X-408-77-100
PREPRINT

TM-79541

GSFC SYSTEMS TEST AND OPERATION LANGUAGE (STOL) FUNCTIONAL REQUIREMENTS AND LANGUAGE DESCRIPTION

FEBRUARY 1978

GSFC

GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

(NASA-TM-79541) GSFC SYSTEMS TEST AND OPERATION LANGUAGE (STOL) FUNCTIONAL REQUIREMENTS AND LANGUAGE DESCRIPTION (NASA) 44 p HC A03/MF A01 CSCL 09B

N78-22777

Unclas
G3/61 16351

For information concerning availability
of this document contact:

Technical Information & Administrative Support Division
Code 250
Goddard Space Flight Center
Greenbelt, Maryland 20771

(Telephone 301-982-4488)

NASA/X-408-77-100

GSFC SYSTEMS TEST AND OPERATION LANGUAGE (STOL)

FUNCTIONAL REQUIREMENTS AND

LANGUAGE DESCRIPTION

Prepared for

GODDARD SPACE FLIGHT CENTER

By

COMPUTER SCIENCES CORPORATION

Under

Contract NAS 5-24300
Task Assignment 115

PREFACE

This document presents the functional requirements and description of the Systems Test and Operation Language (STOL) to be initially baselined at Goddard Space Flight Center (GSFC). STOL represents the synthesis of several independent language developments at GSFC, notably the Procedure Control Language (PCL) family, the Orbiting Solar Observatory/Atmosphere Explorer (OSO/AE) language family and the Applications Technology Satellite/High Energy Astronomy Observatory (ATS/HEAO) language family. By combining the best features of these languages with a mutually agreed upon syntax, the STOL designers have produced a simple basic language to provide the basis for testing and control of payload ground systems in the 1980s.

STOL is intended to be a standard language for controlling GSFC payload integration, test, and operations systems. To achieve this objective, STOL requirements represent the distilled experience of the technical personnel who were most responsible for, and familiar with, the predecessor languages and their strengths and weaknesses.

The success of STOL as a standard depends directly upon its reliance on the essential requirements of the predecessor languages and their representation as a small, readily understandable language nucleus. The nucleus is designed to be standardized and controlled on a GSFC-wide basis while making it easy for each individual system to meet its unique requirements by adding its own locally controlled extensions to STOL.

The requirement that STOL be a standard nucleus allowing local extensions overrides all other requirements. The power of language to interface people to complex operational systems resides fundamentally in this combination of standardization and flexibility. This essential duality must not be compromised away for the sake of expediency in achieving other language design or control objectives.

The following GSFC personnel participated in the STOL design effort: Richard desJardins (510), Gardiner Hall (511), James McGuire (420), Phillip Merwarth (582), William Mocarsky (408), Walter Truszkowski (514), and Anthony Villasenor (734).

The following contractor personnel assisted the design effort: Fred Brosi (CSC), Preston Burch (OAO Corporation), David Carey (RCA), David Carey (Westinghouse), Morris Gunzburg (OAO Corporation), William Havener (CSC), Joseph McCann (RCA), John Nieberding (Westinghouse), Timothy Swanson (CSC), and Fred Zussman (ORI).

The following GSFC management personnel also closely followed and supported the STOL design activity: Robert Bartlett (408), Charles Fuechsel (401), Jerold Hahn (511), Thomas Huber (730), Louis Koschmeder (734), and Ann Merwarth (408).

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

SECTION 1 - INTRODUCTION

The Systems Test and Operation Language (STOL) provides the means for user communication with payloads, applications programs, and other ground system elements. It is a systems operation language that enables an operator or user to communicate a command to a computer system. The system interprets each high-level language directive from the user and performs the indicated action, such as executing a program, printing out a snapshot, or sending a payload command.

By using STOL, payload test and operations personnel may be relieved of repetitive tasks while ensuring that recurring, fixed sequences of operations are always performed in exactly the same order, and guaranteeing repeatability of test procedures or Project Operations Control Center (POCC) operations. However, regardless of the level of automation achieved through use of STOL, human control over all system activities is maintained, both through user definition of the STOL procedures and through user manual override control of the system during execution of a STOL procedure.

The individual statement or line of STOL code entered by a user when using STOL is called a directive. The STOL directives entered by users are checked for correct syntax and are processed by a STOL language processor. This processor interprets the directives and creates messages or data segments to be passed on to the applications programs handling payload commanding, telemetry processing, and displays. This document describes the initial STOL to be baselined under the cognizance of the GSFC STOL Configuration Control Board.

Section 2 describes required language features and processor implementation considerations. In Section 3, basic capabilities are outlined. Sections 4, 5,

and 6 present the telemetry, command, and input/output directives, respectively. Section 7 outlines procedure definition and control. In Section 8, listing, extension, and STOL nucleus capabilities are discussed. Appendix A presents the shortened form and reference page number of each directive. The Discrepancy Report/Engineering Change Proposal Form is given in Appendix B.

SECTION 2 - STOL FEATURES AND IMPLEMENTATION CONSIDERATIONS

The required language features and processor implementation considerations are discussed below.

