NCR CONTROL DATA CORPORATION

ADVANCED SYSTEMS LABORATORY

# CHAPTER 03

# SYSTEM COMMAND LANGUAGE

( S C L )

Doc. No. ASL00282

Rev. 04

Copy No. 87

REV 75/05/27

TABLE OF CONTENTS

## 1.0 INTRODUCTION

Communication with the various components of IPL/OS is achieved with commands writted in a language called System Command Language (SCL). This document represents a definition of that language and a description of the manner in which commands written in it are processed by the system.

## 1.1 DESIGN OBJECTIVES

The effect of a command language statement should not be surprising, i.e., the names of commands and parameters should intuitively convey their meaning and there should not be any hidden side affects.

The command language should be invariant to the mode of access e.g. local batch, remote batch and interactive.

The command language should allow the user to conditionally process commands.

The command language should allow the user to transfer control to a command file.

The command language should allow the user to transfer control within a command file.

The command language should provide a facility which allows the user to define his own commands and to locally redefine system supplied commands.

User supplied commands should be indistinguishable from system supplied commands.

The command language should allow the user to specify the parameters for any command in a positionally independent as well as positionally dependent manner.

The command language should be insensitive to the manner in which the user chooses to specify values. As an example if an integer value is required, 511, 1FF(16) and 2**9-1 should be equally acceptable specifications. In general whenever a value is required, any expression resulting in the required value should be an acceptable specification.

## 1.2 TERMINOLOGY

SYSTEM COMMAND LANGUAGE (SCL) - the language with which the external user communicates with the various components of the operating system.

LOGICAL NAME SPACE (LNS) - the name management section of the operating system responsible for maintaining descriptions of the various objects managed by the system.

STANDARD INPUT FILE - the command file specified as input on the SUBMIT request. The LNS descriptor of this file appears at the bottom of the input control stack in the submitted job, and the FCB of this file is known to the submitted job as JOB#INPUT.

ALTERNATE INPUT FILE - any command file other than the STANDARD INPUT FILE. The LNS descriptor of this file appears in the input control stack, and the FCB of this file is known by a user supplied name.

PROFILE - a permanent command file whose name appears in the profile catalog. These files are processed automatically by the system on behalf of the user.

CURRENT INPUT FILE - the command file currently being processed by the SCL interpreter. The LNS descriptor of this file appears at the top of the input control stack, and the FCB of this file is known by a user supplied name.

SCL INPUT - the series of records read by the SCL interpreter on behalf of a given job. These records are obtained from either the standard input file or alternate input files.

BASE FILE - the default working file for a given job.

(more to be supplied)

## 1.3 STANDARD SYSTEM NAMES

JOB#JCB - name by which a job knows its own control block.

JOB#INPUT - name of the standard input file.

JOB#DISPLAY - name of the standard display file (interactive).

JOB#PRINT - name of the standard print file.

JOB#PUNCH - name of the standard punch file.

JOB#LGO - name of the standard load and go file.

JOB#DAYFILE - name of the job dayfile.

IOC#INPUT - name of the stream to which records read from JOB#INPUT are written. This stream is connected by the system to JOB#DAYFILE.

IOC#ALTERNATE - name of the stream to which records read from alternate input files are written. This stream is connected by the system to JOB#DAYFILE.

SCL#OUTPUT - name of the stream to which normal SCL output is written. This stream is connected by the system to JOB#DISPLAY for interactive jobs, and to JOB#PRINT for batch jobs.

SCL#DIAGNOSTIC - name of the stream to which SCL diagnostic messages are written. This stream is connected by the system to JOB#DISPLAY and JOB#DAYFILE for interactive jobs, and to JOB#PRINT and JOB#DAYFILE for batch jobs.

OCS#LOG - name of the stream to which job/operator messages are written. This stream is connected by the system to JOB#DAYFILE.

JOB#CLOCK_LIMIT - name of the event caused by Job Management when the clock limit is reached. The standard series of commands associated with this event are as follows:

WHEN JOB#CLOCK_LIMIT

```
    DISPLAY '*** CLOCK LIMIT ***'
    TERMINATE JOB#JCB
  WHENEND
```

JOB#TIME_LIMIT - name of the event caused by Job Management when the appropriate limit is reached. The standard series of commands associated with this event are as follows:

```
WHEN JOB#TIME_LIMIT
  DISPLAY '*** TIME LIMIT ***'
  TERMINATE JOB#JCB
WHENEND
```

JOB#COMPUTE_LIMIT - name of the event caused by Job Management when the appropriate limit is reached. The standard series of commands associated with this event are as follows:

```
WHEN JOB#COMPUTE_LIMIT
  DISPLAY '*** COMPUTE LIMIT ***'
  TERMINATE JOB#JCB
WHENEND
```

JOB#PRINT_LIMIT - name of the event caused by Job Management when the appropriate limit is reached. The standard series of commands associated with this event are as follows:

```
WHEN JOB#PRINT_LIMIT
  DISPLAY '*** PRINT LIMIT ***'
  TERMINATE JOB#JCB
WHENEND
```

JOB#PUNCH_LIMIT - name of the event caused by Job Management when the appropriate limit is reached. The standard series of commands associated with this event are as follows:

```
WHEN JOB#PUNCH_LIMIT
  DISPLAY '*** PUNCH LIMIT ***'
  TERMINATE JOB#JCB
WHENEND
```

SCL#BREAK - name of the event caused when the break key is depressed. The standard series of commands associated with this event are as follows:

```
WHEN SCL#BREAK
  DISPLAY '*** BREAK ***'
  RETURN USER
WHENEND
```

SCL#PARAM - name by which command files reference their values in the absence of user supplied parameter names.

SCL#LANGUAGE - name of the default language for the job.

## 1.4 METALANGUAGE

The symbol ::= is read as "IS DEFINED TO BE".

Elements enclosed by < > are to be considered a single syntactic unit in relation to surrounding meta symbols.

Elements enclosed by [ ] are optional and are to be considered a single syntactic unit in relation to surrounding meta symbols.

Elements separated by ! are mutually exclusive, and the symbol is read as "OR".

Elements followed by ... can be repeated.

<...> will be used to indicate that an ellipsis (two or more periods) is required. In this case the ellipsis is part of the language.

## 1.5 COMMAND AND DATA SUBMISSION

Commands and data are submitted to the system through either interactive or batch terminals. When an interactive terminal is used the commands and data are read directly by the SCL interpreter. When a batch terminal is employed the commands and data are read by the system input stager and placed on mass storage for subsequent processing by the SCL interpreter. The interactive terminal or staged mass storage file is called the standard input file, and is the file specified on the submit request by the System Access Manager (see Fig 1-1).

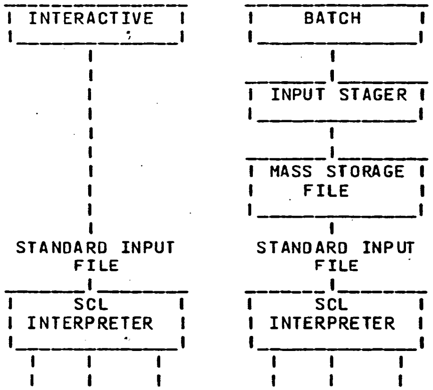

Fig 1-1

The SCL interpreter can be instructed to read its commands and data from other than the standard input file. When this is done the system "remembers" its previous input disposition in a table called the input control stack. The file appearing at the top of the input cont~ol stack is called the current input file (see Fig 1-2).

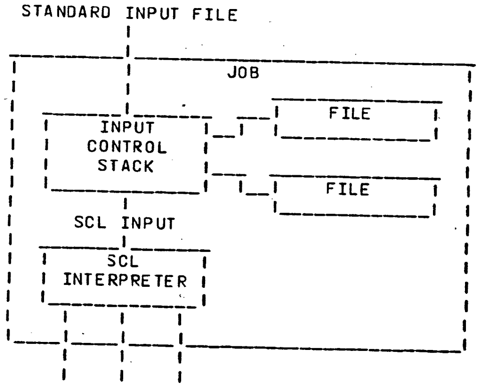

Fig 1-2

Command lines are read and interpreted "on the fly" by the SCL interpreter whereas data lines are read and deposited in a file without interpretation.

Command lines can be continued by placing an ellipsis at the end of the line. In this case the first character of the continuation line replaces the first character of the ellipsis. The total number of characters must not exceed 255.

```
Example: display 'This is a long character string cons...
         tant that is continued from one line to another'
```

## 1.6 WORKING ENVIRONMENT

When a user establishes contact with the system an environment, called his working environment, is created for him. The characteristics of this environment are determined by a series of profiles which are processed by the system on behalf of the newcomer. This mechanism allows the system to create an environment for a particular user which is uniquely suited to the needs of that user (see Fig 1-3).

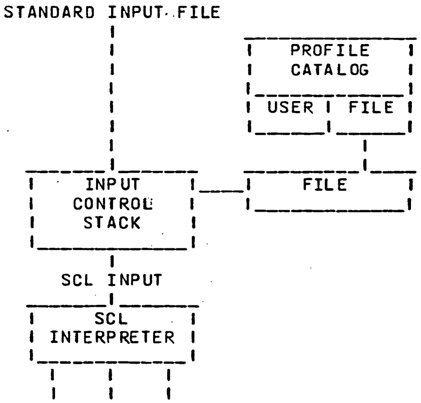

Fig 1-3

1.0 INTRODUCTION
1.6 WORKING ENVIRONMENT

The definition of the working environment for a given user is maintained in the Logical Name Space (LNS) of that user. The definition of the LNS for a given user is maintained in a table called the LNS segment list. This list containes the descriptors of the segments comprising the LNS and the order in which those segments are to be searched when resolving user references (see Fig 1-4).

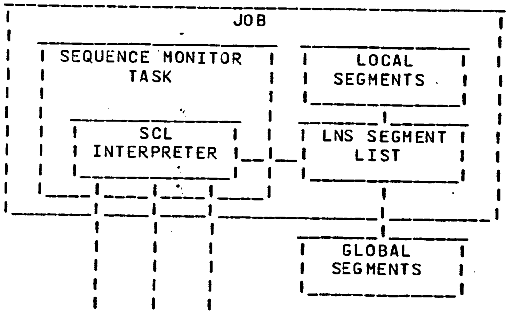

Fig 1-4

1.0 INTRODUCTION
1.6 WORKING ENVIRONMENT

The "flavor" of command language a given user will enjoy is determined by the set of command descriptors contained in his LNS. The user can add and/or delete segments of command descriptors to achieve the desired repertoire (see Fig 1-5).

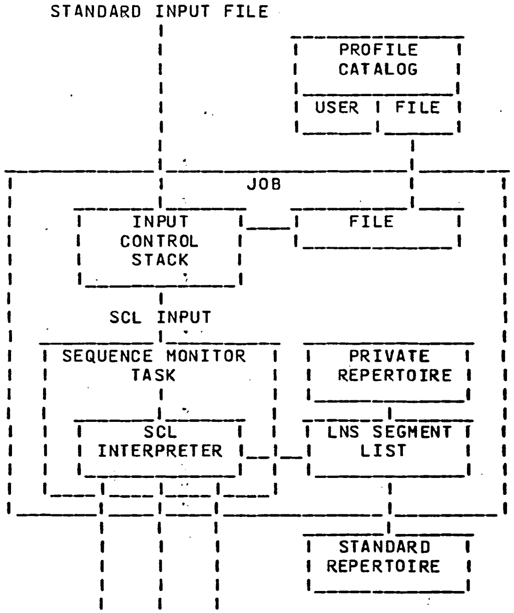

Fig 1-5

## 2.0 BASIC CONCEPTS

An understanding of the concepts presented in this section is necessary for the effective use of the System Command Language.

## 2.1 CONSTANTS

```
<constant> ::= <integer> | <real> | <string>
```

A constant is a data item that denotes itself, i.e., its representation is both its name and its value. A constant does more than state a value; it demonstrates various characteristics of the data item. For example 3.141593 shows that the constant is a real number as opposed to 3 which is an integer. SCL recognizes three types of constants: integer, real and string.

### 2.1.1 INTEGER

```
<integer> ::= <digit> [<hex digit>...]  [(<base>)]
```

Integer constants can be expressed as binary, octal, decimal or hexadecimal representations of an integer value. When the base specification is omitted 10 (decimal) is assumed. It should be noted that hexadecimal representations must still begin with a decimal digit and therefore a leading zero may be required.

Example: 123, 0ff(16)

### 2.1.2 REAL

```
<real> ::= <digit>... .<digit>... [E < + | - > <digit>...]
```

Real constants are always expressed as a decimal representation of a real value. The decimal point must be internal, i.e., surrounded by decimal digits.

Example: 123.456, 1.2E-12

### 2.1.3 STRING

<string> ::= '[ascii character]...'

A string constant is a sequence of characters that is treated as a single data item. The length of a string is the number of characters it contains. The maximum length of a string constant is 255. Any character may appear in a string, however, one internal apostrophe must be stated in the representation as two consecutive apostrophes.

Example: 'abc', 'abc''def'

### 2.2 IDENTIFIERS

<ident> ::= <alpha> [<alpha> | <digit>]...

An identifier is a string of alphanumeric characters not contained in a comment or constant. An identifier must be preceded and followed by a delimiter. The initial character of an identifier must not be a digit and the number of characters in an identifier must not exceed 31.

Example: x, login, task_1

## 2.3 REFERENCE NAMES

```
<ref>  ::=  [<ident>->]  <ident>  [(<expr>)]  [.<ident>
[(<expr>)] ]...
```

Reference names are used to identify various objects in the system. The simplest form of a reference name is the identifier and is used to identify labels and parameters. More complex reference names employ some combination of scope qualification, structure qualification and subscripting.

Scope qualification is indicated by an identifier followed by a right arrow digraph. The identifier is interpreted as the name of the LNS segment containing the object. For example global->x denotes the object called x contained within the LNS segment called global.

Structure qualification is indicated by an identifier (or subscript specification) followed by a period and another identifier. For example x.y denotes the object y contained within the structure x.

Subscripting is indicated by an identifier followed by a subscript. A subscript consists of an expression enclosed in parentheses and denotes the element of the array to be referenced. For example x(2) is interpreted to mean the second element of the array x.

The three structuring rules defined above can be combined to form arbitrarly complex reference names. For example global->x(i+4).y is a reference to the y field of the i+4th element of the global array x.

Example: LNS#GLOBAL->F1.OWNER

## 2.4 PARAMETER LISTS

Most commands require parameters to control their operation. These parameters are supplied by the user in a parameter list following the command name.

```
<command name> [ < ƀ ¦ , > <param list> ]
```

A parameter list consists of a series of parameters separated by spaces or commas. The omission of a parameter can be indicated by two consecutive commas, however, this is not necessary when the parameters are specified by name.

```
<param> [ < ƀ ¦ , > [<param>] ]...
```

Each parameter in the list can be specified in one of three formats. The first format is used to select options e.g. old/new and consists simply of the parameter name. The second and most common format consists of the parameter name followed by a value list. Both of these formats are positionally independent, i.e., the order in which they are quoted is unimportant. The third and only positionally dependent format consists simply of a value list.

```
  <param name>

¦ <param name> < ƀ ¦ = > <value list>

¦ <value list>
```

A value list consists of a series of values separated by commas and enclosed in parentheses. When a single value is quoted the parentheses can be omitted.

```
<value> ¦ (<value> [,<value>]...)
```

A value consists of a single expression representing the value or two expressions separated by an ellipsis representing a range of values.

```
<expr> [<...> <expr>]
```

Example: file f1 lines (1..23, 45)

NOTE: The positional significance of a parameter is one greater than the previous parameter specified in the list.

