR.

Syntax:

```
$H_DCANSTAR [PREFIX=l_identifier16] [ATTRIB=l_char];
```

Parameters:

PREFIX	Character string: `l_identifier16` – to be prefixed to the name of the structure – and to the name of each elementary item. Default: 'H_ '
ATTRIB	Attributes of the structure: `l_char` Default: none

For more information, see the *GPL System Primitives Reference Manual*.

7.9.1.2 ANSTAR

Purpose:

To analyze a STAR convention and fill the structure generated by H_DCANSTAR.

Syntax:

```
$H_ANSTAR i_char63, OSTRUCT=o_structure;
```

Parameters:

string	String containing the star convention: `i_char63` Example: `*NST*$>CA$<PRS`
OSTRUCT	Name of the Structure: `o_structure` – generated by H_DCANSTAR – and filled by H_ANSTAR: SINGLENAME: – "Y" the *i_char63* contains a single-name – "N" the *`i_char63`* contains a valid star convention. NOT: – "Y" first character of *`i_char63`* was character ^ – "N" first character of *`i_char63`* was not character ^. ERRORINDEX: index of erroneous character in `i_char63` – NAME: star name – FIRST: lower-value or "00000000..."H – LAST: upper-value or "FFFFFFFF..."H

Return Codes:

Normal	`DONE`	
Abnormal	`ARGERR:`	Wrong star string syntax -ERRORINDEX shows position of wrong character.

7.9.1.3 CHKSTAR

Purpose:

To check if a name (STAR convention) matches a given star convention. For example, if INSTANT matches the `*NST*$>CA$<PRS` star convention.

Syntax:

```
$H_CHKSTAR i_char31 ISTRUCT=i_structure;
```

Parameters:

string	String containing the name to be checked: `i_char31`
ISTRUCT	Name of the Structure: `i_structure` – declared by H_DCANSTAR – and filled in by H_ANSTAR.

Return Codes:

Normal	`DONE`	
Abnormal	`NOMATCH:`	name does not match
	`ARGERR:`	argument error

7.10 Job Submission

The Input Reader provides interfaces for SYNCHRONOUS or ASYNCHRONOUS submission of jobs for programs written in: GPL and COBOL.

7.10.1 Synchronous Job Submission

Syntax:

GPL:

```
$H_RUN JOBDESC_PTR=I_JOBDESC_PTR  RESULT_PTR =
IO_RESULT_PTR;
```

RETURN CODE G4:

DONE	No error occurs at job introduction.
ARGERR	invalid job description structure and/or invalid result structure (see $H_DCJOBDESC, $H_DCEJRRUNOUT).
OPTERR	errors found in the job description.

COBOL: CALL “H_IN_UEJR” USING JOB-DESCRIPTION RESULT.

7.10.2 Asynchronous Job Submission

Syntax:

GPL:

```
$H_SUBMIT JOBDESC_PTR=I_JOBDESC_PTR
RESULT_PTR=IO_RESULT_PTR;
```

RETURN CODE G4:

DONE	No error occurs at job introduction.
ARGERR	invalid job description structure and/or invalid result structure (see $H_DCJOBDESC, $H_DCEJRRUNOUT).
OPTERR	errors found in the job description.

COBOL:

```
CALL "H_IN_UEJR" USING JOB-DESCRIPTION RESULT.
```

7.10.3 Description of Parameters

I_JOBDESC_PTR	Pointer to an input structure JOB_DESCRIPTION declared by $H_DCJOBDESC (GPL macro).
IO_RESULT_PTR	Pointer to an input-output structure RESULT declared by $H_DCEJRRUNOUT (GPL macro).
JOB_DESCRIPTION	Input data structure that contains the set of parameters applicable to the submitted job. All parameters must be initialized. Blank values mean default values. This structure is declared by the macro $H_DCEJRRUNOUT in GPL language.
RESULT	Input-output data structure that contains the result of the job submission. The input parameters are the size of the structure, and the number of errors. The size parameter must be at least the size of the structure. The error number parameter must be initialized to zero. The structure is declared by the macro $H_DCEJRRUNOUT in GPL language.