2.1 REQUIRED LANGUAGE FEATURES

The features listed below represent the essential generic characteristics of the language. The language should

- Allow the user extensions to handle unique requirements while remaining small and simple
- Provide for a high degree of automatability
- Allow both online interactive use and offline preparation of predefined sequences of directives (procedures)
- Always allow for manual control override
- Provide for structured programming constructs which aid in achieving reliability and maintainability of procedures
- Permit abbreviation of keywords by interactive users to minimize keystrokes
- Allow comment option for each statement
- Provide arithmetic and logic capability
- Provide for combination of statements into procedures with parameter and string substitution capability
- Remain executable in an interpretive mode (i.e., declarations are considered unnecessary so that statements execute independently)

2.2 STOL LANGUAGE PROCESSOR IMPLEMENTATION CONSIDERATIONS

The features listed below are considered desirable for most systems. In some systems, however, it may not always be feasible or desirable to provide all of these options.

- The language processor should provide run-time visibility. The directive currently in execution must be displayed, and the next one in sequence should be displayed if possible
- The language processor should provide run-time traceability. A log should be produced showing each directive executed together with the time of execution
- A mechanism should be provided that allows the user to escape to the host computer's operating system
- The processor should allow several users to enter directives or to execute predefined procedures simultaneously, subject to protection features. In particular, payload commanding should be restricted to one physical terminal device at a time
- Implementors of payload data base languages for displays and other functions should ascertain that their language forms are also compatible with STOL in the interest of simplicity and uniformity. For example, the method of describing a display in the data base definition language should be compatible with the means provided in STOL for defining displays
- Implementors of payload data bases should ensure that run-time modifications of the data base by STOL procedures affect only a run-time temporary copy rather than the officially controlled permanent copy

SECTION 3 - BASIC CAPABILITIES

Syntax and language elements, arithmetic and logical capabilities, starting, linking to, and stopping programs, and binding resources are described in this section.

3.1 STANDARD SYNTAX AND LANGUAGE ELEMENTS

3.1.1 Character Set

The character set is the seven-bit ASCII character set. Cards punched on an IBM 029 keypunch are accepted as input.

3.1.2 Constants

Integers may be represented as decimal, binary, octal, or hexadecimal numbers as shown respectively in the examples below.

```
37, -1, 2483, 0
B'100101', B'1', B'0'
O'45'
X'20', X'0'
```

Real (floating point) numbers are represented with a decimal point, either with or without an exponent of 10, such as

```
1.0, -879.5, 0.0, 2.25E03, 3.6E-01
```

Character strings are enclosed by single quotation marks:

```
'S/C ATT', 'OFF', 'CAN' 'T MEANS WON' 'T'
```

3.1.3 Variables

STOL allows the user to refer to two distinct classes of variables. One class is the set of system global variables that constitute the operational data of the system on which the STOL processor is executing.

Telemetry data, real-time derived parameters, system configuration flags, and parameters are examples of system global data. These variables are referred to in STOL by their system global names.

Global variable names must begin with a letter (A through Z) and must contain eight or fewer characters that are either letters or numbers. Arrays of variables with up to two dimensions are permitted. Variables must be either integer or real.

The values of the system global variables come from outside STOL. They reside in system common memory and are available to all users, including STOL, subject to protection.

The other class of variables to be referenced in STOL are the procedure local variables. These local variables exist (i.e., have a value) only when the procedure is open for execution (i.e., during execution, waiting, or stacked on a nest). The local variables are used for computations within procedures or for passing arguments between procedures of a single user.

The local variables are also of either real or integer type. They have fixed names for simplicity and are typed according to usage. These names are as follows:

X1, X2, ..., Xn, where $n \geq 16$

3.1.4 Directives

The syntax of STOL directives is described below.

- Fields--The basic STOL statement is made up of four separate and distinct fields: label, directive, argument, and comment. The fields are order dependent, but their character position is format free

- Label--The label is a 1- through 8-character alphanumeric field that is followed by a colon. The first character of the label must be alphabetic. The label field is optional

- Directive--The directive field contains either an alphanumeric mnemonic identifier or, in the case of command only, a special character (slash) followed by a possibly null alphanumeric mnemonic identifier. The directive field is terminated by one or more blanks unless the alphanumeric part is null. In the absence of a label field, the directive field becomes the first field in a statement. If the directive field is not recognized as a system name, it is assumed to be the name of an application program which is to be run

- Argument--The argument field is a character string that is passed to the invoked program when requested. It is terminated by a semicolon or end of line. For uniformity, arguments within an argument field should obey the standard syntax of STOL language elements when applicable and should be separated by commas or blanks. The argument field may not begin with an equal sign; this avoids ambiguity between directive names and variable names in statements such as 'KILL = X1'

- Comment--The comment field consists of a string of characters preceded by the special character semicolon (;)

- Additional definition--The first nonblank character defines the start of a statement. A line of all blanks is considered a comment

- Continued line--The occurrence of two successive semicolons (;;) defines a continued line, i.e., the next line is concatenated to the current line at the position indicated by the first semicolon

- Character strings--Pairs of single quote marks are optionally used to set aside character strings within an argument field

- Short forms--Short mnemonic forms are specified for most directives (e.g., TE for TERM). Such short forms must be recognized in addition to, rather than instead of, the standard mnemonics

3.2 ARITHMETIC AND LOGICAL CAPABILITIES

3.2.1 Expressions

The arithmetic and logical expressions and capabilities adhere to the following set of operations on simple integer and real variables:

+, -, *, /, **

.EQ., .NE., .GT., .LT., .GE., .LE.