## 2.5 COMMENTS

<comment> ::= "[ascii character]..."

Comments are not interpreted by the command language interpreter and serve only as documentation. A comment acts syntactically the same as a space, i.e., whenever a space is allowed a comment is allowed, and whenever a space is required as a delimiter a comment will serve the same purpose.

Example: ..login wjh " in the beginning "

## 2.6 STREAMS

Streams are logical data sinks to which various parts of the system direct information. Files can be connected to streams in which case information directed to the stream will be written on the file. More than one file can be connected to a stream in which case information directed to the stream is written on all files connected to it. In addition a file can be connected to more than one stream in which case information directed to any of the streams to which the file is connected will be written on the file.

## 2.7 BLOCKS

The SCL recognizes two types of blocks - LNS blocks and LABEL blocks. These blocks localize the naming context for the user.

During job initiation an LNS block is established by the system for the job. This block forms the job local name space and in the normal case will contain most user defined names.

If the job is a batch job the system will also form a job local label space by establishing a label block for the job. In the normal case this block will contain the labels defined in the standard input file.

## 3.0 USER REPERTOIRE

The use of high level programming languages has greatly reduced the amount of detail the programmer must specify to implement a given algorithm. These languages reduce the amount of detail required by providing a logical environment in which the user operates and then mapping that logical environment onto the physical resources of the computer system. The same benifit can be realized in the area of job control through the use of a high level command language.

While it is true that the normal user can express his problem in a shorter amount of time and in a more reliable fashion through the use of a high level language it is also true that not all problems can be expressed effectively in a high level language. This is because the amount of control the user has over the physical resources of the system is reduced by automating the mapping of his logical requirements onto the physical resources of the system. This problem is typically solved by providing a machine language escape of some sort in the case of programming languages and can be solved by providing an operating system request escape in the case of a command language.

The approach outlined above is the one that has been adopted for the IPL System Command Language. Three logical environments are defined, one for the normal user, one for the system user and one for the system operator. These environments are controlled by three repertoires called the user, system and operator repertoires respectively.

One important aspect of the System Command Language defined for IPL is its extendability. Through the use of command files and procedures the user can augment the standard repertoires with commands suited to his particular requirements. These extensions are on a user by user basis and therefore no interference between specialized extensions will be observed.

The repertoire defined in this section is intended to allow the normal user to state his requirements in a simple and natural manner. When more control over the system is required the repertoire defined in the next section must be

used.

Several commands defined in this section appear in more than one place. These commands are applicable to more than one type of object and produce more than one type of result. They are documented in each area for which they have meaning.

## 3.1 SYSTEM ACCESS

### 3.1.1 LOGIN | LOGON | JOB

The purpose of this command is to gain access to the system. The command format is as follows:

```
login user=<ident>
```

user | u: This parameter identifies the user responsible for the processing. The identifier specified will be looked up in the valid user catalog.

When the user enters the system from an interactive terminal the system will output the following:

```
  SCL VER 1.0  2:30 PM  MONDAY  JUNE 30, 1979
  PLEASE LOGIN
..
```

When the user enters tne system from a batch terminal the solicitation for input will be suppressed.

```
Example:   SCL VER 1.0  2:30 PM  MONDAY  JUNE 30, 1979
           PLEASE LOGIN
         ..login wjh
           PROFILE PROCESSING INITIATED
         ..
```

### 3.1.2 LOCK

The LOCK command can be used to lock the terminal to prevent inadvertent logout. The command format is as follows:

```
lock terminal
```

terminal | t: This is a keyword parameter and is required to distinguish this variation of the LOCK command.

```
Example:   SCL VER 1.0  2:30 PM  MONDAY  JUNE 30, 1979
           PLEASE LOGIN
         ..login wjh
           PROFILE PROCESSING INITIATED
         ..lock terminal
           .
            .
             .
         ..login ras
           TERMINAL LOCKED
           SUPPLY ID TO RESUME
         ..
```

### 3.1.3 LOGOUT | LOGOFF | BYE | JOBEND | ENDJOB

The purpose of this command is to relinquish access to the system. The command format is as follows:

```
logout [user=<ident>]
```

user | u: This parameter is only required wnen the user has locked his terminal. In this case the identifier specified must match the identifier specified on the LOGIN command.

While a user is logged into the system he may issue a LOGIN without first issuing a LOGOUT.

```
Example:   SCL VER 1.0  2:30 PM  MONDAY  JUNE 30, 1979
           PLEASE LOGIN
         ..login wjh
           PROFILE PROCESSING INITIATED
           .
            .
             .
         ..login ras
           AUTOMATIC LOGOUT
           CONNECT TIME = HR:MN:SC
           PROFILE PROCESSING INITIATED
           .
            .
             .
         ..logout
           CONNECT TIME = HR:MN:SC
```

## 3.2 RESOURCE RESERVATION

### 3.2.1 CLAIM

The purpose of this command is to declare the maximum simultaneous usage of peripheral resources. The command format is as follows:

claim resource=(to be supplied) [status=<ref>]

resource ! r: This parameter specifies the resources to be claimed.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: (to be supplied)

### 3.2.2 RESERVE ! RES

The purpose of this command is to reserve peripheral units for exclusive use by the job. The command format is as follows:

reserve unitset=<ref list> [rej=<ref>] [status=<ref>]

unitset ! ucb ! u: This parameter specifies the names of the unit sets to be reserved.

reject ! rej ! r: This parameter specifies a variable into which the name of the unit set causing rejection of the command is to be returned.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: reserve (ucb1,ucb2,ucb3) rej=r

### 3.2.3 RELEASE ! REL

The purpose of this command is to release the units that were previously reserved by the job and return them to the pool of available units. The command format is as follows:

```
release unitset=<ref> [status=<ref>]
```

unitset ! ucb ! u: This parameter specifies the name of the unit set to be released.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: release ucb2

### 3.2.4 LIMIT

The purpose of this command is to limit the amount of resources a job can consume. The command format is as follows:

```
limit                      [clock=(hours,minutes,seconds)]
         [time=(hours,minutes,seconds)]
         [compute=(hours,minutes,seconds)]   [print=<expr>]
         [punch=<expr>] [status=<ref>]
```

clock ! c: This parameter specifies a clock time beyond which the job is not to be allowed to run. The event caused when this limit is reached is called JOB#CLOCK_LIMIT.

time ! t: This parameter specifies the amount of time the job is to be allowed to run. The event caused when this limit is reached is called JOB#TIME_LIMIT.

compute: This parameter specifies the amount of compute time the job is to be allowed. The event caused when this limit is reached is called JOB#COMPUTE_LIMIT.

print ! pr ! p: This parameter specifies the maximum number of lines the job is allowed to print. The event caused when this limit is reached is called JOB#PRINT_LIMIT.

punch ! pu: This parameter specifies the maximum number of cards the job is allowed to punch. The event caused when this limit is reached is JOB#PUNCH_LIMIT.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: limit compute=(0,5,0), print=2000, punch=500

## 3.3 LNS DECLARATIONS

Declarations can occur explicitly when the user directs the system to declare a name or implicitly when the user references a name in a parameter list which has not yet been declared.

### 3.3.1 DECLARE ! DCL

The purpose of this command is to declare a name. The command format is as follows:

```
declare    name=<ref   list>   [type=<ident>]   [len=<expr>]
          [dim=<expr>] [init=<expr list>] [status=<ref>]
```

name ! n: This parameter specifies the names to be declared.

type ! t: This parameter specifies the type of the name being declared. Omission of this parameter will cause the name to be declared as type integer.

length ! len ! l: This parameter is only meaningful when declaring type string. In this case it specifies the numoer of characters the string is to contain. Omission of this parameter will cause a default of 32 to be assumed.

dimension ! dim ! d: This parameter specifies the upper bounds of the array being declared. Omission of this parameter will cause a default of 1 to be assumed.

initialize ! init ! i: This parameter specifies the initial value the name being declared is to assume. The number of values quoted must be equal to the dimension specified.

(more to be supplied)

status: This parameter specifies a variable into which the command status is to be returned. Omission of

this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: declare f1, type=file

### 3.3.2 REMOVE

The purpose of this command is to remove a name. The command format is as follows:

```
remove name=<ref list> [status=<ref>]
```

name ! n: This parameter specifies the names to be removed.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: remove f1

### 3.3.3 LNSBLOCK / LNSEND

The purpose of these commands is to delimit an LNS block. An LNS block constitutes a local naming context, i.e., the default scope of LNS names declared by the user is local to the LNS block in which the declaration occurs. Such names can only be referenced from within that block or internally nested blocks and the life of such names is the life of the block in which they were declared. The general format of an LNS block is as follows:

```
lnsblock
  .
   .
    .
lnsend
```

Example: (to be supplied)

### 3.3.4 LNS

The purpose of this command is to cause segments to be added to and/or deleted from the LNS segment list. The command format is as follows:

```
lns [delete=<ref list>] [old=<ref list>] [new=<ref list>]
          [status=<ref>]
```

delete ! del ! d: This parameter specifies segments which are to be deleted from the LNS segment list. All deletions will be performed prior to any additions.

old ! o: This parameter specifies the segments which are to be added to the LNS segment list without initialization. All segments named in this parameter will be added prior to any segments named in the new parameter.

new ! n: This parameter specifies the segments which are to be added to the LNS segment list and initialized to empty.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

NOTE: The LNSBLOCK and LNS commands are similar in that they can both cause the attachment of a segment to the LNS segment list. They differ in that the LNSBLOCK command causes the attachment of a system supplied segment which is initialized to empty, whereas the LNS command causes the attachment of a user supplied segment with or without initialization.

LNS segments are searched in the reverse order from which they were added to the list, i.e. when a segment is added to the list it becomes the most local segment. Upon completion of an LNS command the current contents of the LNS segment list will be written on SCL#OUTPUT. It should be noted that an LNS command with no parameters will simply cause the current contents of the LNS segment list to be written on SCL#OUTPUT.

```
Example: ..lns
           LNS = LNS#GLOBAL, SCL#UREP, LNS#LOCAL
         ..lns del=lns#local, old=(privrep,lns#local)
           LNS = LNS#GLOBAL, SCL#UREP, PRIVREP, LNS#LOCAL
         ..
```

### 3.3.5 DISPLAY ! DI

The DISPLAY command can be used to display the LNS descriptor of a name. The result will be written on SCL#OUTPUT. The command format is as follows:

```
display desc=<ref>
```

descriptor ! desc ! d: This parameter specifies the name whose descriptor is to be displayed. The parameter name is required in this case to distinguish this variation of the display command.

```
Example: ..display desc f1
           TYPE = FILE
           DIM = 1
           SIZE = 128
           ATTR = 1, 35
           DATA ADDR = 1F007C0(16)
           DESC ADDR = 1F00500(16)
         ..
```

## 3.4 COMMAND DEFINITION

When commands are defined to the system, a command file or a procedure is associated with the command. If a command file is associated with a command that file will be processed by the system when the command is referenced. This mechanism provides a procedure file capability which is powerful enough for most user needs. In some cases, however, a procedure must be associated with a command. The details on how to write such a procedure can be found in the section titled COMMAND WRITERS GUIDE.

### 3.4.1 DEFINE ! DEF

The purpose of this command is to define a new command or to redefine a system supplied command. The command has two formats which are as follows:

```
define   command=<ref>  file=<ref>  [name=<ident   list>]
       [req=<expr  list>]  [fgn=<expr  list>]  [rng=<expr
       list>] [min=<expr list>] [max=<expr list>]
```

command ! c: This parameter specifies the name of the command to be defined.

file ! fcb ! f: This parameter specifies the name of the command file to be processed when the command is referenced.

name ! n: This parameter specifies the names of the parameters. The positional significance of the parameters will be determined by the order in which their names are specified left to right. Omission of this parameter will cause the system to assume a single parameter name of SCL#PARAM.

required ! req ! r: This parameter specifies the set of required parameters. Omission of this parameter will cause the interpreter to assume all parameters are optional.

foreign ! fgn: This parameter specifies the set of

foreign text parameters. Omission of this parameter will cause the interpreter to assume none of the parameters expect foreign text.

range ! rng: This parameter specifies the set of range parameters. Omission of this parameter will cause the interpreter to assume none of the parameters expect ranges of values.

minimum ! min ! m: This parameter specifies the minimum number of values required by each parameter. Omission of this parameter will cause the interpreter to assume all parameters require 1 value.

maximum ! max: This parameter specifies the maximum number of values allowed by each parameter. Omission of this parameter will cause the interpreter to assume each parameter allows only 1 value.

Within the command file parameters are referenced as follows:

<param name>.present

This format yields a 1 if the parameter name was present in the list and a 0 otherwise.

<param name>.count

This format yields a number corresponding to the number of values specified for the parameter.

<param name>(<value number>).low

This format yields the value specified for the low end of a range. If a range was not specified this format yields the value that was specified.

<param name>(<value number>).high

This format yields the value specified for the high end of a range. If a range was not specified this format yields the value that was specified.

```
Example: COLLECT CMD_FILE

           "COMPTESTS 10..14 WILL CAUSE THE SOURCE PROGRAMS"
           "CONTAINED IN FILES SOU10 THROUGH SOU14 TO BE"
           "COMPILED AND THEIR OBJECT MODULES TO BE PLACED"
           "IN FILES OBJ10 THROUGH OBJ14"

           "CHECK RANGE"
           IF  RANGE(1).LOW GE 0 ...
           AND RANGE(1).LOW LE RANGE(1).HIGH ...
           AND RANGE(1).HIGH LE 99
             LNSBLOCK " LOCAL VARIABLE BLOCK "
               DECLARE (I, J, K) TYPE=INTEGER
               DECLARE DIGIT, TYPE=STRING, LEN=1, DIM=10, ...
                 INIT=('0','1','2','3','4','5','6','7','8','9')
               DECLARE (S_FILE, O_FILE) TYPE=STRING, LEN=5
               DECLARE TEMP, TYPE=STRING, LEN=2

               FOR I, RANGE(1).LOW...RANGE(1).HIGH
                 "CREATE NEXT NAME IN SEQUENCE"
                 J = I / 10
                 K = I - J * 10
                 TEMP = DIGIT(J+1) CAT DIGIT(K+1)
                 S_FILE = 'SOU' CAT TEMP
                 O_FILE = 'OBJ' CAT TEMP
                 DISPLAY 'COMPILING ' CAT S_FILE
                 FORTRAN INPUT=REF(S_FILE), OBJECT=REF(O_FILE)
               FOREND
             LNSEND
             RETURN
           IFEND
           DISPLAY 'ERROR ... INVALID RANGE'
           **
           DEFINE COMPTESTS, FILE=CMD_FILE, NAME=RANGE
```

define command=<ref> proc=<ref>

command ! c: This parameter specifies the name of the command to be defined.

procedure ! proc ! p: This parameter specifies the name of the procedure to be executed when the command is referenced.