For a description of H_DCJOBDESC and H_DCEJRRUNOUT, see the *GPL System Primitives Reference Manual*.

7.10.4 Information About the Launched Job

Syntax:

GPL: `$H_JOBINFO JOBSTRUCT=b_jobstruct;`

RETURN CODE G4:

DONE	Function is correct.
ARGERR	Invalid argument REQID.
NOMATCH	The job has not yet been launched or is no longer known to the system.
SYSOVLD	Overflow on a system table.

COBOL: `CALL "H_CBL_UJOBINFO" USING JOBSTRUCT.`

Comments:

The CALL “H_CBL_UJOBINFO” can follow either a CALL “H_IN_ISUBMIT” or a CALL “H_IN_UEJR”.

Description of Parameters:

JOBSTRUCT	Input/output structure declared by H_DCJOBINFO (GPL primitive). In input, only REQID is filled by the REQID-RON(1) of H_DCEJRRUNOUT (GPL macro). All other fields of this structure are filled by the H_JOBINFO primitive.

For a description of H_DCJOBINFO and H_DCEJRRUNOUT, see the *GPL System Primitives Reference Manual*. The significance of all fields of H_DCJOBINFO are given in this manual.

7.10.5 COBOL Equivalents

The COBOL equivalents for GPL values are:

GPL values	COBOL values
CHAR(nn)	PIC X(nn)
FIXED BIN(15)	COMP-1
FIXED BIN(31)	COMP-2

Example: declarations equivalent to $H_DCJOBDESC and $H_DCEJRRUNOUT

```
*** Member JOBDESC-COB ***


 01 JD-JOB-DESCRIPTION.
  03  JD-JOB-SOURCE.
      05  JD-MEMBERS                  PIC X(31).
      05  JD-FILE                     PIC X(78).
      05  JD-SELECTION-FLAG           PIC X.
      05  JD-JOBS-SELECTION.
          07  JD-JOB-ID1              PIC X(8).
          07  JD-JOB-ID2              PIC X(8).
      05  FILLER                      PIC X(18).
  03  JD-JOB-ATTRIBUTES.
      05  JD-JOBLANG                  PIC X.
      05  JD-HOLDOUT                  PIC X.
      05  JD-HOLD                     PIC X.
      05  JD-JOR                      PIC X.
      05  JD-LIST                     PIC X.
      05  JD-JOBCLASS                 PIC X(2).
      05  JD-PRIORITY                 PIC X.
      05  JD-STARTUP                  PIC X.
      05  JD-REPEAT                   PIC X.
      05  JD-DELETE                   PIC X.
      05  JD-HOST                     PIC X(4).
      05  JD-SWITCH.
          07  JD-PASS                 PIC X.
          07  JD-SWITCHES             PIC X(32).
      05  JD-EXPVAL                   PIC X.
      05  JD-NOMESSIOF                PIC X.
      05  FILLER                      PIC X(14).
  03  JD-JOB-SUBMITTER.
      05  JD-COMMIT                   PIC X.
      05  JD-COMMITMENT-ID.
          07  JD-PROCESSOR-ID         PIC X(4).
          07  JD-COMMIT-ID            COMP-2.
          07  JD-TPR-ID               COMP-1.
```

```
        05  JD-SUBMITTER-ID.
            07  JD-USER                 PIC X(12).
            07  JD-PROJECT              PIC X(12).
            07  JD-BILLING              PIC X(12).
            07  FILLER                  PIC X(12).
        05  FILLER                      PIC X(22).
    03  JD-JOB-OUTPUTS.
        05  JD-DESTINATION.
            07  JD-PRIMARY-STATION      PIC X(8).
            07  JD-SECONDARY-STATION    PIC X(8).
        05  JD-BANNER                   PIC X.
        05  JD-BANINF.
            07  JD-BANINF1              PIC X(12).
            07  JD-BANINF2              PIC X(12).
            07  JD-BANINF3              PIC X(12).
            07  JD-BANINF4              PIC X(12).
        05  FILLER                      PIC X(15).
    03  JD-GCL-ARGUMENTS.
        05  JD-DEBUG-GCL                PIC X.
        05  JD-TRACE-GCL                PIC X.
        05  JD-SEV                      PIC X.
        05  FILLER                      PIC X(12).
        05  JD-H-BATCH-OPTIONS.
            07  FILLER                  PIC X.
            07  JD-REPEAT-STEP          PIC X.
            07  JD-JOURNAL              PIC X.
            07  JD-DUMP                 PIC X.
            07  JD-PRIVATE-DUMP         PIC X.
            07  FILLER                  PIC X(14).
            07  JD-PRTFILE              PIC X(78).
        05  FILLER                      PIC X(16).
    03  JD-OTHERS.
        05  FILLER                      PIC X(32).
    03  JD-VALUES-DESCRIPTION.
        05  JD-VALUES-LENGTH            COMP-1.
        05  JD-VALUES-STRING            PIC X(3000).
```