.NOT., .AND., .OR., .XOR.

3.2.2 Mixed Mode Conversions

Implicit mixed mode conversions are performed by the STOL processor. For logical operations, the integer value 0 is interpreted to be .FALSE., whereas all nonzero values are interpreted to be.TRUE.

The logical value ".FALSE." is stored as the integer 0 (i.e., all 0s). The logical value ".TRUE." is stored as the 1's complement of the integer 0 (i.e., all 1s).

3.2.3 Arithmetic Assignment

All arithmetic statements assigning values to variables use the directive LET, followed by a variable name, an equal sign, and an arithmetic expression. STOL implementors are encouraged to permit the directive mnemonic "LET" to be optionally omitted by the user. Some examples of valid assignment statements are the following:

```
        LET X1 = X2
TEST:   LET X1 = 2.5 + X2
        TLMIDX = 0              ; omitting LET allowed but not required
```

3.3 STARTING, LINKING TO, AND STOPPING APPLICATIONS PROGRAMS

All applications programs are initiated by the use of the RUN mnemonic. The RUN mnemonic may be (and is normally) omitted; that is, the short form of RUN is simple null (no characters). If an application program is to be intervaled, i.e., run every multiple n of a system-unique fundamental interval (such as a telemetry minor frame), the directive INTERVAL is used. A running program is terminated by using the TERM mnemonic (short form, TE)

```
RUN programname argumentstring
TERM programname
INTERVAL programname,n argumentstring
programname argumentstring          ; short form for RUN
```

The label and comment fields are optional. The following forms exemplify running application program PROG at a given interval (e.g., every fourth minor frame):

```
INTERVAL PROG,4
IN PROG, 4              ; short form
```

Applications programs may run either synchronously or asynchronously. In synchronous operation, an application program called by a STOL procedure preempts execution of the procedure until execution of the application program is complete; control of the system then returns to the STOL procedure. In asynchronous operation, the invoked application executes in parallel with the invoking procedure. The design of the invoked application program determines whether it will run synchronously or asynchronously.

3.4 BINDING RESOURCES

Real resources (data set names, physical devices) are bound at run time to logical resources (data definition names, logical port names) by the use of the ASSIGN directive. The general form is

```
ASSIGN logicalname,realname
AS logicalname,realname                    ; short form
```

which is optionally followed by a second comma and additional specification information as required. The real name may in fact be a logical reference to some data set or other assignable name. This form is translated by the STOL processor into the native language of the underlying operating system. Both the realname and logicalname may be any legal text string accepted by the underlying system, including the usual slashes or periods used as hierarchical data set extenders, e.g.,

```
ASSIGN HISTAPE,MT1        ; assign history tape to mag tape unit 1
```

Standard resource names such as MT1 will be recommended.

SECTION 4 - TELEMETRY DIRECTIVES

Telemetry points are referred to by their system global variable names. These names generally fall into two general types as defined below. Any name meaningful to the underlying system is acceptable to STOL because STOL simply passes the name to the underlying system for interpretation. (In some implementations, this is effectively done at system assembly time by building a symbol table for STOL processor use.)

A mnemonic name for a telemetry variable must satisfy the standards for variable names and is considered another system global variable. The data point represented by such name could be telemetry, pseudotelemetry, or a derived real-valued parameter.

Telemetry points may also be specified by their matrix position in one of a number of possible matrix formats, depending on what is supported by the system. One possibility is the representation through the actual matrix position in current telemetry format:

TM(I, J), where $0 \leq I, J \leq 127$

This form of representation refers to the telemetry word in the Ith position in the minor frame and Jth position in the major frame (Jth subcom) within the current telemetry stream. Telemetry points can also be specified by their matrix position in mission format:

TF(I, J) → TM(I', J')

This form of representation refers to the telemetry word position in the mission ROM or mission format. If the current format is not the mission format, the system maps the I, J position into the current actual matrix position I', J' in the current format. Specific telemetry capabilities are described below.

4.1 ACQUIRE DIRECTIVE

ACQUIRE is the directive that initiates input of the type of data selected, such as primary realtime, secondary realtime, hardwire sensor, or tape recorder data. Either realtime or previously recorded (i.e., playback) data from a history file can be acquired using this directive. The argument string can include a device operational label, or device control information. The default for ACQUIRE is a system-unique device and data type. Examples of this directive appear below.

```
ACQUIRE ON, type, parameters        ; types include hardwire, PCM,
                                    ;  tape, playback

ACQUIRE OFF, type, parameters

AC                                  ; short form; ON and system
                                    ;  defaults are assumed
```

4.2 LIMITS DIRECTIVE

LIMITS is the directive that defines and controls the limit checking of data values in the system. The data may be telemetry, pseudotelemetry, or derived parameters. Examples are given below.

```
LIMITS ON                           ; turns all limits on

LIMITS ON, TM(20,34)                ; turns limits on for specified
                                    ;  data item

LIMITS OFF, TM(20,34), B1VOLTS      ; turns off several limits

LIM                                 ; short form; ON is as-
                                    ;  sumed
```

LIMITS ON initiates limit checking of specified data values. The directive assumes that limits have been previously defined and exist within the system. The three allowable entry formats permit limit checking to be enabled for all

data values or for a data value specified by matrix or mnemonic identifier. LIMITS OFF terminates limit checking of specified data values; the options are exactly the same as for LIMITS ON.