Example: DEFINE DECLARE, PROC=SCL#LNS#DECLARE

### 3.4.2 DISPLAY ! DI

The DISPLAY command can be used to display the parameters of a command (command file or procedure). The result will be written on SCL#OUTPUT. The command format is as follows:

display name=<ref>

name ! n: This parameter specifies the name of the command whose parameters are to be displayed.

```
Example: ..display declare
            POSN REQ FGN RNG MIN MAX NAME(S)

               1 YES  NO  NO   1   5 NAME, N
               2  NO  NO  NO   1   1 TYPE, T
               3  NO  NO  NO   1   1 LENGTH, LEN, L
               4  NO  NO  NO   1   1 DIMENSION, DIM, D
               5  NO  NO  NO   1   5 INITIALIZE, INIT, I
               6  NO  NO  NO   1   1 STATUS, S
         ..
```

## 3.5 FILE MANAGEMENT

### 3.5.1 ATTACH

The purpose of this command is to attach permanent files and volume sets to a job. When files are attached the system will establish the requestors legal access to the file. Evaluations of the file access control list, and the user id establish the type of access granted to the file. Once access legality has been established, the permanent file is attached to the job and may be used as specified by the ATTACH command. When volume sets are attached to a job they can be attached for either shared or exclusive use. Successful completion of the command in this case results in volumes being mounted, volume labels being validated, and optional temporary assignment of units to the job. The command format is as follows:

```
attach      name=<ref>      [usage=<ident>]     [unitset=<ref>]
            [status=<ref>]
```

name ! n: This parameter specifies the name of the file or volume set to be attached.

usage ! use ! u: This parameter is only meaningful when attaching files. It specifies the intended use of the file. The valid specifications for this parameter are as follows:

ER - exclusive read or execute access
EW - exclusive write access
EP - exclusive private write access
SR - shared read or execute access
SW - shared write access
SP - shared private write access

"Exclusive" access means that multiple exclusive readers or one exclusive writer may have the file attached simultaneously. "Shared" access means that multiple shared readers and writers may have the file attached simultaneously. "Private write" means the user can write on the file but the

changes are not written back to the master file. All changed blocks of the file are kept local to the user. See the Data Management section on concurrent access control for additional information.

unitset ! ucb: This parameter is only meaningful when attaching volume sets. It specifies the name of the unit set describing the units on which the volumes are to be mounted. Omission of this parameter is only allowed for mass storage volume sets and implies that the volume set is being attached for shared use with other jobs. An internal table is referenced to determine if the set has already been attached for shared use by another job. If it has, then the current job is linked directly to it. If a previous job has attached the volume set for exclusive use, then the command is rejected. If the volume set is not currently attached to any job and is not currently on-line, units are temporarily assigned, and the volumes are mounted.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: attach f1, use=sr

### 3.5.2 DETACH

The purpose of this command is to sever the connection between a job and a file or volume set. If a temporary file is being detached, it is deleted from the system and the resources allocated to it are released. If a permanent file is being detached, it is no longer accessible to the job but the definition and contents of the file still exist. If a volume set is being detached, the units to which the set was attached are not released if the units were reserved by the current job. If the units were temporarily assigned because of attaching the volume set for shared use, and if no other jobs are currently attached to the set, then the units are returned to the available pool. The command format is as follows:

```
detach name=<ref> [status=<ref>]
```

name ! n: This parameter specifies the name of the file or volume set to be detached.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: detach f1

### 3.5.3 CREATE

The purpose of this command is to define a new file to the file management system. All parameters that are required to describe the file are located in the file control block. System default values for the parameters may be supplied if the user has not explicitly supplied a value. Resources required by the file are allocated to the file at this time. If the file is defined to be permanent, an entry is made in the appropriate file catalog. The command format is as follows:

```
create file=<ref> [status=<ref>]
```

file ! fcb ! f: This parameter specifies the name of the file to be created.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: create f1

### 3.5.4 DELETE ! DEL

The purpose of this command is to remove the definition of a permanent file from the file catalog. If the file is not currently attached to a job, the file label in the catalog is deleted and the resources allocated to the file are released. If the file is currently attached to a job, the status of the file is changed to a temporary file. The command format is as follows:

```
delete file=<ref> [status=<ref>]
```

file ! fcb ! f: This parameter specifies the name of the file to be deleted.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: delete f1

### 3.5.5 SAVE

The purpose of this command is to convert a temporary file to a permanent file. Since all the space allocation and consistency checks on logical file characteristics were performed by previous CREATEF or EXPANDF requests, the major remaining effort is to construct a file label and enter it in a catalog. The file itself is not copied but rather remains where it was originally allocated. The command format is as follows:

```
save file=<ref> [status=<ref>]
```

file ! fcb ! f: This parameter specifies the name of the file to be saved.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: save f1

### 3.5.6 DISPLAY ! DI

The DISPLAY command can be used to display a file. The content of the file specified will be written on SCL#OUTPUT. The command format is as follows:

```
display [name=<ref>]
```

name ! n: This parameter specifies the name of the file to be displayed. Omission of this pa~ameter will cause the base file to be displayed.

```
Example: ..display f1
           This is the 1st line
           This is the 2nd line
           This is the 3rd line
          ..
```

### 3.5.7 PRINT ! PR

The PRINT command can be used to print a file. The content of the file specified will be written on JOB#PRINT. The command format is as follows:

```
print [name=<ref>] [form=<expr>] [copies=<expr>]
```

name ! n: This parameter specifies the name of the file to be printed. Omission of this parameter will cause the base file to be printed.

form ! f: This parameter specifies the form on which the file is to be printed. Omission of this parameter will cause the site default to be used.

copies ! c: This parameter specifies the number of copies to be printed. Omission of this parameter will cause 1 copy to be printed.

Example: print f1

### 3.5.8 PUNCH ! PU

The purpose of this command is to punch a file. The content of the file specified will be written on JOB#PUNCH. The command format is as follows:

```
punch [file=<ref>]
```

file ! fcb ! f: This parameter specifies the file to be punched. Omission of this parameter will cause the base file to be punched.

Example: punch f1

## 3.6 FILE ROUTING

### 3.6.1 ROUTE

The purpose of this command is to initiate the transmittal of a file to some destination. The command format is as follows:

```
route file=<ref> [dest=<ref>] [form=<expr>] [copies=<expr>]
[status=<ref>]
```

file ! fcb ! f: This parameter specifies the name of the file to be routed.

destination ! dest ! d: This parameter specifies the destination of the file. Omission of this parameter will cause the file to be routed to the printer.

The destination specified on a route or direct command becomes associated with a particular unit via the ONSYSTEM command described under operator communication.

form: This parameter specifies the physical properties of the output medium (e.g. paper size, card type, printer train, ribbon color) upon which the data is to be placed. Omission of this parameter will cause a default to be assumed.

copies ! c: This parameter specifies the number of copies to be placed on the output unit. Omission of this parameter will cause a default value to be assumed.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: route f1, dest=ses

### 3.6.2 DIRECT

The purpose of this command is to permit the specification of a destination for a file. This command differs from the previous command in that ROUTE initiates the transmittal of a specified file to a specified destination whereas DIRECT informs Job Management that upon job termination a specified file is to be transmitted to a specified destination. The command format is as follows:

```
direct file=<ref> [dest=<ref>] [form=<expr>] [copies=<expr>]
[status=<ref>]
```

file ! fcb ! f: This parameter specifies the name of the file to be directed.

destination ! dest ! d: This parameter specifies the destination of the file. Omission of this parameter will cause the file to be directed to the printer.

form: This parameter specifies the physical properties of the output medium (e.g. paper size, card type, printer train, ribbon color) upon which the data is to be placed. Omission of this parameter will cause a default to be assumed.

copies ! c: This parameter specifies the number of copies to be placed on the output unit. Omission of this parameter will cause a default value to be assumed.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: direct f1, dest=aslc_os

### 3.6.3 RETRACT

The purpose of this command is to countermand a previous DIRECT command. The command format is as follows:

```
retract file=<ref>
```

file ! fcb ! f: This parameter specifies the file which was previously directed.

Example: retract f1

## 3.7 DATA INPUT

### 3.7.1 COLLECT ! COL

The purpose of this command is to collect data from SCL INPUT. The command format is as follows:

```
collect [file=<ref>] [until=<expr>]
```

file ! fcb ! f: This parameter specifies the name of a file into which the data is to be placed. Omission of this parameter will cause the data to be placed in the base file.

until ! u: This parameter specifies a string containing an end of data delimiter. Omission of this parameter will cause a default of ** to be assumed.

```
Example: ..collect f1 until '/*'
           SUPPLY RECORDS
         ..This is the 1st line
         ..This is the 2nd line
         ..This is the 3rd line
         ../*
           RECORDS COLLECTED
         ..
```

## 3.8 STREAM MANAGEMENT

The figures shown below reflect the standard stream connections established by the system for a job. The user can, of course, alter the connections at any time.

```
                          INTERACTIVE JOB

                         __________________
JOB#INPUT ________>|                  |__> IOC#INPUT _________
                   | SCL INTERPRETER  |                       |
ALTERNATE INPUT __>|                  |__> IOC#ALTERNATE ___  |
                   |__________________|                     | |
                      /       |       \                     | |
                     /        |        \                    | |
                    |         |         |                   | |
           SCL#OUTPUT         |         |                   | |
                    |  SCL#DIAGNOSTIC   |                   | |
                    |         |      OCS#LOG                | |
                    |         |         |                   | |
JOB#DISPLAY <_______V_________V         |                   | |
                              |         |                   | |
JOB#PRINT                     |         |                   | |
                              |         |                   | |
JOB#DAYFILE <_________________V_________V___________________V_V
```

Fig 3-1

```
                          BATCH JOB

                         __________________
JOB#INPUT ________>|                  |__> IOC#INPUT _________
                   | SCL INTERPRETER  |                       |
ALTERNATE INPUT __>|                  |__> IOC#ALTERNATE ___  |
                   |__________________|                     | |
                      /       |       \                     | |
                     /        |        \                    | |
                    |         |         |                   | |
           SCL#OUTPUT         |         |                   | |
                    |  SCL#DIAGNOSTIC   |                   | |
                    |         |      OCS#LOG                | |
                    |         |         |                   | |
JOB#PRINT <_________V_________V         |                   | |
                              |         |                   | |
JOB#DAYFILE <_________________V_________V___________________V_V
```

Fig 3-2

The connections shown in Fig 3-1 are established by a profile containing the commands shown below.

```
DECLARE IOC#INPUT, TYPE=STREAM
CONNECT IOC#INPUT, JOB#DAYFILE
DECLARE IOC#ALTERNATE, TYPE=STREAM
CONNECT IOC#ALTERNATE, JOB#DAYFILE
DECLARE SCL#OUTPUT, TYPE=STREAM
CONNECT SCL#OUTPUT, JOB#DISPLAY
CONNECT SCL#OUTPUT, JOB#DAYFILE
DECLARE SCL#DIAGNOSTIC, TYPE=STREAM
CONNECT SCL#DIAGNOSTIC, JOB#DISPLAY
CONNECT SCL#DIAGNOSTIC, JOB#DAYFILE
DECLARE OCS#LOG, TYPE=STREAM
CONNECT OCS#LOG, JOB#DAYFILE
```

The connections shown in Fig 3-2 are established by a profile containing the commands shown below.

```
DECLARE IOC#INPUT, TYPE=STREAM
CONNECT IOC#INPUT, JOB#DAYFILE
DECLARE IOC#ALTERNATE, TYPE=STREAM
CONNECT IOC#ALTERNATE, JOB#DAYFILE
DECLARE SCL#OUTPUT, TYPE=STREAM
CONNECT SCL#OUTPUT, JOB#PRINT
CONNECT SCL#OUTPUT, JOB#DAYFILE
DECLARE SCL#DIAGNOSTIC, TYPE=STREAM
CONNECT SCL#DIAGNOSTIC, JOB#PRINT
CONNECT SCL#DIAGNOSTIC, JOB#DAYFILE
DECLARE OCS#LOG, TYPE=STREAM
CONNECT OCS#LOG, JOB#DAYFILE
```

### 3.8.1 CONNECT ! CON

The purpose of this command is to establish a connection between a stream and a file. The command format is as follows:

connect stream=<ref> file=<ref> status=<ref>

stream ! scb ! s: This parameter specifies the stream to be connected.

file ! fcb ! f: This parameter specifies the file to be connected. If the file specified is not currently connected to any streams the file will be connected and opened for output.

status: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: ..connect stream=scl#alternate, file=job#display

### 3.8.2 DISCONNECT ! DISCON

The purpose of this command is to sever the connection between a stream and a file. The command format is as follows:

disconnect stream=<ref> file=<ref> status=<ref>

stream ! scb ! s: This parameter specifies the stream to be disconnected.

file ! fcb ! f: This parameter specifies the file to be disconnected. If the file specified is not currently connected to any other streams the file will be disconnected and closed.

status: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

### 3.8.3 DISPLAY ! DI

The DISPLAY command can be used to display the current status of a specified stream. The result will be written on SCL#OUTPUT. The command format is as follows:

display status=<ref>

status ! s: This parameter specifies the name of the stream whose status is to be displayed.

```
Example: ..display status scl#output
           TYPE = STREAM
           CONNECTIONS = JOB#DISPLAY, JOB#DAYFILE
         ..
```

## 3.9 PROGRAM COMPILATION

Although the standard language translators can be executed as normal tasks, because of their common use, specialized commands are supplied to facilitate their invocation.

### 3.9.1 FORTRAN ! FORT ! FTN

The purpose of this command is to invoke the Fortran compiler. The command format is as follows:

```
fortran    [input=<ref>]      [list=<ref>]      [object=<ref>]
        [opt=<ident list>] [status=<ref>]
```

input ! inp ! i: This parameter specifies the name of the file containing the source program to be compiled. Omission of this parameter will cause the base file to be compiled.

listing ! list ! l: This parameter specifies the name of the file on which the listing is to be written. Omission of this parameter will cause the listing to be written on JOB#PRINT.

object ! obj ! o: This parameter specifies the name of the file on which the object module is to be written. Omission of this parameter will cause the object module to be written on the JOB#LGO.

option ! opt: This parameter specifies a list of compiler options. The Fortran compiler provides the following options:

SL ! NSL - source list
AL ! NAL - assembly list
CR ! NCR - cross reference
MM ! NMM - memory map
OM ! NOM - object module
OP ! NOP - optimization
DB ! NDB - debug tables
SC ! NSC - syntax check only

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: ftn inp=f1, opt=(sl,cr,om)

### 3.9.2 COBOL ! COB

The purpose of this command is to invoke the Cobol compiler. The command format is as follows:

```
cobol [input=<ref>] [library=<ref>] [list=<ref>] [dgn=<ref>]
        [object=<ref>] [opt=<ident list>] [status=<ref>]
```

input ! inp ! i: This parameter specifies the name of the file containing the source program to be compiled. Omission of this parameter will cause the base file to be compiled.

library ! lib ! l: This parameter specifies the name of the source statement library to be used during compilation.

listing ! list: This parameter specifies the name of the file on which the listing is to be written. Omission of this parameter will cause the listing to be written on JOB#PRINT.

diagnostic ! dgn ! d: This parameter specifies the name of the file on which the diagnostic messages generated by the compilation are to be written. Omission of this parameter will cause the diagnostic messages to be written on the list file.

object ! obj ! o: This parameter specifies the name of the file on which the object module is to be written. Omission of this parameter will cause the object module to be written on the JOB#LGO.

option ! opt: This parameter specifies a list of compiler options. The Cobol compiler provides the following options:

SL ! NSL - source list
AL ! NAL - assembly list
CR ! NCR - cross reference
MM ! NMM - memory map
OM ! NOM - object module
FG ! NFG - force generation
DB ! NDB - debug tables
SC ! NSC - syntax check only
BC ! NBC - bounds check
PG ! NPG - propogated diagnostics
AV ! NAV - advisory diagnostics
TR ! NTR - trivial diagnostics
NS ! NNS - non standard usage diagnostics