Comments

MEMBERS	Name of the subfiles of FILE containing the GCL/JCL jobs to be submitted. A set of subfile can be specified using star convention name.
FILE	Mandatory parameter: name of the library or sequential file or remote file that contains the submitted jobs. It is a GCL file literal.
SELECTION_FLAG	Determines job selection: = "1" a sequence of jobs will be selected from JOB_ID1 through JOB_ID2. = "0" or " " Defaults: no job selection:
JOB_ID1	Determines first job selected: if the name does not exist or if JOB_ID1=" ", every job selected up to JOB_ID2. otherwise the first selected job is JOB_ID1.
JOB_ID2	Determines last job selected: if the name does not exist or if JOB_ID2=" ", jobs are selected from to JOB_ID1. otherwise the JOB_ID2 is the last job selected.
JOBLANG	Submitted job language: = "G" GCL = "J" JCL = " " Language depends on parameter JOBLANG of the $JOB card.
HOLDOUT	How output is processed: = "1" outputs produced by the jobs are held until released by the RELEASE_OUTPUT directive. = "0" or " " Defaults: output are printed.
HOLD	How job is processed: = "1" job is held until released by the RELEASE_JOB directive. = "0" or " " Defaults: job is eligible for execution.
JOR	if and how JOR is produced: = "A" a JOR is produced when the job aborts = "N" no JOR produced = "Y" or " " defaults: JOR is produced on job termination.

LIST	Information types to appear in the JOR: = "A" Information 1, 2, 3, 4, 5, 6 and 7 = "N" Information 2, 6 and 7 = "S" or " " Defaults: Information 1, 2, 3 and 7 Information types: 1 sources of GCL/JCL 2 input reader statements 3 records inserted using $SWINPUT with CONSOLE 4 statements got from $INVOKE/$SWINPUT 5 startup 6 comments 7 error messages
JOBCLASS	Class of submitted jobs: either "A" through "P" or "AA" through "PZ"
PRIORITY	Priority of the submitted jobs: ranging from"0" through "7" the highest priority being "0", the lowest "7" = for default priority.
STARTUP	Determines if startup is to be executed: = "0" Do not execute optional startup = "1" or " " Defaults: execute the startup sequence from SITE.STARTUP.
REPEAT	Determines if job is to be repeated: = "1" repeat the job if it is canceled, or if there is a system failure or abort. = "0" or " " Defaults: do not repeat the job.
DELETE	Applicable only if the input comes from a source library: = "F" after the job introduction, the subfile is deleted = "Y" the subfile is deleted only if the job execution is completed = "N" or " " Defaults: no subfile deleted.
HOST	Name of the host where the job is to execute (only for EJR/ SUBMIT). Blank value means local execution.
PASS	Determines if job switches are passed: = "1" pass spawner job SWITCHES. = "0" or " " Defaults: no job switches passed.