LIMITS DEF defines limit conditions for data values in the system, including telemetry, pseudotelemetry, and derived parameters. The parameter formats are system unique although recommended standard formats will be developed. These established limits are maintained within the run-time system. The two allowable entry formats permit limits to be loaded by either matrix or mnemonic identifier. The parameter string might contain limits, masks and values, and program calls. Examples appear below.

```
LIMITS DEF, B1VOLTS, parameters
LIMITS DEF, TM(20,34), parameters
```

4.3 CONVERT DIRECTIVE

CONVERT is the directive that defines engineering conversions for data values in the system, including telemetry and pseudotelemetry. The data named may be any allowable telemetry identifier. The type indicates the type of conversion, either polynomial or piecewise straight lines, to be performed. The parameter string contains the constant coefficients to be used in the conversion. The parameter formats are system unique although recommended standard formats will be developed. Examples are given below.

```
CONVERT B1VOLTS, POLY, parameters
CONVERT TM(20,34), STRAIGHT, parameters
CON TM(20,34), STRAIGHT, parameters
```

SECTION 5 - COMMAND DIRECTIVES

This section defines basic directives for commanding the payload. It is assumed that every installation requires a number of different command modes (e.g., one-stage commanding, two-stage commanding, verification through the command counter in telemetry, verification through functional data in telemetry). All command directives begin with a slash.

5.1 MANUAL COMMANDING

The /CMD directive is used to invoke the command processor with a single command. The command processor may transmit the command directly to the payload, insert the command into a command buffer, or take some other action, depending upon its current mode. The short form of the directive is a slash (/). Various examples of its usage are given below.

```
/CMD cmdfield          ; general form
/CMD VAM, ON           ; use of command mnemonics allowed
/VAM, ON               ; pulse command, short form
/3, 16                 ; pulse command RIU 3, LINE 16 for MMS
/CU, RATE16            ; serial magnitude command, short form
/3, 71, X'1111'        ; serial magnitude command RIU 3, NRZ = 001
                       ;  data = hexadecimal 1111 for MMS
```

The cmdfield may be any text string recognized by the system as identifying a particular command. It may be a command mnemonic or other representation meaningful to the system.

5.2 COMMAND MODE SETTING

The mode for commanding the spacecraft (no verify, verify through telemetry, etc.) is set via the MODE directive. Command modes are system unique. Various forms of the MODE directive appear below.

```
/MODE modeID          ; general form
/MODE VERIFY          ; e.g., verification of commands through com-
                      ;  mand counter
/MODE 2               ; e.g., two-stage commanding
/M 2                  ; shortened form
```

5.3 OBC COMMANDING

Onboard Computers (OBCs) are treated as subsystems of the spacecraft. Commands to load, to dump, to reset the OBC, or to perform similar functions, are of the form

```
/OBC, parameters
```

The capability to patch memory, to load, to dump, to send OBC executive requests, and to perform similar functions falls into this format.

5.4 GROUND SYSTEM-GENERATED COMMAND LOADING (SPACECRAFT LOAD)

The /LOAD directive is provided to enable the system operator to uplink command and data loads to the payload. These loads may have been defined by the user or generated by the Command Management System. Examples are as follows:

```
/LOAD loadname        ; general form
/LOAD SNT1003         ; e.g., command memory load for SNT 1003
/LOAD FILE1           ; load predefined file
/L FILE1              ; shortened form
```

This directive is primarily used to load computers, microprocessors, and command memories. It is assumed that the loadname resides on CLFILE, which has a default assignment in the system or may be reassigned by using the ASSIGN statement. The format of the actual load may include the address and load vectors for the computer or microprocessor.

5.5 PAYLOAD COMMAND, GROUP, AND LOAD TRANSMITTAL

In two-stage commanding, the effect of /CMD and /LOAD is to load the command buffer in the system. The second stage of commanding is to transmit the contents of the buffer to the payload via the /SEND directive as shown below.

```
/SEND          ; general form
/S             ; short form
```

On a system-unique basis, use of the command buffer is avoidable through definition of an appropriate one-stage command mode controlled by the /MODE directive.

5.6 CRITICAL COMMAND CONTROL

A small number of commands in the system may be designated as "critical". This means that to send the command, a separate manual intervention step is required at the time the command is loaded into the system buffer.

/ALLOW instructs the system to permit the loading of a critical payload command or group into the command buffer. /CANCEL has the effect of erasing the current system request for loading that particular command into the buffer. It is assumed that a system-unique interactive signal alerts the operator that the system has blocked on a critical command. These directives are shown below.

```
/ALLOW         ; general form
/A             ; short form
/CANCEL        ; general form
/X             ; short form
```

5.7 COMMAND BUFFER CLEAR

The /CLEAR directive completely clears the command buffer at the request of the user as shown below.

```
/CLEAR          ; general form
/Z              ; short form, zeroes the buffer
```

5.8 COMMAND RETRANSMISSION

The capability exists for the operator to manually request retransmission of any commands in the command buffer that have failed verification, in the following manner:

```
/RETRY          ; general form
/R              ; short form
```

SECTION 6 - INPUT/OUTPUT DIRECTIVES

6.1 PAGE AND SNAP DIRECTIVES

These directives select a predefined display page for viewing on a cathode ray tube (CRT) screen or for a snapshot on a printer. The display image can be created via procedure statements (refer to Section 6.2) and stored beforehand in a data base or generated via a special program called into memory for execution. P and SN are the respective short forms for PAGE and SNAP.