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: cobol inp=f1, obj=f2

### 3.9.3 SWL

The purpose of this command is to invoke the SWL compiler. The command format is as follows:

```
swl  [input=<ref>]  [list=<ref>]  [object=<ref>] [opt=<ident
        list>] [status=<ref>]
```

input ! inp ! i: This parameter specifies the name of the file containing the source program to be compiled. Omission of this parameter will cause the base file to be compiled.

listing ! list ! l: This parameter specifies the name of the file on which the listing is to be written. Omission of this parameter will cause the listing to be written on JOB#PRINT.

object ! obj ! o: This parameter specifies the name of the file on which the object module is to be written. Omission of this parameter will cause the object module to be written on the JOB#LGO.

option ! opt: This parameter specifies a list of compiler options. The SWL compiler provides the following options:

(to be supplied)

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: swl inp=f1, listing=f2

### 3.9.4 COMPILE ! COMP

The purpose of this command is to compile a program. The compiler selected will be determined by examining the data type attribute in the file control block of the input. If the data type is undefined the selection will be determined by the contents of the job local variable called SCL#LANGUAGE. The command format is as follows:

```
compile    [input=<ref>]    [list=<ref>]     [object=<ref>]
        [opt=<ident list>] [status=<ref>]
```

input ! inp ! i: This parameter specifies the name of the file containing the source program to be compiled. Omission of this parameter will cause the base file to be compiled.

listing ! list ! l: This parameter specifies the name of the file on which the listing is to be written. Omission of this parameter will cause the listing to be written on JOB#PRINT.

object ! obj ! o: This parameter specifies the name of the file on which the object module is to be written. Omission of this parameter will cause the object module to be written on the JOB#LGO.

option ! opt: This parameter specifies a list of compiler options. Which options are meaningful will depend on which compiler is invoked. An attempt will be made to retain consistency between the compiler options where possible, i.e. if a given compiler provides an option for source listing generation the option will be specified by an S or NS in the option parameter.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: comp inp=f1, obj=f2

## 3.10 PROGRAM EXECUTION

### 3.10.1 LIBRARY ! LIB

The purpose of this command is to cause entries to be added to and/or deleted from the loader library list. The command format is as follows:

library [delete=<ref list>] [add=<ref list>] [status=<ref>]

delete ! del ! d: This parameter specifies the names of the libraries which are to be deleted from the library list. All deletions will be performed prior to any additions. Omission of this parameter will cause no deletions to be made.

add ! a: This parameter specifies the names of the libraries which are to be added to the library list. The libraries are searched in the order in which they are added to the list. Omission of this parameter will cause no additions to be made.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

NOTE: Upon completion of a LIBRARY command the current contents of the library list will be written on SCL#OUTPUT. It should be noted that a LIBRARY command with no parameters will simply cause the current contents of the library list to be written on SCL#OUTPUT.

```
Example: ..lib (lib1,lib2)
           LIB = SYS#LIB, LIB1, LIB2
           ..
```

### 3.10.2 OBJECT ! OBJ

The purpose of this command is to add and/or delete entries from the loader object list. The command format is as follows:

```
object [delete=<ref list>] [add=<ref list>] [status=<ref>]
```

delete ! del ! d: This parameter specifies the names of the files which are to be deleted from the object list. All deletions will be performed prior to any additions. Omission of this parameter will cause no deletions to be made.

add ! a: This parameter specifies the names of the files which are to be added to the object list. Omission of this parameter will cause no additions to be made.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

NOTE: Upon completion of the OBJECT command the current contents of the object list will be written on SCL#OUTPUT. It should be noted that an OBJECT command with no parameters will simply cause the current contents of the object list to be written on SCL#OUTPUT.

```
Example: ..obj d=test1, a=test2
            OBJ = JOB#LGO, TEST2
         ..
```

### 3.10.3 EXECUTE ! EXEC ! XEQ

The purpose of this command is to cause the named program to be loaded and executed as a task. For a description of the libraries and object files used by the loader see the LIBRARY and OBJECT command descriptions. The command format is as follows:

```
execute  program=<ident>  [param=<expr  list>]  [task=<ref>]
       [event=<ref>] [status=<ref>]
```

program ! prog ! pcb ! p: This parameter specifies the name of the program to be executed.

parameter ! param: This parameter specifies a list of values which are to be passed to the executed program. Omission of this parameter will cause a null value list to be passed.

task ! tcb ! t: This parameter specifies the name by which the new task is to be known. Omission of this parameter will cause the system to supply a name, however, since this name will not be known by the user no control over the processing of that task will be possible.

event ! ecb ! e: This parameter specifies the name of an event which is to be caused upon task termination. Omission of this parameter will cause the SCL interpreter to await the termination of the task before returning control to the user.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

```
Example: ..lib f1
           LIB = SYS#LIB, F1
         ..execute prog=main, param=(x,y), task=t1
         ..
```

### 3.10.4 STOP

The stop command can be used to stop the processing of a specified task. The command format is as follows:

```
stop name=<ref> [status=<ref>]
```

name | n: This parameter specifies the name of the task which is to be stopped. If the task specified is already stopped no operation will take place.

status | s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: stop t1

### 3.10.5 START

The start command can be used to restart the processing of a stopped task. The command format is as follows:

```
start name=<ref> [status=<ref>]
```

name | n: This parameter specifies the name of the task which is to be started. If the task specified is already in a started state no operation will take place.

status | s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: start t1

### 3.10.6 TERMINATE | TERM

The terminate command can be used to terminate processing of a specified task. The command format is as follows:

```
terminate name=<ref> [status=<ref>]
```

name | n: This parameter specifies the name of the task which is to be terminated.

status | s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: term t1

### 3.10.7 DISPLAY | DI

The DISPLAY command can be used to display the current status of a specified task. The result will be written on SCL#OUTPUT. The command format is as follows:

```
display status=<ref>
```

status | s: This parameter specifies the name of the task whose status is to be displayed.

```
Example: ..display status t1
           TYPE = TASK
           CONDITION = EXECUTING
         ..
```

## 3.11 JOB SUBMISSION

### 3.11.1 SUBMIT

The purpose of this command is to submit a new job to the system. The command format is as follows:

```
submit [input=<ref>]  [param=<value  list>]  [job=<ref>]
       [event=<ref>] [status=<ref>]
```

input ! inp ! i: This parameter specifies the name of the file which is to serve as the standard input for the submitted job. Omission of this parameter will cause the base file to be submitted.

parameter ! param ! p: This parameter specifies a list of values which are to be passed to the submitted job. Omission of this parameter will cause a null value list to be passed.

job ! jcb ! j: This parameter specifies the name by which the submitted job is to be known. Omission of the job parameter will cause the system to supply a name to the submitted job, however, since this name will not be known to the user no control over the processing of that job will be possible.

event ! ecb ! e: This parameter specifies the name of an event which is to be caused upon job termination. Omission of the event parameter will mean that no indication of the termination of the submitted job will be received by the submitting job.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: submit f1, job=j1

Within the submitted job parameters are referenced as follows:

SCL#PARAM.count

This format yields a number corresponding to the number of values passed to the job.

SCL#PARAM(<value number>)

This format yields the nth value specified where n is the value number quoted.

### 3.11.2 STOP

The stop command can be used to stop the processing of a specified job. The command format is as follows:

```
stop name=<ref> [status=<ref>]
```

name ! n: This parameter specifies the name of the job which is to be stopped. If the job specified is already stopped no operation will take place.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: stop j1

### 3.11.3 START

The start command can be used to restart the processing of a stopped job. The command format is as follows:

```
start name=<ref> [status=<ref>]
```

name ! n: This parameter specifies the name of the job which is to be started. If the job specified is already in a started state no operation will take place.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: start j1

### 3.11.4 TERMINATE ! TERM

The terminate command can be used to terminate processing of a specified job. The command format is as follows:

```
terminate name=<ref> [status=<ref>]
```

name ! n: This parameter specifies the name of the job which is to be terminated.

status ! s: This parameter specifies a variable into which the command status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

Example: term j1

### 3.11.5 DISPLAY ! DI

The DISPLAY command can be used to display the current status of a specified job. The result will be written on SCL#OUTPUT. The command format is as follows:

```
display status=<ref>
```

status ! s: This parameter specifies the name of the job whose status is to be displayed.

```
Example: ..display status j1
           TYPE = JOB
           CONDITION = TERMINATED
         ..
```

## 3.12 EVENT_HANDLING

### 3.12.1 TIMER

The purpose of this command is to inform the system timer that a specified event is to be caused upon satisfaction of a specified condition. The command format is as follows:

```
timer        event=<ref>         [clock=(hour,minute,second)]
        [time=(hour,minute,second)]
        [compute=(hour,minute,second)]
```

event | ecb | e: This parameter specifies the name of the event which is to be caused.

clock | c: This parameter specifies a clock time at which the specified event is to be caused. If the time specified has already passed the event will be caused immediately.

time | t: This parameter specifies an amount of real time after which the specified event is to be caused.

compute: This parameter specifies an amount of compute time spent on this job after which the specified event is to be caused.

Example: timer delay_30_mils, time=(0,0,0.030)

### 3.12.2 WHEN / WHENEND

The purpose of these commands is to delimit a series of commands which are associated with a specified condition. The command format is as follows:

```
when condition=<expr>
  .
   .
    .
whenend
```

condition | cond | c: This parameter specifies a logical expression which will determine when the commands between the WHEN and WHENEND are to be processed. When the expression becomes true (non-zero) they will be processed.

```
Example: ..lib f1
           LIB = SYS#LIB, F1
         ..execute prog=main, task=t1, event=e1
         ..timer e2, time=(0,5,0)
         ..when e1 or e2
         ..  if e1
         ..    clear e1
         ..    display 'task complete'
         ..  ifend
         ..  if e2
         ..    clear e2
         ..    stop t1
         ..    display 'task stopped at ' CAT clock(ampm)
         ..  ifend
         ..whenend
         ..
```

### 3.12.3 WAIT

The purpose of this command is to await a specified condition. The command format is as follows:

```
wait condition=<expr>
```

condition | cond | c: This parameter specifies a logical expression which will determine when control will be returned to the user. When the expression becomes true (non-zero) control will be returned.

```
Example: ..lib f1
           LIB = SYS#LIB, F1
         ..execute main, event=end_of_main
           .
            .
             .
         ..wait end_of_main
         ..
```

### 3.12.4 CAUSE

The purpose of this command is to cause the occurrence of a specified event. The command format is as follows:

cause event=<ref>

event ! ecb ! e: This parameter specifies the name of the event which is to be caused. If the event specified is already in a caused state no operation will take place.

```
Example: ..when e1
         ..  clear e1
         ..  display 'e1 occurred at ' cat clock(ampm)
         ..whenend
           .
            .
             .
         ..cause e1
           e1 occurred at 2:30 pm
         ..
```

### 3.12.5 CLEAR

The purpose of this command is to clear a specified event. The command format is as follows:

clear event=<ref>

event ! ecb ! e: This parameter specifies the name of the event which is to be cleared. If the event specified is already in a cleared state no operation will take place.

Example: clear e1

### 3.12.6 ENABLE

The purpose of this command is to enable a specified event. The command format is as follows:

enable event=<ref>

event ! ecb ! e: This parameter specifies the name of the event which is to be enabled. If the event specified is already in a enabled state no operation will take place.

Example: enable e1

### 3.12.7 DISABLE

The purpose of this command is to disable a specified event. If an event is caused while it is disabled, any outstanding WHEN commands which rely upon the event will not be honored until the event is enabled. The command format is as follows:

disable event=<ref>

event ! ecb ! e: This parameter specifies the name of the event which is to be disabled. If the event specified is already in a disabled state no operation will take place.

Example: disable e1

### 3.12.8 DISPLAY ! DI

The DISPLAY command can be used to display the current status of a specified event. The result will be written on SCL#OUTPUT. The command format is as follows:

display status=<ref>

status ! s: This parameter specifies the name of the event whose status is to be displayed.

```
Example: ..display status e1
         TYPE = EVENT
         CONDITION = CAUSED
         ACTION = DISABLED
        ..
```

## 3.13 MESSAGE SWITCHING

### 3.13.1 SEND

The purpose of this command is to send a message to another user or set of users. The command format is as follows:

send [to=<ident>] [until=<expr>] [mail]

to ! t: This parameter specifies the name of the user or set of users to which the message is to be sent. Omission of this parameter will cause the message to be sent to the operator. In this case if the job sending the message is local batch or interactive the message will be sent to SYSTEM#OP, and if it is remote batch the message will be sent to REMOVE#OP.

until ! u: This parameter specifies a string containing an end of data delimiter. Omission of this parameter will cause a default of ** to be assumed.

mail: This parameter specifies that the message is to be mailed to the specified user or members of the set of users if they are not currently online.

```
Example: ..send to ras
           SUPPLY MESSAGE
         ..Rick I have the library built now.
         ..**
           MESSAGE SENT
         ..
```

### 3.13.2 MAIL

The purpose of this command is to mail a message to a user or set of users. The command format is as follows:

```
mail [to=<ident>] [until=<expr>]
```

to ! t: This parameter specifies the name of the user or set of users to which the message is to be mailed. Omission of this parameter will cause the message to be mailed to the operator. In this case if the job mailing the message is local batch or interactive the message will be mailed to SYSTEM#OP, and if it is remote batch the message will be mailed to REMOVE#OP.

until ! u: This parameter specifies a string containing an end of data delimiter. Omission of this parameter will cause a default of ** to be assumed.

```
Example: ..mail to ras
          SUPPLY MESSAGE
         ..Rick I have the library built now.
         ..**
          MESSAGE MAILED
         ..
```

### 3.13.3 DISPLAY ! DI

The DISPLAY command can be used to display the contents of the user's mailbox. The result will be written on SCL#OUTPUT. The command format is as follows:

```
display mail
```

mail ! m: This parameter is a keyword parameter and is used to distinguish this variation of the DISPLAY command.

```
Example:   SCL VER 1.0  2:30 PM  MONDAY  JUNE 30, 1979
           PLEASE LOGIN
         ..login ras
           PROFILE PROCESSING INITIATED
           THERE IS MAIL IN YOUR MAILBOX
         ..display mail
           FROM WJH
           Rick I have the library built now.
         ..
```

### 3.13.4 PRINT ! PR

The PRINT command can be used to print the contents of the user's mailbox. The command format is as follows:

```
print mail
```

mail ! m: This parameter is a keyword parameter and is used to distinguish this variation of the PRINT command.

```
Example: print mail
```

### 3.13.5 DELETE ! DEL

The DELETE command can be used to delete the contents of the user's mailbox. The command format is as follows:

```
delete mail
```

mail ! m: This parameter is a keyword parameter and is used to distinguish this variation of the DELETE command.

```
Example: delete mail
```

## 3.14 SCL CONTROL STRUCTURES

### 3.14.1 RETURN

The purpose of this command is to return control from the command file in which it appears. The command format is as follows:

```
return [user]
```

user | u: This is a keyword parameter, i.e. it does not require values, and is used to indicate that control is to be returned to the next command line in the standard input file.