SWITCHES	List of 32 characters as "0" or "1" passed as masks to the spawned jobs. Defaults: "0" (space is interpreted as "0")
EXPVAL	Determines if value parameters are expanded in the JOR: = "1" Expand the JCL job value parameters. = "0" or " " Defaults: no expansion of value.
NOMESSIOF	Determines the destination of the messages IN, STARTED, COMPLETED and OUTPUT COMPLETED. = "Y", messages sent only to the main operator and not to the iof submitter. = "N" or " ", standard destination.
COMMIT	Determines the submitter of the job (see SUBMITTER_ID) = "1" validates COMMITMENT_ID. = "0" or " " Defaults: do not validate COMMITMENT_ID.
COMMITMENT_ID	Meaningless if COMMIT = "0". Commitment identity (reserved for TDS) PROCESSOR_ID: processor identity COMMIT_ID: COMMIT identity TPR_ID: TPR identity
SUBMITTER_ID	Submitter depending on setting of COMMIT: COMMIT = "1" the submitter of the job (TDS) COMMIT = "0" or " " job owner (secondary submitter)
DESTINATION	Specifies the destination of the output(s): PRIMARY_STATION: host name SECONDARY_STATION: station name
BANNER	Determines if output banners are generated: = "0" Do not generate the output banners = "1" or " " Defaults: generate the output banners (see BANINF)
BANINF	Output banners of 4 optional parameters built from bottom to top, each parameter pushing up the previous one. = " " Default: Ron, Username, Jobname and Billing.

DEBUG_GCL	Applicable only when JOBLANG=’’G’’ (Gcl) = "1" Debug GCL procedure, equivalent to the directive "MODIFY_PROFILE DEBUG": each time a line is executed, it is displayed with evaluated variables, prefixed by procedure name and line number. = "0" or " " Defaults: no debug
TRACE_GCL	Applicable only when JOBLANG = "G" (Gcl) = "1" equivalent to the directive "MODIFY_PROFILE TRACE". Traces all CALL statements executed on the current output device. = "0" or " " Defaults: no trace.
SEV	Severity level from 1 to 5, when the batch job aborts at the end of the current step if: JOBLANG="G" (gcl) and DEFAULT is the current value for ON_ERROR.
REPEAT_STEP	Applicable to step H_BATCH on abort or system crash: = "1" repeat the step H_BATCH = "0" or " " Defaults: no repeat.
JOURNAL	Applies to files accessed at system level by GCL commands: = "B" Before journal = "A" After = "O" both = "N" or " " None: default value.
DUMP	Applicable to H_BATCH for information to dump on abort: = "D" Data = "A" All = "N" or " " None: default value.
PRIVATE_DUMP	Specifies if the dump applies only to PRIVATE segments: = "1" if DUMP="D": dump all Private Data Segments (PDATA), if DUMP="A": dump all Private Segments (PALL) = "0" or " " Default: the dump is not private.
PRTFILE	Report file using File literal syntax, applicable only to the GCL job. = " " Report appears in the standard system file SYS.OUT.

VALUES DESCRIPTION	Values passed to the spawned job in the external format: VALUES_LENGTH defines in bytes the length of VALUES_STRING. VALUES_STRING is an optional list of positional values followed by an optional list of keyword values. Each value is separated by either space or comma. If there is no values specified, the values length must be set to 0.

```
*** Member EJRRUNOUT-COB ***

 01 RES-RESULT.
  03  RES-SEG-SIZE                   COMP-2.
  03  RES-START-TIME.
      05  RES-TIME                   COMP-2.
      05  RES-DATE.
          07 RES-YEAR                COMP-1.
          07 RES-YDAY                COMP-1.
  03  RES-RETCODE                    COMP-2.
  03  RES-ERROR-DESC.
      05  RES-NB-ERROR               COMP-1.
      05  RES-ERROR OCCURS 16.
          07  RES-CLASS              COMP-1.
          07  RES-NUMBER             COMP-1.
  03  RES-JOB-ENTRY-DESC.
      05  RES-NB-JOB-ENTRY           COMP-1.
      05  RES-JOB-ENTRY OCCURS 16.
          07  RES-STATE              COMP-1.
          07  RES-REQID-RON          COMP-2.
```

Comments

SEG_SIZE	Mandatory INPUT parameter containing the size in bytes of the area to receive the result of the job submitted. SEG_SIZE must be at least greater or equal than 180 bytes.
START_TIME	Contains the time in milliseconds and the date (year and day in the year) of the first submission.
RETCODE	Contains the return code issued after job submission. If DONE, the request and/or job was successfully introduced.
NB_ERROR	Number of errors found in the job description.