Each implementation has a default system-unique fundamental interval at which display pages are nominally updated. In addition, implementors are encouraged to allow intervaling at other rates via an additional parameter. Example forms are shown below.

```
PAGE formatname, devicename      ; devicename is optional
PAGE POWER1, KC4                 ; display of format POWER1 on
                                 ;  device KC4
SNAP ACS2, LP                    ; snapshot of ACS2 to device LP
PAGE ACS2                        ; display of ACS2 on local device
PAGE CLEAR, KC4                  ; clearing of device KC4
PAGE ACS2, OFF                   ; turning off (freezing) of specified
                                   display
SNAP devicename                  ; making of a hardcopy of specified
                                 ;  device
```

All device names, format names, and the fundamental interval are system unique. Formatname CLEAR is reserved on all systems for clearing CRT screens.

6.2 FORMAT DIRECTIVE

This directive defines a display format (layout of fixed text and variable fields) as follows:

```
FORMAT formatname, arguments        ; general form

FORMAT 2, 3, 5, 'POWER SUBSYSTEMS'

FORMAT ACS2, 4, 20, TM(20,34), ...

FORMAT ACS2, CLEAR          ; clearing of FORMAT ACS2; erasing of
                            ;  old format
```

The first example of a FORMAT statement given above defines the placing of the text string 'POWER SUBSYSTEMS' on page 2, line 3, column 5. The second statement indicates that the current telemetry value TM(20, 34) is to go on page ACS2, line 4, column 20; additional specifiers might give the field width and conversion format.

The allowable arguments that specify a particular formatname are system unique. However, a standard set of formats will be developed. The short form for FORMAT is FO.

6.3 HISTORY DIRECTIVE

This directive generates a history file of ongoing operations such as telemetry or attitude as specified. The logical device HISTFILE is assigned to a default device or can be reassigned using the ASSIGN directive. Additional system-unique options may be included as additional arguments. This directive is also used to record telemetry data for later playback. The various uses of this directive are shown below.

```
HISTORY ON, datatypes       ; general form

HI                          ; short form, telemetry is default
                            ; datatype

HISTORY OFF, datatypes
```

6.4 LOG DIRECTIVE

This directive provides documentation of procedure execution annotated with Greenwich Mean Time (GMT). All statements executed are written to the system-defined LOGFILE data set, which provides backup capability even in the event of printer failure. Commands are always written to this data set (as a kind of flight recorder), together with bit pattern, even if LOG OFF is specified. The LOGFILE data set is defaulted or assigned. No short form is required. Examples are as follows:

```
LOG ON, type, device        ; general form
LOG OFF                     ; disabling of logging
LOG parameters              ; ON assumed
```

6.5 CHART DIRECTIVE

This directive assigns specified telemetry and hardwire data to analog and/or event stripchart recorder pen numbers as follows:

```
CHART ON, 8, B1VOLTS        ; assigning of B1VOLTS to pen 8
CHART 9, TM(20,34)          ; ON assumed
CH OFF, parameters          ; short form
```

SECTION 7 - PROCEDURE DEFINITION AND CONTROL

The directives described in this section are used to define new STOL procedures, to modify old ones, and to control the execution of these procedures. It is assumed that each procedure has a name and that all procedures defined on a given system are stored in a system STOL procedure library and are accessible by name.

7.1 PROCEDURE DEFINITION

7.1.1 PROC and ENDPROC Directives

The PROC and ENDPROC directives are respectively used as header and trailer directives to enclose a procedure definition. The name of the procedure and the number of parameters to be passed to it are specified in the PROC statement. The directives are given below.

```
PROC SC(n)      ; beginning of the definition of a procedure
                ;  named 'SC' which receives n parameters when
                ;  it is invoked

ENDPROC         ; end of the definition of SC
```

The integer n specifies the number of parameters to be passed to the procedure. If no parameters are to be passed (a typical occurrence), the parentheses may be omitted. The parameters are passed as specified in subsequent sections.

7.1.1.1 Parameter Passing by Text Substitution

This type of parameter passing involves passing a character text string to a procedure as shown below. The receiving procedure references the text string by $n, where n is the nth argument in the calling sequence. This capability is used to pass the names of displays and global variable references. This type

of call is known as "call by name" in compiler theory; it is the mechanism used in macro expansion. The following is an example:

```
PROC SC(3)          ; definition of procedure SC with three parameters
.
.
PAGE $1, $3
.
.
WAIT $2
```

A sample calling sequence for the above defined procedure is

```
LET WTIME = 10
START SC (POWER1, WTIME, KC4)
```

This calling sequence would expand into

```
PROC SC(3)
.
.
PAGE POWER1, KC4      ; $1, $3 replaced
.
.
WAIT WTIME            ; $2 replaced with WTIME (global variable)
```

7.1.1.2 Parameter Passing by Local Variable Reference

When the text string given as one of the parameters to be passed has the form Xn, it is interpreted as a local variable reference rather than as a text string to be passed. For example, nothing is accomplished by passing 'X12' as a text string, because the name 'X12' refers to a different variable in the calling procedure from that in the called procedure. Instead, this type of parameter is used to refer to the address of the local variable in the calling procedure. This type of call is known as "call by reference" in compiler theory; it is the mechanism used in a FORTRAN subroutine call.