NOTE: The appearance of an end of file prior to a RETURN command will cause control to be returned as though a RETURN command had appeared as the last command in the command file.

```
Example: ..connect loc#alternate, job#display
         ..define run, file=runfile
         ..run (source,listing)
           FORTRAN I=SCL#PARAM(1).LOW, O=SCL#PARAM(2).LOW
           IF SCL#PARAM.COUNT EQ 2
             RETURN
         ..
```

### 3.14.2 FOR / FOREND

The purpose of these commands is to delimit a FOR loop. The loop format is as follows:

```
for index=<ref> range=<value list>
  .
   .
    .
forend
```

index | i: This parameter specifies the name of the variable to be used as the index variable.

range | r: This parameter specifies the values the index variable is to assume. When a range of values is quoted the assumed increment will be 1 if the left hand side is less than the right hand side and -1 otherwise.

```
Example: for i (1..5, 10)
           submit input=array_of_files(i)
         forend
```

### 3.14.3 IF / IFEND

The purpose of these commands is to delimit conditionally processed commands. The conditional command format is as follows:

```
if cond=<expr>
  .
   .
    .
ifend
```

condition | cond | c: This parameter specifies a logical expression which will determine whether or not the commands between the IF and the IFEND are to be processed. If and only if the logical expression is true (non-zero) will they be processed.

```
Example: if (x eq y)
           display ('x = y')
           go to label1
         ifend
```

### 3.14.4 GOTO ! GO

The purpose of this command is to transfer control to the command line containing the label specified. The command format is as follows:

```
goto label=<ident>
```

label ! to ! l: This parameter specifies the name of the label to which control is to be transferred.

Example: go to label1

### 3.14.5 LABELBLOCK / LABELEND

The purpose of these commands is to delimit a label block. All labels defined in SCL are local to the label block in which their definition appears and can only be referenced from within that block. The general format of a label block is as follows:

```
labelblock
  .
   .
    .
labelend
```

NOTE: The maximum number of labels that can be defined in a label block is 16 and the maximum depth of label block nesting is 16.

## 3.15 <u>MISCELLANEOUS</u>

### 3.15.1 MICRO ! MIC

The purpose of this command is to define a micro. The command format is as follows:

```
micro name=<ref> text=<expr>
```

name ! n: This parameter specifies the name of the micro to be defined.

text ! t: This parameter specifies the string of SCL text comprising the micro.

```
Example: ..micro k, text='*1024'
         ..limit punch=5 k
         ..
```

### 3.15.2 ACCEPT

The purpose of this command is to read data from the standard input. The command format is as follows:

```
accept value=<ref list>
```

value ! val ! v: This parameter specifies the names of one or more variables into which the data is to be read.

```
Example: ..collect f1
           SUPPLY RECORDS
         ..accept (x,y,z)
         ..execute main, param=(x,y,z)
         ..**
           RECORDS COLLECTED
         ..define cmd, file=f1
         ..cmd
           SUPPLY X
         ..12, 3.4
           SUPPLY Z
         ..'This is a silly string'
         ..
```

### 3.15.3 DISPLAY ! DI

The DISPLAY command can be used to display a list of values. The result will be written on SCL#OUTPUT. The command format is as follows:

```
display value=<expr list>
```

value ! val ! v: This parameter specifies one or more expressions whose values are to be displayed.

```
Example: ..display (x,y,z)
           12
           3.4
           This is a silly string
         ..
```

### 3.15.4 SET

The system allocates an array of 64 boolean elements called SCL#TOGGLE for each job known to the system. Each element of the array is identified by an ordinal in the range 1..64. The initial setting of the SCL#TOGGLE array is a site parameter. The purpose of this command is to

set/clear SCL toggles. The command format is as follows:

```
set [on=<expr list>] [off=<expr list>]
```

on ! o: This parameter specifies the toggles which are to be set on.

off: This parameter specifies the toggles which are to be set off.

```
Example: set on=(1,2)
```

The following toggles have been assigned to the described functions:

1 - This toggle controls the dialog mode of the SCL interpreter. When this toggle is on the interpreter will solicit missing required parameters from the user. Dialog mode is only meaningful for interactive jobs and therefore the toggle has no effect in a batch job.

```
Example: ..set on 1
         ..connect
           SUPPLY STREAM PARAMETER
         ..loc#alternate
           SUPPLY FILE PARAMETER
         ..job#display
         ..
```

### 3.15.5 BASEFILE

The purpose of this command is to appoint a file as the base file. The command format is as follows:

```
basefile file=<ref>
```

file ! fcb ! f: This parameter specifies the name of the file being appointed.

```
Example: basefile abc
```

## 3.15.6 COPY

The purpose of this command is to copy data. The command format is as follows:

```
copy [from=<ref>] into=<ref>
```

from ! fr ! f: This parameter specifies the name of the data source. Omission of this parameter will cause the base file to be used as the data source.

into ! to: This parameter specifies the name of the data destination. Omission of this parameter will cause the base file to be used as the destination.

NOTE: The following table indicates the valid type combinations for the source and destination.

(to be supplied)

Example: copy from n1 to n2

## 4.0 SYSTEM REPERTOIRE

The system repertoire is comprised of a set of commands which represent an externalization of the operating system requests available to a running program. These commands allow the user to invoke most of the operating system request processors directly. The command descriptions are contained in the various sections of the GDS at the points where the requests are defined.

## 5.0 OPERATOR REPERTOIRE

(to be supplied)

## 6.0 SAMPLE JOBS

```
JOB WJH " COBOL COMPILE AND EXECUTE "
COLLECT SOURCE
.
 . COBOL SOURCE DECK APPEARS HERE
  .
**
COBOL I=SOURCE, O=OBJECT, L=LISTING
SAVE OBJECT
PRINT LISTING
COLLECT DATA UNTIL '/*'
.
 . DATA DECK APPEARS HERE
  .
/*
OBJ ADD=OBJECT
EXEC PROG=MAIN, PARAM=DATA
JOBEND
```

```
JOB WJH " COBOL EXECUTE ONLY "
COLLECT DATA UNTIL '/*'
.
 . DATA DECK APPEARS HERE
  .
/*
OBJ ADD=OBJECT
EXEC PROG=MAIN, PARAM=DATA
JOBEND
```

## 7.0 EXPRESSIONS

An expression is an algorithm used for computing a value. An expression may be a single constant or a name, or it may be a combination of them, including operators and other delimiters. Parentheses within an expression indicate that the parenthesized portion is considered as a single value in relation to its surrounding operators. The parenthesized portion of an expression is evaluated first, with the innermost parenthesized material taking precedence. Although an expression may contain more than one data item, it represents the single value obtained after the expression is evaluated.

## 7.1 OPERATORS

The valid SCL operators are as follows:

| | |
|---|---|
| + .................. | addition |
| - .................. | subtraction |
| * .................. | multiplication |
| / .................. | division |
| ** ................. | exponentiation |
| CAT ................ | concatenation |
| GT ................. | greater than |
| GE ................. | greater than or equal to |
| LT ................. | less than |
| LE ................. | less than or equal to |
| EQ ................. | equal to |
| NE ................. | not equal to |
| AND ................ | and |
| OR ................. | or |
| NOT ................ | not |

## 7.2 OPERANDS

When operations are performed in SCL expressions the operands are converted to type real or string depending upon the operator involved.

The arithmetic operators +, -, *, / and ** insist on numeric operands and yield a real result. The string operator CAT insists on string operands and yields a string result. The relational operators GT, GE, LT, LE, EQ and NE accept numeric or string operands but insist that both be of the same class, i.e. numeric or string. These operators yield an integer result of 1 or 0 depending upon whether the relation is true or false respectively. The logical operators AND, OR and NOT insist on numeric operands and yield an integer result of 1 or 0 depending upon the truth value of the logical expression. Zero is considered false and non-zero is considered true by these operators.

NUMERIC OPERANDS

- INTEGER
- REAL
- LNS SUBRANGE
- LNS INTEGER
- LNS REAL
- LNS BOOLEAN - TRUE = 1 and FALSE = 0
- LNS EVENT - CAUSED = 1 and CLEAR = 0

STRING OPERANDS

- STRING
- LNS STRING

## 7.3 ORDER OF EVALUATION

Expressions are scanned from left to right. The following table reflects the precedence of the operators. The exponentiate operator has the highest precedence and the OR operator has the lowest precedence.

| | |
|---|---|
| ** | exponentiate |
| * ! / | multiply ! divide |
| + ! - | add ! subtract |
| CAT | concatenate |
| GT ! GE ! LT ! LE ! EQ ! NE | relationals |
| NOT | not |
| AND | and |
| OR | or |

To override the normal order of evaluation the user must parenthesize the expression.

| expression | result |
|---|---|
| 3+3 | 6 |
| 3-3 | 0 |
| 3*3 | 9 |
| 3/3 | 1 |
| 3**3 | 27 |
| 3 GT 3 | 0 |
| 3 GE 3 | 1 |
| 3 LT 3 | 0 |
| 3 LE 3 | 1 |
| 3 EQ 3 | 1 |
| 3 NE 3 | 0 |
| 3 AND 3 | 1 |
| 3 OR 3 | 1 |
| NOT 3 | 0 |
| 3+3*3 | 12 |
| (3+3)*3 | 18 |

## 7.4 FUNCTIONS

### 7.4.1 CLOCK

The CLOCK function returns a string containing the clock time. The calling sequence is as follows:

clock(format)

format: This argument specifies the format in which the time is to be returned. The defined formats are as follows:

AMPM: This specification will cause the time to be returned in hours and minutes am or pm (e.g. 2:30 pm).

HM: This specification will cause the time to be returned in hours and minutes (e.g. 14:30).

HMSM: This specification will cause the time to be returned in hours, minutes, seconds and milliseconds (e.g. 14:30:29.435).

```
Example: ..display 'Time now is ' cat clock(ampm)
           Time now is 2:30 pm
         ..
```

### 7.4.2 DATE

The DATE function returns a string containing the date. The calling sequence is as follows:

date(format)

format: This argument specifies the format in which the date is to be returned. The defined formats are

as follows:

MDY: This specification will cause the date to be returned as month day, year (e.g. June 30, 1979).

ISO: This specification will cause the date to be returned as year-month-day (e.g. 1979-06-30).

JULEAN: This specification will cause the date to be returned as YYDDD (e.g. 79181).

```
Example: ..display clock(ampm) cat '  ' cat date(mdy)
          2:30 pm  June 30, 1979
         ..
```

### 7.4.3 UNIQUE

The UNIQUE function returns a string containing a unique identifier each time it is referenced. The calling sequence is as follows:

unique(format)

format: This argument specifies whether a user identifier or system identifier is to be generated. USER will cause a user identifier to be generated and SYSTEM will cause a system identifier to be generated The format of the generated identifier will be as follows:

U_XXXXXX or S#XXXXXX

where X will be in the set of digits or upper case letters.

```
Example: strvar = unique(user)
```

### 7.4.4 REF

The REF function returns a reference name whose string representation is supplied as an argument to the function. This function allows the user to reference an object whose name is contained in a string variable, i.e., indirect the reference. The calling sequence is as follows:

```
ref(string)
```

string: This argument specifies the string representation of the name being referenced.

Example: create file = ref(strvar)

### 7.4.5 SUBSTR

The SUBSTR function returns a substring of user defined length from a user supplied string. The calling sequence is as follows:

```
substr(string, start [,length])
```

string: This argument specifies the string from which the substring is to be extracted.

start: This argument specifies the starting point of the substring within the string - the 1st character being 1.

length: This argument specifies the length of the substring being extracted. Omission of this argument will cause the remainder of the string to be extracted.

Example: strvar = substr('...substring...',4,9)

### 7.4.6 LOC

The LOC function returns a pointer to the data area associated with the LNS name supplied as an argument. The calling sequence is as follows:

```
loc(name)
```

name: This argument specifies the name whose data pointer is to be returned.

Example: ptrvar = loc(f1.owner)

# 8.0 ASSIGNMENT

Values are assigned to LNS variables by use of the assignment statement. The general format of an assignment statement is as follows:

<ref> = <expr>

The following diagram shows the conversion rules applied to mixed mode assignment. The rows indicate the type of the variable on the left hand side of the equal sign and the columns indicate the type of the result of the expression on the right hand side.

| | integer | real | string | other |
|---|---|---|---|---|
| integer | rule_1 | rule_2 | error | error |
| real | rule_3 | rule_1 | error | error |
| string | error | error | rule_4 | error |
| alias | error | error | rule_4 | error |
| other | error | error | error | rule_5 |

rule_1: The assignment is direct provided tne dimension of each side is equal. This rule will be relaxed in future versions.

rule_2: The real value is truncated and the result is assigned according to rule_1.

rule_3: The integer value is converted to real and the result is assigned according to rule_1.

rule_4: If the length of the string on the left is greater than the length of the string on the right the string on the right is blank filled to the right. If the length of the string on the left is less than the length of the string on the right the string on the right is truncated on the right. The result is then assigned according to rule_1.

rule_5: If the type of the left is equal to the type of the right assignment takes place according to rule_1. If they are of unlike type an error condition will result.

Example: x(2).y = 12.34

# 9.0 LANGUAGE SYNTAX

The following syntax definition is designed for readability and does not represent a rigorous definition of the language. It is intended to give the reader sufficient information to be able to construct SCL text.

## 9.1 CHARACTER SET

characters used for identifiers

| | |
|---|---|
| A...Z ! a...z | letters |
| 0...9 | digits |
| # | number sign |
| $ | dollar sign |
| @ | commercial at |
| _ | underline |

characters used for integer constants

| | |
|---|---|
| 0...9 | digits |
| A...F ! a...f | hex digits |
| ( | open parentheses |
| ) | close parentheses |

characters used for real constants

| | |
|---|---|
| 0...9 | digits |
| . | decimal point |
| E ! e | letter e |
| + | plus |
| - | minus |

characters used for operators

| | |
|---|---|
| + | plus |
| - | minus |
| * | asterisk |
| / | slant |

characters used for qualification

| | |
|---|---|
| -> | right arrow digraph |
| . | period |

characters used for punctuation

| | |
|---|---|
| b̸ | space |
| ( | open parentheses |
| ) | close parentheses |
| , | comma |
| : | colon |
| ; | semicolon |
| = | equal sign |

string delimiter

' ................. apostrophe

comment delimiter

" ................. quotation mark

NOTE: All ascii characters not listed in the above character set have no defined meaning to the SCL interpreter. These characters may however be used as data characters by the user.

## 9.2 FORMAL DESCRIPTION

```
<input line> ::= <command line> | <data line>

<command line> ::= [<label>]...  <command list>

<label> ::= <ident> :

<command list> ::= <command> [; <command>]...

<command> ::= <command name> [ < ƀ | , > <param list> ]
            | <assignment>
            | <query>

<command name> ::= <ref>

<param list> ::= <param> [ < ƀ | , > [<param>] ]...

<param> ::= <param name>
          | <param name> < ƀ | = > <value list>
          | <value list>

<param name> ::= <ident>

<value list> ::= <value> | (<value> [,<value>]...)

<value> ::= <expr> [<...> <expr>]...

<assignment> ::= <ref> = <expr>

<query> ::= <expr>

<expr> ::= <lterm> [ ƀ OR ƀ <lterm>]...

<lterm> ::= <lfactor> [ ƀ AND ƀ <lfactor>]...

<lfactor> ::= [NOT ƀ ] <lprimary>

<lprimary> ::= <sterm> [ ƀ <relation> ƀ <sterm>]...

<relation> ::= GT | GE | LT | LE | EQ | NE

<sterm> ::= <term> [ ƀ CAT ƀ <term>]...

<term> ::= [+ | -] <factor> [< + | - > <factor>]...

<factor> ::= <primary> [< * | / > <primary>]...
```

```
<primary> ::= <operand> [** <operand>]...

<operand> ::= <constant>
            | <ref>
            | <function>
            | (<expr>)

<constant> ::= <integer> | <real> | <string>

<integer> ::= <digit> [<hex digit>...]  [(<base>)]

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<hex digit> ::= <digit> | A | B | C | D | E | F
                        | a | b | c | d | e | f

<base> ::= <digit>...

<real> ::= <digit>... .<digit>... [E < + | - > <digit>...]

<string> ::= '[ascii character]...'

<ref>   ::=   [<ident>->]   <ident>   [(<expr>)]   [.<ident>
[(<expr>)] ]...

<ref list> ::= (<ref> [,<ref>]...)

<ident> ::= <alpha> [<alpha> | <digit>]...

<alpha> ::= <letter> | # | $ | @ | _

<letter> ::= A | B | C | D | E | F | G | H
           | I | J | K | L | M | N | O | P
           | Q | R | S | T | U | V | W | X
           | Y | Z | a | b | c | d | e | f
           | g | h | i | j | k | l | m | n
           | o | p | q | r | s | t | u | v
           | w | x | y | z

<ident list> ::= (<ident> [,<ident>]...)

<function> ::= <ident> [<expr list>]

<expr list> ::= (<expr> [,<expr>]...)

<comment> ::= "[ascii character]..."

<data line> ::= [ascii character]...
```

NOTE: Spaces may be used freely between syntactic units to inprove readability and must be used where indicated. Whenever a space is allowed, multiple spaces may be specified.