ERROR	Array of NB_ERROR (<=16) entries. Each entry contains the class of error CLASS and its NUMBER.
NB_JOB_ENTRY	Number of spawned jobs introduced or not, and is also the number of the entries of the array JOB_ENTRY. If the job is submitted using EJR/SUBMIT, NB_JOB_ENTRY is set to 1 and REQID_RON contains the Request Index.
JOB_ENTRY	Array of NB_JOB_ENTRY of entries depending on SEG_SIZE, each entry containing the result of one job submission.
STATUS	= 0 the request/job was successfully introduced. = 1 EJR/SUBMIT: no request introduced (see ERROR) RUN: the submitted job is aborted.
REQID_RON	EJR/SUBMIT: contains the internal request identification that may be used in DISPLAY_USER_REQUEST and CANCEL_USER_REQUEST. RUN: contains the RON of the job.

7.10.6 Obtaining Error Messages from Error Numbers and Classes

Syntax:

GPL:	`$H_LSJDERR ERROR_PTR=I_ERROR_PTR;`
COBOL:	`CALL "H_IN_UJDERR" USING I_ERROR_STRUCT I_LIST.`

Parameters:

I_ERROR_PTR — is a pointer to I_ERROR_STRUCT which contains the number of errors followed by the error array as declared in RESULT.

GPL Ex:

```
DCL I_ERROR_PTR PTR;

DCL 1 I_ERROR_STRUCT,
2 NB_ERROR  FIXED BIN(15),
2 ERROR(16),
3 CLASS     FIXED BIN(15),
3 NUMBER    FIXED BIN(15);
```

I_LIST — is an integer equal to zero.

7.10.7 Examples

Each field of the input structure JOB_DESCRIPTION must be initialized. The blank values mean default values.

The first field (SEG_SIZE) of the input/output structure RESULT must be initialized to the size of the structure. If the value of NB_ERROR is negative or greater than 16, the READER will initialize the RESULT structure.

7.10.7.1 Job Submitted on Local Site through GPL Program

Submit a JCL_JOB with values from the LIBRARY using the macro $H_SUBMIT and a GCL_JOB using $H_RUN. Hold all outputs.

```
EJR_RUN_GPL : PROC;

$H_DCJOBDESC PREFIX = K_JD_ INIT;
$H_DCEJRRUNOUT PREFIX = K_RES_ INIT;

$H_DCJOBDESC PREFIX = L_JD_;
$H_DCEJRRUNOUT PREFIX = L_RES_;

DCL L_JDPTR   PTR;
DCL L_RESPTR  PTR;

BEGIN;
 L_JDPTR = ADDR(L_JD_JOB_DESCRIPTION);
 L_RESPTR = ADDR(L_RES_RESULT);

/*****  Test of $;H_SUBMIT  *****************************************/

 L_RES_SEG_SIZE = MEASURE(L_RES_RESULT);
 L_RES_NB_ERROR = 0; /* ask the READER to initialize the result structure */

 L_JD_JOB_DESCRIPTION = K_JD_JOB_DESCRIPTION;

 L_JD_MEMBERS = "JCL_JOB";
 L_JD_FILE = "LIBRARY";
 L_JD_HOLDOUT = "1";
 L_JD_VALUES_LENGTH = 34;
 L_JD_VALUES_STRING = "TEST OF GPL MACRO NAME='$;H_SUBMIT'";

 $H_SUBMIT  JOBDESC_PTR=L_JDPTR RESULT_PTR=L_RESPTR;

 $H_LSJDERR ERROR_PTR=ADDR(L_RES_NB_ERROR);

/*****  Test of $;H_RUN  *****************************************/

 L_RES_NB_ERROR = 0; /* Reinitialize the result structure */

 L_JD_MEMBERS = " ";
 L_JD_FILE = "LIBRARY..GCL_JOB";
 L_JD_JOBLANG="G";
 L_JD_VALUES_LENGTH = 0;   /* no value */
 L_JD_VALUES_STRING = " ";

 $H_RUN  JOBDESC_PTR=L_JDPTR RESULT_PTR=L_RESPTR;

 $H_LSJDERR ERROR_PTR=ADDR(L_RES_NB_ERROR);
END;

END EJR_RUN_GPL;
```