7.1.1.3 Parameter Pass Through

Both types of parameter passing allow pass through to several levels of nesting as in the following example:

```
PROC PROC4(1)              ; definition of PROC4 with one parameter
.
.
START PROC5($1, ...)       ; passing of the argument provided in any
                           ;  PROC4 call on through to the nested
                           ;  PROC5
.
.
ENDPROC
```

A minimum of three levels of nesting procedures is required. Implementors are cautioned that three levels of nesting might also represent a good maximum number as keeping track of deeper nesting is very difficult for a user.

7.1.2 EDIT Directive

This directive invokes the STOL editor for creation or modification of a STOL procedure as indicated below. The STOL editor is assumed to be interactive and system unique. However, when the editor is entered, only one procedure is open for modification. When the user saves or scratches that procedure, the editor returns the user to the main STOL processor. At this point, the user can go on to execute other STOL statements or into another EDIT session by naming another procedure to be edited (new or old).

```
EDIT SC                    ; opening of editfile 'SC' and invoking of editor
ED SC                      ; short form
```

The editor is responsible for creating a runnable procedure out of the statements or statement modifications entered by the user. When the user completes the editing session by directing the editor to save the procedure code upon which

the user was working, the editor renumbers the statements in sequential order, creates a correct label table, and prepares linkage and maintenance information data such as catalog entries as required.

7.2 PROCEDURE CONTROL

These directives are used to control the overall execution of STOL procedures. Some directives are for use by an interactive user desiring to start up, to stop, or otherwise to control a STOL procedure. Other directives are used within a procedure to control the sequence of execution. A few directives may be used for both purposes. Manually entered statements have priority over statements executed within a procedure.

7.2.1 START and RETURN Directives

These directives are used to transfer to, and return from, a named procedure. The named procedure must reside in the system procedure library or the optional device. It is assumed that there will be some system-unique type of physical or logical access control to protect procedures from unauthorized use. The START directive may be used by an operator directly or may be included in a procedure body to call another procedure. When procedures are nested in this way, RETURN is to the next higher procedure.

A procedure also RETURNs when it runs out of statements to execute (i.e., when it runs into an ENDPROC). Examples are as follows:

```
START SC (parameters), devicename (optional)
                                ; start execution of procedure SC from the
                                ;  device specified; if no device specified,
                                ;  disk procedure library assumed

S SC                            ; short form; no parentheses needed
                                ;  if no parameters

START SC AT 45                  ; start of procedure SC at line 45
```

```
START SC UNTIL LABL          ; start SC and enter WAIT mode just
                             ;  before executing statement LABL

START SC AT 45 UNTIL LABL

START SC(TM(20,34),X2)       ; passing of the global variable name
                             ;  'TM(20, 34)' and the address of local
                             ;  variable X2 in the calling procedure to
                             ;  the called procedure SC

RETURN                       ; processing complete

IF (X1 .LT. X2) RETURN       ; conditional return
```

A recursive START call (i.e., a procedure starting itself) is not allowed.

7.2.2 WAIT Directive

Procedure execution is suspended until the conditions of the WAIT are met. If a WAIT directive is entered manually during the execution of a procedure, the currently executing statement is completed before the WAIT is honored (refer to Section 7.2.5). Examples are as follows:

```
WAIT 5                       ; wait 5 seconds

WAIT 2.5                     ; wait 2.5 seconds

WAIT (POWER .EQ. 20)         ; wait until the variable POWER = 20

WAIT 10 .OR. (POWER .EQ. 20) ; logical operators .OR., .XOR.,
                             ;  .AND., .NOT.

WAIT                         ; indefinite wait

WAIT AT 45                   ; enter WAIT mode immediately be-
                             ;  fore executing statement num-
                             ;  ber 45

WAIT 10:23:46.2Z             ; wait until specified GMT
```

```
WAIT 23:11R          ; wait specified time relative to some event

WAIT AT LABL         ; enter WAIT mode immediately before executing
                     ;  the statement labeled 'LABL'

W                    ; short form
```

WAIT modes are initiated and terminated according to the type of WAIT statement. The simple WAIT is unconditional; it is honored immediately and is terminated only by a GO statement. The conditional WAIT (e.g., WAIT 10, WAIT (POWER .EQ. 20), WAIT 10:23:46.2Z) is honored immediately and is terminated only if the wait condition becomes valid. The preconditional WAIT AT statement is honored only when the precondition becomes satisfied; WAIT mode is entered unconditionally at that point and is terminated only by a GO statement.

Any WAIT statement supersedes any previous WAIT statement; e.g., if a simple WAIT is entered manually while a WAIT 20 seconds is in effect, the 20-second condition is superseded and the system goes into an unconditional wait.