# 10.0 COMMAND WRITERS GUIDE

This section is intended to provide guidance for those who wish to produce new procedures in order to interface request macros to the external user, that is, to externalize a request.

The actual production of a new command interface procedure is a fairly simple task, and it is recognized that the provision of the command procedure is best performed by the person or persons providing the request macro. This implies that the documentation for the command procedure appears with and complements that of the request macro.

## 10.1 NAMING CONVENTIONS

### 10.1.1 COMMAND PROCEDURE NAMING CONVENTION

There is a consistent naming convention for command procedures, as follows.

If one wishes to externalize the ATTACH request of the LNS section, then the command procedure name will be:

SCL#LNS#ATTACH

which means "The SCL (System Command Language) interface to the ATTACH request processor of the LNS (Logical Name Space) section". This naming convention should be followed throughout.

The PDT (Parameter Description Table) must be declared within the command procedure with the XDCL attribute and must be given the name of the procedure suffixed by _PDT.

SCL#LNS#ATTACH_PDT

### 10.1.2 PARAMETER KEYWORD NAMING CONVENTION

A convention for parameter names, synonyms and abbreviations has been empirically established. The rules for choosing these names are fairly simple:

- o Use the parameter name in full.
- o Choose as many synonyms and abbreviations as thought necessary. Any abbreviations chosen should have mnemonic significance, or be easily recognized because of widespread prior usage, and should preferably be easily pronounced. If possible, try to satisfy all these conditions.

- o The final synonym is the initial letter of the parameter. It is recognized that this can in certain circumstances lead to conflict. When this occurs, use the initial letters of the parameters in order of ocurrence until conflict.

An example of the LNS#DECLARE command will illustrate these points.

```
                SEG                    ENT
LNS#DECLARE [SEGMENT=<IDENT>] ENTRY=<IDENT>
                S                      E

                 LEN                  DIM
[TYPE=<IDENT>] [LENGTH=<EXPR>] [DIMENSION=<EXPR>]
 T               L                    D

[STATUS=<NAME>]
```

Note that in this example, there would be a clash between the "S" in "SEGMENT" and "STATUS". Since "STATUS" is not the first ocurrence of a parameter beginning with "S" in the command, the "S" initial is used for "SEGMENT".

A list of parameter names, synonyms and abbreviations are attached at the end of this section. This is for guideline purposes only and no attempt will be made to keep this up to date with future enhancements.

## 10.2 FORMAT AND LAYOUT OF DOCUMENTATION

When a Request Processor Macro Call Format is documented, the corresponding Command Format will be documented along with it.

The necessary documentation includes :

- o The macro name.
- o A short explanatory text, which briefly describes the function.
- o A model reference to the request macro.
- o Explanations of each Parameter. This should also include the manner in which null parameters are to be conveyed to the request processor, and the effect of the null parameter on the request.
- o The syntax of the System Repertoire Command.
- o Explanations of each parameter. The alternative keywords for the parameters should be given at this point.

### 10.2.1 EXAMPLE OF DOCUMENTATION LAYOUT

LNS#DECLARE

The purpose of the LNS#DECLARE request is to declare an entry in the LNS. The macro format is as follows.

LNS#DECLARE ( SEGMENT,ENTRY,TYPE,LENGTH,DIM,LOCATOR,STATUS )

SEGMENT : The segment parameter specifies a string containing the name of the segment in which the entry is to be declared. Omission of the segment parameter ( indicated by a blank string ) will cause the entry to be declared in the most local segment.

10.0 COMMAND WRITERS GUIDE
10.2.1 EXAMPLE OF DOCUMENTATION LAYOUT

ENTRY : The entry parameter specifies a string containing the name of the entry being declared.

TYPE : The type parameter specifies a string containing the type of the entry being declared. Omission of the type parameter ( indicated by a blank string ) will cause an entry of type INTEGER to be declared. The valid LNS types are those described under "data types" or any complex type previously defined by LNS#RECORD and LNS#FIELD.

LENGTH : The length parameter is only meaningful when declaring string variables. In this case the length parameter specifies an integer containing the number of bytes to be allocated for the string. Omission of the length parameter ( indicated by a 0 ) will cause a default of 32 to be assumed.

DIM : The dim parameter specifies an integer containing the dimension of the entry being declared. Omission of the dim parameter ( indicated by a 0 ) will cause a default of 1 to be assumed.

LOCATOR : The locator parameter specifies a pointer variable into which the system will place a pointer to the LNS internal descriptor for the entry. If the user specifies the same pointer on subsequent requests for the entry a search will be eliminated.

STATUS : The status parameter specifies a variable into which the status record is to be placed. The status codes returned are described under "error conditions".

The System Repertoire Command Format is as follows.

```
LNS#DECLARE [SEGMENT=<IDENT>] ENTRY=<IDENT> [TYPE=<IDENT>]

      [LENGTH=<EXPR>] [DIMENSION=<EXPR>] [STATUS=<REFNAME>]
```

SEGMENT ! SEG ! S : This parameter specifies the name of the LNS segment in which the entry is to be declared. Omission of the segment pa~ameter will cause the entry to be declared in the most local segment.

10.0 COMMAND WRITERS GUIDE
10.2.1 EXAMPLE OF DOCUMENTATION LAYOUT

ENTRY ! E : This parameter specifies the name of the entry to be declared.

TYPE ! T : This parameter specifies the type of the entry to be declared. Omission of the type parameter will cause the entry to be declared as type integer.

LENGTH ! LEN ! L : This parameter is only meaningful when declaring string variables. In this case it specifies the number of bytes to be allocated for the string. Omission of the length parameter will cause a default of 32 to be assumed.

DIMENSION ! DIM ! D : This parameter specifies the number of occurrences of the entry to be declared. Omission of the dimension parameter will cause a default of 1 to be assumed.

STATUS : This parameter specifies a variable into which the status is to be returned. Omission of this parameter will cause the SCL error handler to be invoked upon the occurrence of an error condition.

## 10.3 CODING A COMMAND PROCEDURE

It is desirable that the layout and coding of command procedures should retain a degree of consistency throughout the system. This will ensure that maintenance of coding will be somewhat eased, since a consistent layout will make familiarization easier for the maintenance programmer.

Instead of attempting to give 'verbal' descriptions of the manner in which Command Procedures should be laid out and the declarations necessary, an example of a typical Command Procedure is attached at the end of the section. Hopefully this example will be followed reasonably closely in style for the reasons mentioned above.

## 10.4 COMMAND PROCEDURE INTERFACE

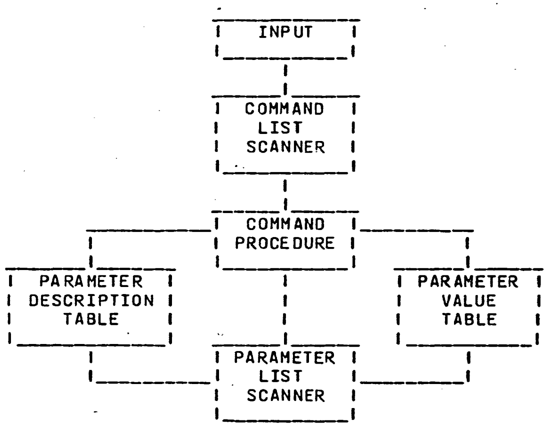

Fig 10-1

## 10.5 LIST OF KEYWORDS AND SYNONYMS TO DATE

ADD
ADDRESS - ADDR
ATTRIBUTE - ATTR

BLOCK - BLK
BUFFER - BUF

CATALOG - CATLG
CHAIN
CLOCK
COBOL - COB
CONDITION - COND
COPIES

DELETE - DEL
DESCRIPTOR - DESC
DIMENSION - DIM

EDITION - ED
ENTRY - ENT

FIELD
FILE - FCB
FORM
FORTRAN - FORT, FTN
FROM - FR

GENERATION - GEN

INCREMENT - INCR
INITIALIZE - INIT
INPUT - INP
INTO - TO
ITEM

KEY

LABEL
LENGTH - LEN
LISTING - LIST

MAIL

NAME

OBJECT - OBJ

OUTPUT - OUT

PARAMETER - PARAM
POSITION - POSN
PRINT - PR
PROCEDURE - PROC
PROGRAM - PROG, PCB
PUNCH - PU

QUALIFIER - QUAL

RECORD - REC
REJECT - REJ
RESOURCE

SEGMENT - SEG
SITE
STATUS
STREAM
SWL

TASK - TCB
TERMINAL
TIME
TRACK - TRK
TRAP
TYPE

UNIT
UNITSET - UCB
UNTIL
USAGE - USE
USER

VALUE - VAL
VOLID - VID
VOLSET - VCB

WORD - WRD

## 10.6 EXAMPLE OF A SWL COMMAND PROCEDURE

( To be Supplied )

# 11.0 APPENDIX A ... SCL SCANNERS

The information in this appendix is internal maintainance level documentation and is included in this document to assist those responsible for writing command procedures in support of the system repertoire. The data structures defined are subject to change at the field level, however, the different of data structures defined will in all probability be those found in the final BTS.

## 11.1 DATA STRUCTURES

### 11.1.1 SCL STRING - SCL#STRING

The definition of SCL#STRING is as follows:

```
TYPE
  SCL#STRING = RECORD
    LHI: 1..256, " left hand index "
    RHI: 0..255, " right hand index "
    BUF: STRING(255) OF CHAR, " character buffer "
  RECEND;
```

LHI: This field contains the position of the 1st character of the string within the buffer. The position of the 1st character of the buffer is 1 and therefore if the string is left justified in the buffer LHI=1.

RHI: This field contains the position of the last character of the string within the buffer. The length of a string is defined to be RHI-LHI+1.

BUF: This field contains the characters comprising the string.

### 11.1.2 SCL SYMBOL - SCL#SYMBOL

The definition of SCL#SYMBOL is as follows:

```
TYPE
  SCL#SYMBOL = RECORD
    TYP: INTEGER, " type code "
    SV: SCL#STRING, " string value "
  RECEND;
```

TYP: This field contains the encoded type of the symbol.

SV: This field contains the string representation of the symbol.

### 11.1.3 SCL TOKEN - SCL#TOKEN

The definition of SCL#TOKEN is as follows:

```
TYPE
  SCL#TOKEN = RECORD
    TYP: INTEGER, " type code "
    DESC: LNS#DESC, " LNS descriptor "
    IV: INTEGER, " integer value "
    RV: REAL, " real value "
    SV: SCL#STRING, " string value "
  RECEND;
```

TYP: This field contains the encoded type of the token.

DESC: This field contains the LNS descriptor of the name when the token is of type name.

IV: This field contains the numeric value when the token is of type integer.

RV: This field contains the numeric value when the token is of type real.

SV: This field contains the string value when the token is of type string. When the token is of type name this field contains the string representation of the name and when the token is of type foreign this field contains the string representation of the foreign text.

### 11.1.4 TYPE SCL#PDT = RECORD

```
    CMD: STRING(31) OF CHAR, " command name "
    REQ: SET OF 1..64, " set of required parameters "
    FGN: SET OF 1..64, " set of foreign text parameters "
    RNG: SET OF 1..64, " set of range parameters "
    MIN: ARRAY[1..64] OF INTEGER, " minimum # of values "
    MAX: ARRAY[1..64] OF INTEGER, " maximum # of values "
    PID: ARRAY[1..64] OF " parameter identifiers "
           RECORD
             NAME: STRING(31) OF CHAR, " parameter name "
             POSN: INTEGER, " parameter position "
           RECEND,
  RECEND;
```

REQ: This field contains the set of required parameters. If a parameter appears in this set and is not supplied by the user in the parameter list an error condition will result.

FGN: This field contains the set of foreign text parameters. The values for these parameters will be returned in LOV as type foreign and the SV field will contain the text specified.

RNG: This field contains the set of parameters which allow ranges as values. If a range is supplied by the user for a parameter which does not appear in this set an error condition will result.

MIN: This field contains the minimum number of values allowed for each parameter. The field only applies when the parameter is supplied by the user in the parameter list. It is reasonable, therefore, to specify a PDT where a given parameter does not appear in the REQ set but whose minimum number of values is greater than 0. In this case the parameter is optional, but if supplied must contain at least the stated number of values.

MAX: This field contains the maximum number of values allowed for each parameter. This field only applies when the parameter is supplied by the user in the parameter list.

PID: This field contains the names and positions of the parameters. Every parameter supplied by the user must be defined in the PID to achieve normal completion. If the parameter is specified by name its name must appear in the PID, and if the parameter is specified positionally its position must appear in the PID.

### 11.1.5 SCL PARAMETER VALUE TABLE - SCL#PVT

The definition of SCL#PVT is as follows:

```
TYPE
  SCL#PVT = RECORD
    DEFP: SET OF 1..64, " set of defined parameters "
    DEFN: SET OF 1..64, " set of defined parameter names "
    CNT: ARRAY[1..10] OF INTEGER, " # of values / parameter "
    LOV: ARRAY[1..64,1..5] OF SCL#TOKEN, " low values "
    HIV: ARRAY[1..64,1..5] OF SCL#TOKEN, " high values "
  RECEND;
```

DEFP: This field contains the set of quoted parameters. The number of each parameter supplied by the user in the parameter list will be contained in the DEFP set upon normal completion.

DEFN: This field contains the set of names quoted in the list.

CNT: This field contains the number of values quoted for the parameter.

LOV: This field contains the low values quoted in the list.

HIV: This field contains the high values quoted in the list. When a range is not quoted the low value and high value are equal ... LOV[P#,V#] = HIV[P#,V#].

### 11.1.6 OS STATUS - OS#STATUS

The definition of OS#STATUS is as follows:

```
TYPE
  OS#STATUS = RECORD
    LEVEL: 0..0FF(16), " general level indicator "
    FROM: STRING(2) OF CHAR, " issuing os section "
    ST_CODE: 0..0FFFF(16), " specific status code "
    MESG: STRING(32) OF CHAR, " message mask "
  RECEND;
```

LEVEL: This field contains the general status, the values of which are shown in the IPLOS structure overview document.

FROM: This field contains the operating system section that issued the status.

ST_CODE: This field contains the specific status code issued.

MESG: This field contains the message mask to be used by the system message generator when constructing diagnostic messages.

