GOAL OPERATIONS

AEROSPACE LANGUAGE

JOHN F. KENNEDY SPACE CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

PREFACE
GROUND OPERATIONS AEROSPACE LANGUAGE (GOAL)

GOAL OVERVIEW

This Overview Document relates the history that led to the development of the GOAL Language and provides a summary of the features and capabilities of GOAL.

OTHER DOCUMENTS TO BE DEVELOPED ARE:

A GOAL Text Book to be used in conjunction with instruction. It will be functionally organized to allow an instructor to cover similar statements together.

A GOAL Reference Manual to facilitate quick access to desired statements. It will be alphabetically arranged.

A GOAL Self-Instruction Manual for individuals desiring to learn GOAL without additional assistance. This manual will probably follow a "building block" approach.

Prepared for: Director of Center Planning
and Future Programs

By: Checkout Automation and
Programming Office
Launch Vehicle Operations

TABLE OF CONTENTS

I. HISTORICAL REVIEW

The initiative to produce a standard test language for the Space Shuttle emerged from the experiences gained in test automation during the Saturn/Apollo program. Design of the Saturn launch vehicle and its associated ground support equipment was such that most commands could be issued by, and most vehicle and ground support status data, could be sensed by the ground support computer complex. Consequently, a high level potential for automation existed in this basic design. Initially, the applications program (test programs) written for execution in the ground computer complex were almost entirely for loading and verifying the onboard guidance and navigation computer, for checkout of the control system, and for checkout of the emergency detection system. Other systems were initially checked out manually.

For these early automated packages, the typical user-programmer communications gap was experienced firsthand as test personnel attempted to comprehend the programmer's interpretation of the test requirements. Generally, changes were difficult to implement and to understand. Full control over the test came only after much actual experience. Even then the details of some operations were obscurely embedded in the test packages so that the test engineers had difficulties in comprehending the subtleties of the programmer's logic. At the time when automatic checkout could have eased the mounting strain associated with the pressing schedule, the lack of a common language to communicate requirements and to describe the computer programs further burdened the launch team. Problems arose from the lack of concise uniform test notations that could be readily understood by personnel of all the different engineering elements.

During the Saturn IB launch period, a basic set of coded operators, suitable for applications programming, was added to the Ground Computer Operating System. This source language was entitled ATOLL for Acceptance Test or Launch Language and was conceived and implemented by Marshall Space Flight Center. In the early application of this system, it was learned that the success of the computer language could be strongly dependent upon its method of implementation through the language processor and its execution through the operating system (real time executive). The first ATOLL capability employed an on-line translator, which in the Saturn Ground Computer System, was operationally inefficient. Changes were made to correct the disadvantages, and a much more efficient system was available at the outset of the Saturn V program. Nevertheless, prejudices against test automation lingered for several years until the test engineers were convinced that automation would prove to be a useful and responsive tool. Now the language has evolved to the point where most vehicle disciplines use the language extensively for automatic test procedures.

For Saturn/Apollo NASA was in the same category for language development as most of the other groups who develop test languages within industry or Government. That is, the language was developed only after the equipment and applications were firmly established and was among the last items to be implemented. Under this circumstance, the language was subject to criticism as another new, unfamiliar, troublesome system that complicated the engineer's life and added to his list of problems.

We now have the opportunity to standardize the basic communications among elements of ground and in-flight testing at a point in the system acquisition cycle where it will become a natural part of the program. This will avoid costly retrofits and difficult "unlearning" exercises at a later time.

A properly defined engineer oriented language will serve as the basic tool in insuring commonality for Orbiter/Booster/GSE Test and Ground Operations Procedures while inherently providing the capability to: (1) efficiently automate manual procedures, (2) readily adapt design procedures for operational use, (3) reduce supporting documentation, (4) efficiently cross-train test personnel, (5) minimize impacts from changes, and (6), in general, will be a prime contributor in support of the rapid turnaround requirements.

Thus, in July 1970 contracts were awarded to Martin-Marietta, Denver Division and M & S Computing, Inc., Huntsville to develop requirements for a standard test language. From the results of their reports a language requirements document was published by KSC in May 1971. Since July, 1971, a detailed language specification has been under development by a team of NASA software engineers.