7.10.7.2 Job Submission through COBOL Interface H_IN_URUN

Submit for the user OPERATOR, on the local site, the job GCL_JOB from the library LIBRARY. The option DEBUG_GCL is enabled. The global list of the job is requested and the GCL job output is stored in the member RGCL_JOB of LISLIB. No value is passed to the job. The RON of the job is edited if no errors occur. The procedure H_IN_UJDERR will edit the error messages equivalent to the error classes and numbers (see EJRRUNOUT-COB).

```
IDENTIFICATION DIVISION.
PROGRAM-ID. URUN-COB.

DATA DIVISION.
WORKING-STORAGE SECTION.

COPY JOBDESC-COB.
COPY EJRRUNOUT-COB.

77  LIST-ALL-ERR COMP-2 VALUE 0.

PROCEDURE DIVISION .
DEBUT.
  MOVE SPACE TO JD-JOB-DESCRIPTION.
  MOVE "LIBRARY..GCL_JOB" TO JD-FILE.
  MOVE "1" TO JD-HOLDOUT.
  MOVE "y" TO JD-JOR.
  MOVE "A" TO JD-LIST.
  MOVE "010010001" TO JD-SWITCHES.
  MOVE "0" TO JD-COMMIT.
  MOVE "OPERATOR" TO JD-USER.
  MOVE "1" TO JD-DEBUG-GCL.
  MOVE "LISLIB..RGCL_JOB" TO JD-PRTFILE.

  MOVE 0 TO JD-VALUES-LENGTH.

  MOVE ZERO  TO RES-RESULT.
  MOVE 180 TO RES-SEG-SIZE.

  CALL "H_IN_URUN" USING JD-JOB-DESCRIPTION RES-RESULT.

  IF RES-RETCODE = ZERO
     DISPLAY "Job introduced ==> RON:" RES-REQID-RON(1) UPON TERMINAL
  ELSE DISPLAY "No job submitted" UPON TERMINAL.

  CALL "H_IN_UJDERR" USING RES-ERROR-DESC LIST-ALL-ERR.
  STOP RUN.
```

7.10.7.3 Job Submission through COBOL Interface H_IN_UEJR

Submit on the remote site RMOT the JCL_JOB11 and JCL_JOB12 selected from the members JCL_JOB1 and JCL_JOB2 of the library LIBRARY, for the user JOB_OWNER. All jobs are submitted on the class C with the same values and their outputs will be held. The request index is returned if no errors occur. Call the procedure H_IN_UJDERR to edit the error messages equivalent to the error classes and numbers.

```
JCL_JOB1:    $JOB JCL_JOB11;
             $ENDJOB;
             $JOB JCL_JOB12;
             $ENDJOB;
             $JOB JCL_JOB13;
             $ENDJOB;
JCL_JOB2:    $JOB JCL_JOB2;
             $ENDJOB;
IDENTIFICATION DIVISION.
PROGRAM-ID. UEJR-COB.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY JOBDESC-COB.
COPY EJRRUNOUT-COB.
77  LIST-ALL-ERR COMP-2 VALUE 0.
PROCEDURE DIVISION .
DEBUT.
  MOVE SPACE TO JD-JOB-DESCRIPTION.
  MOVE "LIBRARY" TO JD-FILE.
  MOVE "JCL_JOB*" TO JD-MEMBERS.
  MOVE "1" TO JD-SELECTION-FLAG.
  MOVE "JCL_JOB12" TO JD-JOB-ID2.
  MOVE "1" TO JD-HOLDOUT.
  MOVE "C" TO JD-JOBCLASS.
  MOVE "RMOT" TO JD-HOST.
  MOVE "JOB_OWNER" TO JD-USER.
  MOVE "DUMMY" TO JD-PRTFILE.
  MOVE 41 TO JD-VALUES-LENGTH.
  MOVE "test of H_IN_UEJR entry PROCNAME=UEJR-COB" TO JD-VALUES-STRING.
  MOVE ZERO  TO RES-RESULT.
  MOVE 180 TO RES-SEG-SIZE.
  CALL "H_IN_UEJR" USING JD-JOB-DESCRIPTION RES-RESULT.
  IF RES-RETCODE = ZERO
     DISPLAY "Request introduced ==> JON:" RES-REQID-RON(1) UPON TERMINAL
  ELSE DISPLAY "No job submitted" UPON TERMINAL.
  CALL "H_IN_UJDERR" USING RES-ERROR-DESC LIST-ALL-ERR.
  STOP RUN.
```