## 11.2 PROCEDURES

### 11.2.1 SCAN SYMBOL - SCL#GET_SYMBOL

The purpose of this procedure is to scan the source and return the next symbol. The calling sequence is as follows:

SCL#GET_SYMBOL(source,symbol,status)

source: This parameter specifies the string of text to be scanned. As the text is scanned the left hand index of the string is updated to reflect the current scan position.

symbol: This parameter specifies the name of a variable into which the symbol is to be returned. The symbol types returned are as follows:

| | |
|---|---|
| type_space ......... | string of spaces |
| type_an ............ | string of alphanumeric characters |
| type_digit ......... | string of digits |
| type_foreign ....... | foreign character |
| type_end ........... | end of text |

status: This parameter specifies the name of a variable into which the status is to be returned.

### 11.2.2 SCAN TOKEN - SCL#GET_TOKEN

The purpose of this procedure is to scan the input and return the next token. If the token stack (SCL#TOKEN_STACK) is not empty the token at the top of the stack will be removed and returned to the caller. If the token stack is empty the source will be scanned for the next token. Names are looked up in the LNS, numbers are converted to their internal representation and mnemonic operators are classified by this procedure. The calling sequence is as

follows:

SCL#GET_TOKEN(source,token,status)

source: This parameter specifies the string of text to be scanned. As the text is scanned the left hand index of the string is updated to reflect the current scan position.

token: This parameter specifies the name of a variable into which the token is to be returned. The token types returned are as follows:

| | |
|---|---|
| type_unknown ....... | unknown value |
| type_name .......... | name |
| type_integer ....... | integer value |
| type_real .......... | real value |
| type_string ....... | string value |
| type_add ........... | + |
| type_sub ........... | - |
| type_mult .......... | * |
| type_div ........... | / |
| type_exp ........... | ** |
| type_cat ........... | CAT operator |
| type_gt ............ | GT operator |
| type_ge ............ | GE operator |
| type_lt ............ | LT operator |
| type_le ............ | LE operator |
| type_eq ............ | EQ operator |
| type_ne ............ | NE operator |
| type_and ........... | AND operator |
| type_or ............ | OR operator |
| type_not ........... | NOT operator |
| type_assign ........ | = |
| type_open .......... | ( |
| type_close ......... | ) |
| type_comma ......... | , |
| type_period ........ | . |
| type_ellipsis ...... | .. |
| type_colon ......... | : |
| type_semicolon ..... | ; |
| type_foreign ....... | foreign text |
| type_end ........... | end of text |

status: This parameter specifies the name of a variable into which the status is to be returned.

### 11.2.3 SCAN REFERENCE NAME - SCL#REF

The purpose of this procedure is to scan the string representation of a name and return its LNS descriptor. The calling sequence is as follows:

SCL#REF(sv,desc,status)

sv: This parameter specifies the string to be converted.

desc: This parameter specifies the name of a variable into which the LNS descriptor is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 11.2.4 STACK TOKEN - SCL#STACK_TOKEN

The purpose of this procedure is to stack a token for later retrieval by SCL#GET_TOKEN. The calling sequence is as follows:

SCL#STACK_TOKEN(token,status)

token: This parameter specifies the token to be stacked.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 11.2.5 SCAN AND EVALUATE EXPRESSION - SCL#EXPR

The purpose of this procedure is to scan and evaluate an expression. If the expression is composed of a single name the token returned will contain the LNS descriptor for the name but the value described will not be returned. This

is done to allow call by reference in parameter lists. If the expression is composed of operands and operators it will be evaluated and the token returned will describe the resultant value. The calling sequence is as follows:

SCL#EXPR(source,token,status)

source: This parameter specifies the string of text to be scanned. As the text is scanned the left hand index of the string is updated to reflect the current scan position.

token: This parameter specifies the name of a variable into which the token is to be returned. The token types returned are as follows:

| | |
|---|---|
| type_name .......... | name |
| type_integer ....... | integer value |
| type_real .......... | real value |
| type_string ........ | string value |
| type_foreign ....... | foreign text |

status: This parameter specifies the name of a variable into which the status is to be returned.

### 11.2.6 SCAN PARAMETER LIST - SCL#PLIST

The purpose of this procedure is to scan a parameter list. A description of the parameters expected is passed in a table called the parameter description table (SCL#PDT) and the results are returned in a table called the parameter value table (SCL#PVT). With the exception of foreign text parameters, all values are passed to SCL#EXPR for evaluation. Foreign text parameters are scanned directly by this procedure and serve as an excape mechanism for the user who wishes to retain the SCL parameter list structure but needs to accept values which do not conform to the SCL syntax. Values in foreign text parameters are delimited by an unenclosed comma, semicolon or end of string. The calling sequence is as follows:

SCL#PLIST(source,pdt,pvt,status)

source: This parameter specifies the string of text to be scanned. As the text is scanned the left hand index of the string is updated to reflect the current scan position.

pdt: This parameter specifies a parameter description table.

pvt: This parameter specifies the name of a variable into which the parameter value table is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 11.2.7 SCAN COMMAND LIST - SCL#CLIST

The purpose of this procedure is to scan and interpret a command list. The calling sequence is as follows:

```
SCL#CLIST(source,status)
```

source: This parameter specifies the string of text to be scanned. As the text is scanned the left hand index of the string is updated to reflect the current scan position.

status: This parameter specifies the name of a variable into which the status is to be returned.

## 12.0 APPENDIX B ... CONVERSION PROCEDURES

The information in this appendix is internal maintainance level documentation and is included in this document to assist those responsible for writing command procedures in support of the system repertoire. The data structures defined are subject to change at the field level, however, the different of data structures defined will in all probability be those found in the final BTS.

## 12.1 DATA STRUCTURES

### 12.1.1 SCL STRING - SCL#STRING

The definition of SCL#STRING is as follows:

```
TYPE
  SCL#STRING = RECORD
    LHI: 1..256, " left hand index "
    RHI: 0..255, " right hand index "
    BUF: STRING(255) OF CHAR, " character buffer "
  RECEND;
```

LHI: This field contains the position of the 1st character of the string within the buffer. The position of the 1st character of the buffer is 1 and therefore if the string is left justified in the buffer LHI=1.

RHI: This field contains the position of the last character of the string within the buffer. The length of a string is defined to be RHI-LHI+1.

BUF: This field contains the characters comprising the string.

### 12.1.2 SCL TOKEN - SCL#TOKEN

The definition of SCL#TOKEN is as follows:

```
TYPE
  SCL#TOKEN = RECORD
    TYP: INTEGER, " type code "
    DESC: LNS#DESC, " LNS descriptor "
    IV: INTEGER, " integer value "
    RV: REAL, " real value "
    SV: SCL#STRING, " string value "
  RECEND;
```

TYP: This field contains the encoded type of the token.

DESC: This field contains the LNS descriptor of the name when the token is of type name.

IV: This field contains the numeric value when the token is of type integer.

RV: This field contains the numeric value when the token is of type real.

SV: This field contains the string value when the token is of type string. When the token is of type name this field contains the string representation of the name and when the token is of type foreign this field contains the string representation of the foreign text.

### 12.1.3 OS STATUS - OS#STATUS

The definition of OS#STATUS is as follows:

```
TYPE
  OS#STATUS = RECORD
    LEVEL: 0..0FF(16), " general level indicator "
    FROM: STRING(2) OF CHAR, " issuing os section "
    ST_CODE: 0..0FFFF(16), " specific status code "
    MESG: STRING(32) OF CHAR, " message mask "
  RECEND;
```

LEVEL: This field contains the general status, the values of which are shown in the IPLOS structure overview document.

FROM: This field contains the operating system section that issued the status.

ST_CODE: This field contains the specific status code issued.

MESG: This field contains the message mask to be used by the system message generator when constructing diagnostic messages.

## 12.2 PROCEDURES

### 12.2.1 CONVERT STRING TO INTEGER - SCL#SV_IV

The purpose of this procedure is to convert a string containing the character representation of an integer to its internal representation. The calling sequence is as follows:

SCL#SV_IV(sv,iv,status)

sv: This parameter specifies the string to be converted.

iv: This parameter specifies the name of a variable into which the integer value is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 12.2.2 CONVERT INTEGER TO STRING - SCL#IV_SV

The purpose of this procedure is to convert the internal representation of an integer value to its character string representation with the base specified. The calling sequence is as follows:

SCL#IV_SV(iv,base,sv)

iv: This parameter specifies the integer value to be converted.

base: This parameter specifies an integer in the range 2..16 denoting the base of the desired representation.

sv: This parameter specifies the name of a variable

into which the string is to be returned.

### 12.2.3 CONVERT STRING TO REAL - SCL#SV_RV

The purpose of this procedure is to convert a string containing the character representation of a real number to its internal representation. The calling sequence is as follows:

SCL#SV_RV(sv,rv,status)

sv: This parameter specifies the string to be converted.

rv: This parameter specifies the name of a variable into which the real value is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 12.2.4 CONVERT REAL TO STRING - SCL#RV_SV

The purpose of this procedure is to convert the internal representation of a real value to its character string representation. The calling sequence is as follows:

SCL#RV_SV(rv,sv)

rv: This parameter specifies the real value to be converted.

sv: This parameter specifies the name of a variable into which the string is to be returned.

## 12.2.5 CONVERT FIXED STRING TO SCL STRING - SCL#FS_SV

The purpose of this procedure is to convert a fixed string to a string. Trailing blanks are truncated during the conversion. The calling sequence is as follows:

SCL#FS_SV(fs,sv)

fs: This parameter specifies the fixed string to be converted.

sv: This parameter specifies the name of a variable into which the string is to be returned.

## 12.2.6 CONVERT TOKEN TO INTEGER - SCL#TOK_IV

The purpose of this procedure is to convert a token to an integer value. If the token represents an integer constant the procedure simply returns the value contained in the IV field of the token. If the token represents a real constant the value contained in the RV field of the token is truncated and returned. If the token represents an LNS integer variable the value is obtained from the LNS and returned. If the token represents an LNS real variable the value is obtained from the LNS, truncated and then returned. Anything else will give rise to an error condition. The calling sequence is as follows:

SCL#TOK_IV(tok,iv,status)

tok: This parameter specifies the token to be converted.

iv: This parameter specifies the name of a variable into which the integer value is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

## 12.2.7 CONVERT TOKEN TO REAL - SCL#TOK_RV

The purpose of this procedure is to convert a token to a real value. If the token represents an integer constant the value contained in the IV field of the token will be converted to real and returned. If the token represents a real constant the procedure simply returns the value contained in the RV field of the token. If the token represents an LNS integer variable the value will be obtained from the LNS, converted to real and returned. If the token represents an LNS real variable the value will be obtained from the LNS and returned. Anything else will give rise to an error condition. The calling sequence is as follows:

SCL#TOK_RV(tok,rv,status)

tok: This parameter specifies the token to be converted.

rv: This parameter specifies the name of a variable into which the real value is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

## 12.2.8 CONVERT TOKEN TO STRING - SCL#TOK_SV

The purpose of this procedure is to convert a token to a string. If the token represents a string constant the procedure simply returns the value contained in the SV field of the token. If the token represents an LNS string variable the value will be obtained from the LNS converted to varying (with truncation of trailing blanks) and returned. Anything else will give rise to an error condition. The calling sequence is as follows:

SCL#TOK_SV(tok,sv,status)

tok: This parameter specifies the token to be converted.

sv: This parameter specifies the name of a variable into which the string is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

## 12.2.9 CONVERT TOKEN TO TYPE DESCRIPTION - SCL#TOK_TYPE

The purpose of this procedure is to construct a string of text describing the token which was passed to the procedure. This string is commonly placed in the parameter field of the status record for subsequent message generation. The calling sequence is as follows:

SCL#TOK_TYPE(tok, fs)

tok: This parameter specifies the token to be described. The description strings generated for the various token types are as follows:

| | |
|---|---|
| type_unknown ....... | "unknown value" |
| type_name .......... | "LNS type" |
| type_integer ....... | "integer value" |
| type_real .......... | "real value" |
| type_string ........ | "string value" |
| type_add ........... | "+" |
| type_sub ........... | "-" |
| type_mult .......... | "*" |
| type_div ........... | "/" |
| type_exp ........... | "**" |
| type_cat ........... | "CAT operator" |
| type_gt ............ | "GT operator" |
| type_ge ............ | "GE operator |
| type_lt ............ | "LT operator |
| type_le ............ | "LE operator |
| type_eq ............ | "EQ operator |
| type_ne ............ | "NE operator |
| type_and ........... | "AND operator |
| type_or ............ | "OR operator |
| type_not ........... | "NOT operator |
| type_assign ........ | "=" |
| type_open .......... | "(" |
| type_close ......... | ")" |
| type_comma ......... | "," |
| type_period ........ | "." |
| type_ellipsis ...... | ".." |
| type_colon ......... | ":" |
| type_semicolon ..... | ";" |
| type_foreign ....... | "foreign text" |
| type_end ........... | "end of text" |

fs: This parameter specifies the name of a fixed string variable into which the description is to be returned.

## 12.2.10 OUTPUT A VALUE - SCL#PUT_VAL

The purpose of this procedure is to output a value. The calling sequence is as follows:

SCL#PUT_VAL(stream,tok,status)

stream: This parameter specifies the IOC stream to which the value is to output.

tok: This parameter specifies the token whose value is to be output.

status: This parameter specifies the name of a variable into which the status is to be returned.

## 13.0 APPENDIX C ... INPUT/OUTPUT CONTROL (IOC)

The information in this appendix is internal maintainance level documentation and is included in this document to assist those responsible for writing command procedures in support of the system repertoire. The data structures defined are subject to change at the field level, however, the different of data structures defined will in all probability be those found in the final BTS.

The procedures described in this appendix define an input/output interface which allows users to logically concatenate input from multiple sources and distribute output to multiple destinations.

During job initiation the standard input file (JOB#INPUT) is opened and placed at the bottom of the input control stack. The standard print file (JOB#PRINT) is allocated and connected to the standard output and diagnostic streams.

## 13.1 DATA STRUCTURES

Knowledge of the following data structures is required to interface with IOC.

### 13.1.1 SCL STRING - SCL#STRING

The definition of SCL#STRING is as follows:

```
TYPE
  SCL#STRING = RECORD
    LHI: 1..256, " left hand index "
    RHI: 0..255, " right hand index "
    BUF: STRING(255) OF CHAR, " character buffer "
  RECEND;
```

LHI: This field contains the position of the 1st character of the string within the buffer. The position of the 1st character of the buffer is 1 and therefore if the string is left justified in the buffer LHI=1.

RHI: This field contains the position of the last character of the string within the buffer. The length of a string is defined to be RHI-LHI+1.

BUF: This field contains the characters comprising the string.

### 13.1.2 INPUT CONTROL STACK - IOC#INPUT

The definition of IOC#INPUT is as follows:

```
IOC#INPUT: [XDCL]
  RECORD
    TOP: INTEGER, " TOP OF STACK "
    FRAME: ARRAY [1..*] OF
      RECORD
        FCB: LNS#DESC, " FCB DESCRIPTOR "
        OD: DM#OD, " OPEN DESCRIPTOR "
        POS: INTEGER, " RELATIVE POSITION "
      RECEND,
  RECEND;
```

### 13.1.3 IOC STREAM CONNECTION TABLE - IOC#STREAM

The definition of IOC#STREAM is as follows:

```
IOC#STREAM: [XDCL]
  RECORD
    DEF: ARRAY [1..64] OF BOOLEAN, " DEFINITION MAP "
    CON: ARRAY [1..64, 1..16] OF BOOLEAN, " CONNECTION MAP "
    LIST: ARRAY [1..16] OF
      RECORD
        FCB: LNS#DESC, " FCB DESCRIPTOR "
        OD: DM#OD, " OPEN DESCRIPTOR "
        COUNT: 0..64, " CONNECTION COUNT "
      RECEND,
  RECEND;
```

DEF: This field contains a boolean map representing a pool of 64 stream ordinals. True indicates an assigned stream ordinal and false indicates a free stream ordinal. When a stream is declared, an IOC trap procedure is invoked by LNS. This procedure searches the definition map for a free stream ordinal and sets the ORD field of the stream descriptor to this number. The definition map is then adjusted to reflect the assignment. When a stream is removed, the IOC trap procedure is again invoked by LNS. The procedure then disconnects all files currently connected to the stream being removed and adjusts the definition map to reflect the release.