II. DEVELOPMENT OBJECTIVES AND REQUIREMENTS

The development of GOAL was based upon several objectives, namely, the language:

a. Requirements must be consistent with and support the design concepts and requirements of the Space Shuttle.

b. Will not be constrained by specific test equipment.

c. Will allow the same procedure to be used for both manual and automatic testing.

d. Will provide for a flexible monitoring capability.

e. Will provide the capability for test personnel to communicate with mission software.

f. Will be easy to use by test-oriented personnel not necessarily skilled in programming techniques.

g. Will be easy to read and will be self-documenting.

h. Must be compatible with the philosophy of performing concurrent testing.

From the results of years of ground testing experience and an extensive review of many existing languages the following language requirements were tabulated:

English-like Words, Structure, and Punctuation

The keywords of the test language form the building blocks of the language and care should be taken to select words natural to the Space Shuttle test-environment. Abbreviations should generally be avoided and only in cases

where the abbreviation has gained universal acceptance will a deviation be considered. These keywords will be ordered in a logical English form. This ordering will promote learning and retention while allowing comprehensive error checking. The punctuation symbols and their meaning should be consistent with general usage.

Comments

A comment is an expression which clarifies a particular statement or functional aspect of a group of statements but is not required to technically define the procedural actions of the operation. When automated, comments have no effect on the operation of the computer performing the assigned tasks. In some applications comments provide the added flexibility required to allow the computer listing to be the single control document. Therefore, they should be easily inserted into the language statements.

Total Control of the System Under Test

The language should allow test personnel to specify test point control to the lowest level that is available under the given system configuration. The level of access to a Line Replaceable Unit will probably be different in off-line test environment than in the operational system configurations. The types of signals used in controlling the test or function operations must not be constrained by the standardized test language. For the Space Shuttle System, the language must recognize the requirement for discrete events, digital codes, and proportional values (digital representation of analog values).

Data Sampling

The test language should support data gathering consistent with system constraints and ground rules. The ability to exercise control over the system under test and the ability to measure parameters and status of the test item are the foundations for testing. Sampling rates should be by a system limit and not a language limit. It is possible, after the system constraints have been defined, to incorporate the constraints into a language processor which will alert the user if his procedure conflicts with system constraints. As with control, data samples may be discrete events (ON/OFF), digital codes (LRU address), or proportional values (0-100%).

Data Comparison

Data comparison is the next basic level above the ability to control test items and the ability to acquire the data from the test article. The comparison may take many forms (test versus predicted, per cent change from last value, deviation during time interval) and the results may be saved for use later or may immediately effect a change in the processing sequence. The data comparison capability should include arithmetic and Boolean terms in a form familiar to the test environment.

Time Controlled Events

The extensive use of time factors in sequencing and testing is a salient feature of test and ground operation procedures. During launch preparations, test and functional operations are often controlled by a specific time relative to liftoff (COUNTDOWN CLOCK). Other functions are based on given time of day;

e.g., usually referenced to Greenwich Mean Time. Mission elapse time may be used for inflight test and operations. Many of the system sequences must be performed in a close time-controlled sequence relative to an occurrence of an event within the vehicle. There are also those indicators that must be checked at a periodic rate. The comprehensive use of time in testing warrants major consideration in selecting the proper keywords to be used in a test language.

Monitoring the System Under Test

Recognizing that detailed checkout philosophies have not been defined for future vehicles, an increased dependence upon monitoring is an established trend in the space and airlines industries. Though most of the existing space system checkout facilities include monitoring capabilities, most of the automated test languages seem to exclude this capability. Realizing that interaction with the real time hardware/software system could dictate some adjustments to a predefined test language, the basic language should be able to define items to be monitored; e.g., conditioned by time (start/stop and sampling interval). The general capabilities of the language should also be available for specifying monitoring packages.

Information Presentation and Recording

In the automation of test requirements, the manner in which the data is presented to the test evaulator can significantly influence the effort required in deducing the proper action to be taken or determining whether or not all aspects of the test were completed satisfactorily. The ability to record or save selected data, usually correlated with time, is also an operation that is frequently performed throughout system testing and must be supported by the test language.