7.10.8 Error Messages

Severity	Class	Number	Message
1	00	10000	ERROR TABLE OVERFLOW: TOO MANY ERRORS
1	00	10041	JOBLANG=GCL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10201	DEBUG_GCL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10211	TRACE_GCL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10221	SEV IS IGNORED WHEN HOST IS SPECIFIED
1	00	10241	REPEAT_STEP IS IGNORED WHEN HOST IS SPECIFIED
1	00	10251	JOURNAL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10261	DUMP IS IGNORED WHEN HOST IS SPECIFIED
1	00	10271	PRIVATE_DUMP IS IGNORED WHEN HOST IS SPECIFIED
1	00	10281	PRTFILE IS IGNORED WHEN HOST IS SPECIFIED
1	00	10042	JOBLANG=RTL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10202	DEBUG_GCL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10212	TRACE_GCL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10222	SEV IS IGNORED WHEN HOST IS SPECIFIED
1	00	10242	REPEAT_STEP IS IGNORED WHEN HOST IS SPECIFIED
1	00	10252	JOURNAL IS IGNORED WHEN HOST IS SPECIFIED
1	00	10262	DUMP IS IGNORED WHEN HOST IS SPECIFIED
1	00	10272	PRIVATE_DUMP IS IGNORED WHEN HOST IS SPECIFIED
1	00	10282	PRTFILE IS IGNORED WHEN HOST IS SPECIFIED

Severity	Class	Number	Message
3	01	00020	FILE VALUE IS NOT A FILE LITERAL
3	01	00030	SELECTION_FLAG VALUE MUST BE /0/1
3	01	00040	JOBLANG VALUE MUST BE G/J/
3	01	00050	HOLDOUT VALUE MUST BE /0/1
3	01	00060	HOLD VALUE MUST BE /0/1
3	01	00070	JOR VALUE MUST BE /Y/A/N
3	01	00080	LIST VALUE MUST BE /S/N/A
3	01	00090	CLASS VALUE MUST BE IN [A-P AA-PZ]
3	01	00100	PRIORITY VALUE MUST BE IN { ,0 TO 7}
3	01	00110	STARTUP VALUE MUST BE /1/0
3	01	00120	REPEAT VALUE MUST BE /0/1
3	01	00130	DELETE VALUE MUST BE /N/Y/F
3	01	00150	PASS VALUE MUST BE /0/1
3	01	00160	SWITCHES VALUE MUST BE /0/1
3	01	00170	COMMIT VALUE MUST BE /0/1
3	01	00180	PASSWORD VALUE MUST BE SET TO BLANK
3	01	00190	BANNER VALUE MUST BE /1/0
3	01	00200	DEBUG_GCL VALUE MUST BE /0/1
3	01	00210	TRACE_GCL VALUE MUST BE /0/1
3	01	00220	SEV VALUE MUST BE IN { ,1 TO 5}
3	01	00230	PCF VALUE MUST BE /0/1
3	01	00240	REPEAT_STEP VALUE MUST BE /0/1
3	01	00250	JOURNAL VALUE MUST BE /N/B/A/O
3	01	00260	DUMP VALUE MUST BE /N/D/A
3	01	00270	PRIVATE DUMP VALUE MUST BE /0/1
3	01	00280	PRTFILE VALUE IS NOT A FILE LITERAL
3	01	00290	ILLEGAL VALUE STRING
3	01	00300	UNKNOWN DEST: HOST NOT ATTACHED TO WORKING PROJECT IN CATALOG
3	01	00310	UNKNOWN DEST: STATION NOT ATTACHED TO WORKING PROJECT IN CATALOG
3	01	00320	EXPVAL VALUE MUST BE /0/1
3	01	00330	NOMESSIOF VALUE MUST BE /N/Y
3	02	00021	FILE MUST NOT BE BLANK CHARACTERS
3	02	00022	SUBFILE AND MEMBER ARE MUTUALLY EXCLUSIVE
3	02	00131	DELETE ASKED FOR A NOT SPECIFIED MEMBER
3	02	00141	RUN AND HOST ARE MUTUALLY EXCLUSIVE