7.2.3 GO Directive

All forms of the GO directive cause the system to go, i.e., to start executing statements. Examples are shown below.

```
GO                   ; canceling of WAIT mode and execution of next
                     ;  statement in sequence

G                    ; short form

GO 20                ; position at line 20 and GO

GO LABL              ; position at statement LABL and GO

GOTO 20              ; regular form
```

7.2.4 POSITION Directive

This directive positions the procedure statement pointer at the statement indicated but does not execute the statement. The system enters WAIT mode to allow the user to verify that the proper statement has been located prior to execution. Procedure execution resumes only when the user enters a GO statement. Examples are as follows:

```
POSITION SET1        ; position at statement labeled 'SET1'
PO SET1              ; short form
POSITION 20          ; position at statement number 20
```

7.2.5 KILL Directive

This directive immediately stops execution of the procedure as shown below. The automatic mode of procedure execution is canceled and WAIT mode is entered. The currently executing statement is aborted (refer to Section 7.2.2). There is no short form of directive KILL; it is not desirable to facilitate the entry of a directive with so drastic an effect.

```
KILL                 ; enter WAIT mode immediately, position at next
                     ;  statement in sequence

KILLPROC             ; enter WAIT mode immediately, return from cur-
                     ;  rent PROC and point to next statement in se-
                     ;  quence in calling PROC

KILLP                ; short form of KILLPROC

KILLPROC ALL         ; enter WAIT mode immediately, return from all
                     ;  nested PROCs and await next statement from
                     ;  operator
```

Operator entry of the KILL directive during execution of another directive from the same operator immediately cancels the previous directive and causes the system to enter WAIT mode. If a user wants the current directive to complete

before halting, the WAIT directive rather than the KILL directive should be entered. Implementors are encouraged to utilize suicide as a more reliable means of implementing the KILL directive in most cases, while reserving murder for the more recalcitrant applications programs.

7.2.6 STEP Directive

This directive executes the procedure one statement at a time under manual control. After the execution of each statement, the system automatically enters WAIT mode. The next step is executed upon receipt of a GO directive. After a STEP statement is executed, the system always enters WAIT mode. Examples are shown below.

```
STEP ON
STEP                    ; ON assumed
STEP OFF
```

A short form is not provided. Additional arguments may be used on some systems to indicate slow motion or other unique execution modes.

7.2.7 Conditional (IF-Type) Directive

Two forms of the conditional statement are provided. In both forms, the IF condition is an arbitrary logical expression (using the logical and relational operators of STOL) enclosed in parentheses. The IF condition is evaluated at run time. If the IF condition evaluates to .TRUE., the system executes one specified set of actions; however, if the IF condition evaluates to .FALSE., the system simply goes on to the next directive, or optionally executes a different specified set of actions. All conditional statements begin with the directive IF, followed by the IF condition in parentheses:

```
IF (X .EQ. Y)

IF (TM(20,34) .GE. 0)
```

```
IF (POWER .GT. 20 .OR. POWER .LT. 10)      ; determination of whether
                                           ;  POWER out of limits
```

7.2.7.1 Conditional Perform

One form of conditional statement, the conditional perform statement, causes an entire statement to be performed only if the IF condition is .TRUE.. The conditional statement may be any STOL statement other than another IF:

```
IF (IF-condition) statement

IF (IF-condition) WAIT                     ; conditional WAIT

IF (IF-condition) LET COUNT = COUNT + 1    ; counting of number of
                                           ;  occurrences of some
                                           ;  event

IF (IF-condition) RUN PROG                 ; conditional running of
                                           ;  PROG (e.g., alarm)

IF (IF-condition) GOTO LABL                ; conditional branch to
                                           ;  statement LABL
```

7.2.7.2 Conditional Perform Block

The second form of conditional statement, the conditional perform block or IF-THEN-ELSE, causes entire blocks of statements to be executed conditionally as shown below. One block of statements, the THEN block, is executed if the IF condition evaluates to .TRUE.; optionally another block of statements, the ELSE block, is executed if the IF condition evaluates to .FALSE.:

```
IF (IF-condition)
. ;
. ; THEN-block
. ;
ENDIF
```

```
IF (IF-condition)
. ;
. ; THEN-block
. ;
ELSE
. ;
. ; ELSE-block (optional)
. ;
ENDIF
```

```
IF (X .EQ. Y)
START SC(1)
ELSE
START SC(2)
LET X = Y
ENDIF
```

The last example above starts the specified procedure SC with argument "1" when X is equal to Y; if X is not equal to Y, it starts SC with parameter "2" and then sets X equal to Y so that SC is to be started with parameter "1" in subsequent times through the procedure. In this manner, procedures can initialize themselves immediately upon being started after system initialization. The word "THEN" does not appear explicitly in the statement format as it is not necessary for recognition.

Implementors of STOL are encouraged to restrict the complexity of conditional statements to enhance implementability and procedure readability.