Console Interaction

At times during a test, an anomaly may appear that justifies suspending activity until the system status and integrity can be confirmed. The decision may be just to resume operating steps, or rerun certain steps, or to deviate in some way by changing test parameters. The basic language requirements to be derived from this situation are: (a) the language must be able to suspend execution until requested to continue, and (b) the language must accommodate the need to change test parameters from a console for certain predefined parameters. This feature is also dependent upon the operational hardware/software system and refinements to the language may result from later system definition.

Data Manipulation

Data manipulation is considered to encompass numeric formulas, relational formulas, and computer associated assignment statements. Generally, the languages that provide arithmetic capabilities provide these capabilities in a formula type statement (e.g., FORTRAN type statement). This is a reasonably natural and compact way to describe the required calculations. The relational formulas are of the comparison type usually expressed in a form of 'EQUAL TO' or 'NOT EQUAL TO'. For automatic testing, a closely related requirement exists, which is the moving of data items between storage cells.

Computer-to-Computer Communications

It appears that the Space Shuttle will have inter-computer communication in some form. It could be between the central computers, between a central computer and an engine computer, or between a central computer and an off-vehicle computer (ground system or space station system). The test language will provide this capability in a manner that will ensure two-way communication between digital devices. This will then include the capability to transmit and receive data from some of the more complex data bus interface units.

Incorporation of Packages Written in Other Languages

The need to specify certain functions in assembly language is not expected to disappear entirely. It complicates a language considerably to include every capability necessary to handle highly exceptional requirements. However, to ensure that exceptional requirements can be fulfilled, it is necessary to have some capability to incorporate assembly language programming consistent with the design intent of the language. This feature will probably be used only by the sophisticated test programmer and under a higher level of control and validation than required for packages written in the standard test language. This requirement recognizes that other languages, assembly level or high order, may be needed and the standard test language must support this concept.

Test Sequence Designation

A desired test sequence may be stated in several ways. A test procedure usually contains many functional elements. These functional elements are comprised of a varying number of individual operating steps (statements). If the functional element must be repeated a number of times, then it may be come a subprocedure and referenced by the main procedure, or the steps of the elements may be inserted the proper number of times, or direction may be given to repeat the required steps the appropriate number of times. This looping type capability is even more important when the procedure is automated because it often has a direct impact on storage allocations required for the procedure. This requirement includes the need for directing the sequence based on the condition of test indicators.

Identification of Language Packages and Components

This includes the obvious need of the ability to reference a specific test procedure for use during testing. For incorporating changes, and for configuration control. Often procedures must reference other procedures. Individual statements within the procedure have the same need; therefore, the language will provide for labeling of separate packages as well as individual statements.

Data Bank Requirement

The data bank concept is the feature of the language that allows the language to be independent of the test equipment. It is basically a cross reference table that relates the engineering terminology of a test point to the test equipment parameters required to access the test point. For example: the procedure might read APPLY BOOSTER MEASURING POWER. The data bank would take BOOSTER MEASURING POWER and provide the necessary data for the automatic test

equipment such as data bus number, interface unit address, line replaceable unit designation, and other system related values necessary to locate the test point and to accomplish the desired results. For the Space Shuttle, there will probably be a centrally defined and controlled list of test points similar to Apollo documents; e.g., the Saturn V Discrete Running List, Saturn V IP&CL, and ACE-S/C Programming Requirements Process Specification Parameter List. Such a document for the Space Shuttle would furnish much of the information required in the data bank. This is the final link between the language and the test system. It also allows procedures to be written independent of the test system and in advance of the final configuration.

Table Definition

Special attention should be given to the definition and use of tables. They should prove to be a significant aid in test preparation. The flexibility and usefulness of table operations warrants the inclusion of this capability even though it might appear more complex than desired. Tables are currently being used in Saturn checkout procedures and have become an integral part of daily operations and major tests. Generally, they support such functions as system status checks, switch scans, and performance monitoring.