Severity	Class	Number	Message
3	02	00142	RUN AND REMOTE FILE ARE MUTUALLY EXCLUSIVE
3	02	00143	JOB SUBMITTED ON SITE A,FILE ON B,HOST ON C IS NOT ALLOWED
3	02	00181	PROTECTED SYSTEM: USER MODIFICATION NOT ALLOWED
3	02	00191	BANINF AND BANNER=0 ARE MUTUALLY EXCLUSIVE
3	02	00291	ILLEGAL POS/KWD VALUE LENGTH
3	02	00292	VALUE STRING TOO LONG
3	02	00293	ILLEGAL KEYWORD NAME
3	02	00294	QUOTATION MARKS DO NOT BALANCE
3	02	00295	ILLEGAL KEYWORD VALUE
3	02	00296	KEYWORD NAME APPEARS TWO OR MORE TIMES
3	02	00321	KEYWORD NAME APPEARS TWO OR MORE TIMES
3	03	10000	UNABLE TO GET DATE AND TIME
3	03	10110	UNABLE TO GET A REQUEST INDEX
3	03	10120	UNABLE TO GET THE USER IDENTIFICATION FROM THE CATALOG
3	03	10130	UNABLE TO CHECK THE SECONDARY USER FROM THE CATALOG
3	03	10140	UNABLE TO CREATE JOBSET REQUEST
3	03	10150	UNABLE TO ACCESS TO NEW OPTION SEGMENT
3	03	10160	INPUT COMMAND VALUE IS NOT IN {65 TO 68,81 TO 84}
3	03	10170	RESULT DESCRIPTION: JOB ENTRY TABLE OVERFLOW
3	03	10180	RESULT DESCRIPTION: ERROR NUMBER IS NOT POSITIVE
3	03	10190	UNABLE TO CHECK THE USER: USER MANDATORY

❑

Index

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 GCL Programmer's Manual Job Control and IOF

Reference N° :	47 A2 36UJ 05	**Date:**	**August 1999**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME : ____________________ Date : __________

COMPANY : ____________________

ADDRESS : ____________________

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation D$^{ept.}$
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC		
357 AVENUE PATTON	**Phone:**	+33 (0) 2 41 73 72 66
B.P.20845	**FAX:**	+33 (0) 2 41 73 70 66
49008 ANGERS CEDEX 01	**E-Mail:**	srv.Duplicopy@bull.net
FRANCE		

CEDOC Reference #	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		

[_ _] : The latest revision will be provided if no revision number is given.

NAME: ______________________________ Date: __________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PHONE: ____________________ FAX: ____________________

E-MAIL: ______________________________

For Bull Subsidiaries:

Identification: ______________________________

For Bull Affiliated Customers:

Customer Code: ______________________________

For Bull Internal Customers:

Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 36UJ 05