CON: This field contains a boolean map representing the connections currently established. The rows represent the 64 possible streams and the columns represent the 16 possible connections for a stream. True indicates an established connection, and false indicates the absence of a connection.

LIST: This field contains the files currently connected to streams. The LNS descriptor of the file control block and the open file descriptor are maintained for each file connected together with a count indicating the number of streams to which the file is currently connected.

### 13.1.4 IOC CHARACTER TRANSLATION TABLE - IOC#XLATE

The definition of IOC#XLATE is as follows:

```
IOC#XLATE: [XDCL] ARRAY [0..255] OF CHAR;
```

The default character translation set is the input character set.

### 13.1.5 IOC TAB TABLE - IOC#TAB

The definition of IOC#TAB is as follows:

```
IOC#TAB: [XDCL]
  RECORD
    TCHAR: CHAR, " TAB CHARACTER "
    TPOSN: ARRAY [1..16] OF 1..255, " TAB POSITIONS "
  RECEND;
```

The default tab character is circumflex and the default tab positions are [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80]. If the tab character is set to "space" no tabs will be acknowledged.

### 13.1.6 HEX ESCAPE CHARACTER - IOC#HEX

The definition of IOC#HEX is as follows:

```
IOC#HEX: [XDCL] CHAR;
```

The default hex escape character is reverse slant. If the hex escape character is set to "space" no hex escapes will be acknowledged.

### 13.1.7 OS STATUS - OS#STATUS

The definition of OS#STATUS is as follows:

```
TYPE
  OS#STATUS = RECORD
    LEVEL: 0..0FF(16), " general level indicator "
    FROM: STRING(2) OF CHAR, " issuing os section "
    ST_CODE: 0..0FFFF(16), " specific status code "
    MESG: STRING(32) OF CHAR, " message mask "
  RECEND;
```

LEVEL: This field contains the general status, the values of which are shown in the IPLOS structure overview document.

FROM: This field contains the operating system section that issued the status.

ST_CODE: This field contains the specific status code issued.

MESG: This field contains the message mask to be used by the system message generator when constructing diagnostic messages.

## 13.2 PROCEDURES

The following procedures may be called to obtain the services of IOC.

### 13.2.1 OPEN INPUT - IOC#OPEN

The purpose of this procedure is to open a file for input. The file specified will become the current input file for the job. When a file is opened by this procedure its LNS descriptor is added to the input control stack and the file is physically opened for input. The calling sequence is as follows:

```
IOC#OPEN (file, status)
```

file: This parameter specifies the LNS descriptor of the file to be opened. If the file specified does not exist an error condition will result.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 13.2.2 CLOSE INPUT - IOC#CLOSE

The purpose of this procedure is to close the current input file. When a file is closed by this procedure its LNS descriptor is removed from the input control stack and the file is physically closed. The calling sequence is as follows:

```
IOC#CLOSE (status)
```

status: This parameter specifies the name of a variable into which the status is to be returned.

NOTE: The standard input file cannot be closed by this procedure.

### 13.2.3 GET POSITION - IOC#GETPOS

The purpose of this procedure is to get the position of the current input file. The calling sequence is as follows:

IOC#GETPOS (pos, status)

pos: The pos parameter specifies the name of an integer variable into which the position is to be returned. The position returned is the position of the last record obtained from the file.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 13.2.4 SET POSITION - IOC#SETPOS

The purpose of this procedure is to set the position of the current input file. The calling sequence is as follows:

IOC#SETPOS (pos, status)

pos: The pos parameter specifies the position to which the file is to be set.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 13.2.5 GET FROM STANDARD INPUT - IOC#GETSTD

The purpose of this procedure is to get the next record from the standard input file. The calling sequence is as follows:

IOC#GETSTD (string, status)

string: The string parameter specifies the name of a string variable into which the record is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

NOTE: Each record obtained from this procedure is translated character by character according to the character translation table. In addition each record is scanned for tab and hex escape characters. Subsequent to these operations each record is written to IOC#INPUT.

### 13.2.6 GET FROM CURRENT INPUT - IOC#GET

The purpose of this procedure is to get the next record from the current input file. The calling sequence is as follows:

IOC#GET (string, status)

string: The string parameter specifies the name of a string variable into which the record is to be returned.

status: This parameter specifies the name of a variable into which the status is to be returned.

NOTE: Each record obtained from this procedure is written to IOC#ALTERNATE.

### 13.2.7 CONNECT OUTPUT - IOC#CON

The purpose of this procedure is to establish a connection between a file and a stream. When this procedure is called the map field of the stream connection table is altered and if necessary the LNS descriptor of the file specified is added to the table and the file is physically opened for output. The calling sequence is as follows:

IOC#CON (file, stream, status)

file: This parameter specifies the LNS descriptor of the file to be connected. If the file specified does not exist an error condition will result.

stream: This parameter specifies the LNS descriptor of the stream to be connected.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 13.2.8 DISCONNECT OUTPUT - IOC#DISCON

The purpose of this procedure is to sever the connection between a file and a stream. When this procedure is called the map field of the stream connection table is altered and if necessary the LNS descriptor is removed from the table and the file is physically closed. The calling sequence is as follows:

IOC#DISCON (file, stream, status)

file: This parameter specifies the LNS descriptor of the file to be disconnected.

stream: This parameter specifies the LNS descriptor of the stream to be disconnected.

status: This parameter specifies the name of a variable into which the status is to be returned.

### 13.2.9 PUT TO STREAM - IOC#PUT

The purpose of this procedure is to output a record. When a record is output by this procedure it is written on all files connected to the stream specified. The calling sequence is as follows:

IOC#PUT (stream, string, status)

stream: This parameter specifies the LNS descriptor of the stream to which the record is to be output.

string: The string parameter specifies the string of text to be output.

status: This parameter specifies the name of a variable into which the status is to be returned.

## 14.0 APPENDIX D ... MESSAGE GENERATOR

There will be a facility for the formatting and output of system messages from the IPL Operating System.

The general requirements of such a system are roughly as follows:

- There shall be a consistent method of calling for message output from the system.
- As far as possible all messages will be accessible from a central message file.
- There should be a consistent format for messages. This will have a side benefit in that users will grow accustomed to a consistent system of reporting.
- The messages produced by the system should be amenable to analysis and data gathering by automatic means.

### 14.0.1 FUNCTIONAL BREAKDOWN.

The actual message system splits into two fairly distinct functional areas as follows.

- Message Generator. Provides the basis of operating system response to the user.
- Message File Update. Provides a means of adding new status codes and message control strings to the system.

## 14.1 MESSAGE GENERATOR

This program provides the basis for issuing messages to the user from the operating system. It is a fairly simple program both in conception and use. Its parameters are a stream descriptor specifying the stream to which output is to be sent and an OS#STATUS record as defined below.

The message generator maintains an indexed sequential file of character strings called Message Control Strings. These are used in conjunction with the Message Mask supplied in an OS#STATUS record to format the output message. The message mask supplied in the OS#STATUS record contains the variable portions of the output message and the message control string from the message file contains the fixed portions. The first character of both the message mask and the message control string represent control characters. The control character in the message mask is used as an indicator to separate the variable items which are to be inserted into the output message. The control character in the message control string is used to indicate where the variable items from the message mask are to be inserted in the output message.

### 14.1.1 MESSAGE GENERATOR CALLING METHOD.

The call upon the message generator passes two parameters, namely the descriptor of the stream to which output is to be sent and an OS#STATUS record as defined below. The section code and the specific status key from the OS#STATUS record are used to look up an indexed sequential file to obtain a message control string. The level indicator in the status record plays no part in the lookup.

The caller of the message generator will be responsible for deciding which streams the message should be output to, since it can only be the caller who is aware of the significance of the status.

The macro call format is as follows.

MG#REPORT( stream , status )

stream : this parameter is the descriptor of the stream to which the message is to be sent.

status : this parameter is an OS#STATUS record as defined below.

## 14.1.2 CONTROL CHARACTERS.

As previously mentioned, the first character of the message control string should be a delimiter, which will be subsequently used to indicate control information. The control character and the character immediately following it represent different actions that the message generator can perform with the message mask. The possible control sequences currently available are as follows. ( + will represent the control character ).

+P Take the next delimited sequence from the message mask in the status record and insert it into the output text.

+N Insert a newline character into the output text at this point.

## 14.1.3 CONTROL OF OUTPUT DETAIL LEVEL.

When the message generator is called, there can be a certain degree of control over the amount of detail supplied in the message. This can be controlled by setting the LNS variable LNS#LOCAL->MG#HEAD to a particular value. There are at present three possible settings of the variable giving three types of output in varying amounts of detail as follows.

MG#HEAD = 1 : The message will be output in "plain English". The message mask will be formatted

according to the message control string in the message file. This mode of the message generator will be considered the normal; any setting of MG#HEAD which is outside the permissible range of settings will cause the head style to be defaulted to head style 1.

MG#HEAD = 2 : output only the status code and the message mask. The message generator will not look for the message control string to perform any replacement of the message mask.

MG#HEAD = 3 : This setting of the switch will output the most detail, consisting of the OS section name, accepted/rejected state and the message itself.

Example : Suppose the OS#STATUS record contains the following information.

status code = 8LN0205
message mask = '*VERMOUTH*MARTINI*'

Message generator will use the key LN0205 to look up the indexed sequential file of message control strings. Suppose the message control string corresponding to the key LN0205 is :

message control string = '+ENTRY +P NOT FOUND IN SEGMENT +P'.

MG#HEAD = 1 will output :

```
8LN0205 ENTRY VERMOUTH NOT FOUND IN SEGMENT MARTINI
```

MG#HEAD = 2 will output :

```
8LN0205 *VERMOUTH*MARTINI*
```

MG#HEAD = 3 will output :

```
REQUEST REJECTED DUE TO USER PROBLEM.
ERROR 0205 DETECTED BY LNS MANAGER.
ENTRY VERMOUTH NOT FOUND IN SEGMENT MARTINI
```

## 14.2 MESSAGE FILE UPDATE

This program will accept input consisting of a status code and the corresponding control string. It will make checks on the prior existence of that code and will enter or update the new data into the indexed sequential file of message control strings for use by the message generator.

To assist in maintaining the file of message control strings the message generator has the following commands. Note that the use of these commands will be restricted to systems personnel and will not be available to the casual user.

### 14.2.1 MG#EDIT

The purpose of this command is to edit the file of message control strings, either to insert new entries or to change or delete old entries. The format of the command is as follows.

```
MG#EDIT   section=<string>   [code=<integer>]   [message=<str
        [new]  [old]  [delete]  [status=<ref name>]
```

section ! s : This parameter is the abbreviation for the OS section name which is responsible for generating the specific status concerned.

code ! c : This parameter is an integer representing the actual status code.

message ! m : This parameter is a character string representing the message control string for that particular status code.

new ! old ! delete : This parameter specifies the current status of the entry in the message control file. Quoting "new" indicates that this is a new entry; "old" indicates that this entry will replace an already existing entry; "delete" indicates that an existing entry is to be deleted. If "delete" is quoted then the 'message' parameter in the command is redundant and need not be specified.

status : This parameter represents an LNS variable into which the status of the request will be returned on completion of the command. If the 'status' parameter is not specified then the SCL error handler will be invoked upon detection of any error condition.

Example of use :

```
    MG#EDIT   section = 'LN'    code = 0205     new.....
              message = '+ENTRY +P NOT FOUND IN SEGMENT +P'
```

Notes on the MG#EDIT Command.

1. Note that all entries in a particular section of the message control file may be deleted by typing the command in the form :

```
MG#EDIT    section='XX'   delete
```

2. Also the whole message control file may be deleted by simply typing the command in the form :

```
MG#EDIT   delete
```

Extreme care should be exercised in using these variants of the command.

Also see the MG#SECT command below, where there is a relationship between the entries in the MG#NAMELIST and the entries in the message control file.

### 14.2.2 MG#SECT

The purpose of this command is to associate a section abbreviation with the full name of the section, for example 'LN' is associated with 'LNS MANAGER'. The format of the command is as follows.

```
MG#SECT section=<string> name=<string> [new] [old] [delete]
```

section ! s : This parameter is the abbreviation for the OS section.

name ! n : This parameter is a string representing the full name of the OS section concerned.

new ! old ! delete : This parameter indicates the current status of the entry in question. Quoting "new" indicates that this is a new entry; "old" indicates that this is a replacement of an existing entry; "delete" indicates that the specified entry is to be deleted. If "delete" is quoted the "name" parameter is redundant and need not be quoted.

Example of use :

```
MG#SECT   section='LN'  name='LNS MANAGER'  new
```

Notes on the command.

1. The MG#SECT command can be typed in the following manner :

```
MG#SECT    section='XX'   delete
```

If the command is typed in this way then as well as deleting the entry in the namelist table, all entries belonging to that OS section in the message control file will also be deleted. Care should be exercised in the use of this variant of the command.

### 14.2.3 MG#DISPLAY

The purpose of this command is to provide listings of the entries currently on the message control file. All entries in the file may be listed, or just those appertaining to a specific section of OS. The format of the command is as follows.

```
MG#DISPLAY    [section=<string>]    [code=<integer>]
```

section ! s : This parameter is the abbreviation for the particular OS section if only the codes for that section are required to be listed.

code ! c : This parameter is quoted when the data relating to a specific code is required. If this parameter is quoted then the 'section' parameter must also be quoted.

Note that if the MG#DISPLAY command is typed with no parameters at all then all the entries in the file will be listed.

Example of use :

```
MG#DISPLAY.    section='LN'
```

## 14.3 DATA STRUCTURES

### 14.3.1 OS STATUS - OS#STATUS

The definition of OS#STATUS is as follows:

```
TYPE
  OS#STATUS = RECORD
    LEVEL: 0..0FF(16), " general level indicator "
    FROM: STRING(2) OF CHAR, " issuing os section "
    ST_CODE: 0..0FFFF(16), " specific status code "
    MESG: STRING(32) OF CHAR, " message mask ".
  RECEND;
```

LEVEL: This field contains the general status, the values of which are shown in the IPLOS structure overview document.

FROM: This field contains the operating system section that issued the status.

ST_CODE: This field contains the specific status code issued.

MESG: This field contains the message mask to be used by the system message generator when constructing diagnostic messages.

### 14.3.2 OS SECTION NAME LIST.

The message generator system will require a list of OS section names to be maintained in LNS#GLOBAL in order that the section abbreviation may be converted to the appropriate character string when the full section name is required. The SWL type definition of this structure is as follows.

```
MG#NAMELIST =
  ARRAY[ 1 .. * ] OF
    RECORD
      SECTION : STRING( 2 ) OF CHAR , "  OS Section Mnemonic
      NAME    : STRING( 32 ) OF CHAR , "  OS Section Name  "
    RECEND ;
```