7.2.8 Loop Command

There is no directive in STOL specifically for looping, i.e., repeating a sequence of statements over and over until some condition is satisfied. Experience in previous languages antecedent to STOL has shown that a loop capability

is rarely used even if readily available in the language. For simplicity of STOL, there is no explicit loop control construct. If looping is ever required, it can be constructed from the conditional branch as in the following example:

```
LOOP: statement
      IF (IF-condition) GOTO LOOP
```

SECTION 8 - MISCELLANEOUS

8.1 LISTING OF CONTENTS OF RUN-TIME DATA BASE

The LIST directive is used to list (i.e., print out or display) some of the contents of the run-time system data base. The following are examples of this capability:

```
LIST PROC                  ; list of names of all PROCs
LIST PROC SC               ; list of specified PROC, SC
LIST CURVE                 ; list of calibration curves
LI CURVE                   ;short form
LIST LIMITS                ; list of current limits
LIST LIMITS TM(20,34)      ; list of current limits for specified var-
                           ;  iable, TM(20,34)
LIST PGM                   ; list of names of programs that can be run
LIST FORMAT                ; list of names of all display formats
LIST FORMAT POWER1         ; list of specified format, POWER1
```

8.2 EXTENSION MECHANISMS

A "pure" implementation of STOL will seldom, if ever, exist. Users will invariably have more operational requirements than those given in this document. The STOL philosophy encourages each implementor to extend the language to accomplish any additional unique requirements. The extension mechanisms are described in the subsequent paragraphs.

8.2.1 STOL Nucleus Capability

All STOL implementations are strongly encouraged to accept the STOL language nucleus defined in this document as controlled by the GSFC STOL Configuration

Control Board (CCB). Exceptions should be limited to those systems for which STOL provides clearly excessive functionality (e.g., a data system that would never command a spacecraft and for which the command functions are unnecessary and meaningless). Unofficial subsets of STOL, while not encouraged, are certainly better than inventing a one-of-a-kind syntax for operating a system.

8.2.2 Additional Parameters Allowed for a Directive

Any implementation may accept additional parameters in the argument field of a directive over and above those few mandated in the nucleus. The syntax of any additional parameters must be compatible with standard STOL syntax.

8.2.3 Additional Directives Allowed

Any implementation may define a new directive simply by providing an application program of that name that is capable of interpreting the specified argumentstring.

8.2.4 Standard Controlled Extensions

Various standard extensions will be defined and controlled either by the GSFC STOL CCB or by a local STOL implementation group (e.g., SMM Project). Implementors are cautioned not to compromise the essential features of STOL given in Section 2. The Systems Division for payload integration and test systems and the Mission Operations Division for POCC systems will undoubtedly standardize on some extensions for their particular applications.

It is requested that all STOL implementations, extensions, and local configuration control activities be brought to the attention of the GSFC STOL CCB by memorandum to the Chairman, Code 730. This measure will ensure an up-to-date mailing list for CCB minutes and language revisions and an informed implementation community.

8.3 DISCREPANCY REPORT/ENGINEERING CHANGE PROPOSAL FORM

STOL as a GSFC standard is formally controlled by the GSFC STOL CCB, Systems Division, Engineering Directorate, Code 730. The Discrepancy Report/Engineering Change Proposal (DR/ECP) Form shown in Appendix B may be copied and used to submit observed discrepancies and ECPs for modifying or extending STOL. ECPs will be serially numbered by the Systems Division secretary upon receipt and tracked to closure by the CCB.

APPENDIX A - STOL DIRECTIVES AND SHORT FORMS

The short form and section reference for each mnemonic are provided below (a dash is used where no short form is defined).

Mnemonic	Short Form	Section Reference
/ALLOW	/A	5.6
/CANCEL	/X	5.6
/CLEAR	/Z	5.7
/CMD	/	5.1
/LOAD	/L	5.4
/MODE	/M	5.2
/OBC	-	5.3
/RETRY	/R	5.8
/SEND	/S	5.5
ACQUIRE	AC	4.1
ASSIGN	AS	3.4
CHART	CH	6.5
CONVERT	CON	4.3
EDIT	ED	7.1.2
ELSE	-	7.2.7
ENDIF	-	7.2.7
ENDPROC	-	7.1.1
FORMAT	FO	6.2
GOTO, GO	G	7.2.3
HISTORY	HI	6.3
IF	-	7.2.7
INTERVAL	IN	3.3
KILL	-	7.2.5
KILLPROC	KILLP	7.2.5

Mnemonic	Short Form	Section Reference
LET	(null)	3.2.3
LIMITS	LIM	4.2
LIST	LI	8.1
PAGE	P	6.1
POSITION	PO	7.2.4
PROC	-	7.1.1
RETURN	-	7.2.1
RUN	(null)	3.3
SNAP	SN	6.1
START	S	7.2.1
STEP	-	7.2.6
TERM	TE	3.3
WAIT	W	7.2.2

APPENDIX B - DISCREPANCY REPORT/ENGINEERING CHANGE PROPOSAL FORM

The form shown in Figure B-1 should be used to report any discrepancies discovered in the STOL description. It may also be used to propose changes or additions to STOL.

GSFC STOL DISCREPANCY REPORT/ENGINEERING CHANGE PROPOSAL

OFFICE USE ONLY

SERIAL NUMBER	RECEIVED BY	DATE

ORIGINATOR USE ONLY

ORIGINATED BY	PHONE	DATE

DESCRIPTION OF DISCREPANCY OR PROPOSAL (ATTACH OR LIST SUPPORTING INFORMATION)

CCB USE ONLY

RESOLUTION OR DISPOSITION

RESPONSIBLE ENGINEER	APPROVED BY/DATE	CLOSED BY/DATE

Figure B-1. Discrepancy Report/Engineering Change Proposal Form