Data Types

Investigation of the Space Shuttle test and checkout applications and previous efforts at the definition of test languages leads to the conclusion that the following constant and data types are required in the new language.

(1) Constants

- (a) Integer
- (b) Fixed Point
- (c) Boolean
- (d) Text
- (e) Binary (Octal or Hexadecimal)

(2) Data Variables

- (a) Integer
- (b) Fixed
- (c) Boolean
- (d) Text
- (e) Time

Writer Aids

It appears consistent throughout aviation and space vehicle checkout procedures that the writing task is a relatively small portion of the overall procedure cycle. The number of people using, reading, validating, or changing procedures can sometimes become rather large. Therefore, while primary consideration must be given to the larger group, the test procedures writer is certainly a vital link in the cycle and should be afforded the capability required to insure maximum economy without compromising the primary language objectives. The requirement includes such standard concept as replacing the name of one item

with another, macro features, and subroutine capabilities. Portions of other language requirements also may be implemented in a manner which facilitates procedure writing. The final selection must be constrained by the fact that the user is ill-served by a language which allows him to conveniently describe an erroneous procedure.

Reaction to System Changes

Test and checkout requirements include the basic needs of being able to respond to such general system indicators as 'start processing,' 'terminate processing,' and 'suspend processing.' The command could have been originated by a manual entry, another procedure, or an internally-generated command due to detection of a serious anomaly. Although the Space Shuttle does not appear to be using system interrupts, the language should be able to accommodate interrupts for component type testing and to preclude a language impact if the Space Shuttle or Space Station implements interrupts at a later time.

Language Character Set

To promote general applicability of the language to as many test applications and test equipment as practical, only characters may be used that are common to the USA Standard Code for Information Interchange Code (ASCII) and the Extended Binary Code Decimal Interchange Code (EBCDIC). These characters are as follows:

(1)	Capital Letters:	A-Z		
(2)	Numbers:	0-9		
(3)	Special Characters:	+	:	@
		-	!	blank
		=	?	-
		"	<	*
		'	>	#
		.	(	$
		,	)	%
		;	/	&

III. LANGUAGE SCOPE AND FORMAT

GOAL, (Ground Operations Aerospace Language) is a test engineer oriented language designed to be used to standardize procedure terminology and as the test programming language to be used for ground checkout operations in a space vehicle launch environment. It encompasses a wide range of testing, including vehicle systems and subsystems preflight checkout, ground preflight operation such as propellant transfer, support systems verification, ground power control and monitoring, etc. The language is compatible with a wide variety of engineering design, requiring primarily command/response (analog and digital) to the systems to be tested. It may be used in the checkout of line replaceable units, both on-board preflight, and in the shop. It allows the same procedure to be used in both automatic and manual modes. GOAL permits a high degree of readability and retainability by providing the necessary operators required for testing, expressed in a familiar notation. Therefore, it is easily learned and understood by personnel not necessarily skilled in programming techniques. After much consideration it was decided to standardize the language statements in a

readable format prior to compilation, rather than require language readability dependency on a conversion program.

Language Components

There are five distinct components in GOAL. The two primary components are the PROGRAM and the DATA BANK.

PROGRAM:

A program is an ordered group of statements which, when executed by a computer, will progress through the predefined test steps. Within the program, there are two types of statements: DECLARATION and PROCEDURAL.

DECLARATION STATEMENTS consist of data, table, or list declarations. DECLARATION STATEMENTS must be grouped at the beginning of a GOAL program. They are non-executable statements which reserve storage and signify data types during program compilation.

PROCEDURAL STATEMENTS comprise the remainder of the program. These are the statements which will actually be executed by the object machine to perform the desired test operations. PROCEDURAL STATEMENTS are further classified into external action and internal action statements. External action statements stimulate action external to the program. Internal action statements are used for the more "programmer oriented" tasks such as directing program control, timing, and sequencing.

DATA BANK:

Past experience has revealed the need for implementation of a data bank to supply certain declarations, translations from English notation to address patterns, calibration data, and other modules of common usage requiring centralized control. This concept is vital to minimize the languages dependence on the test equipment.

While the initial work has aready been started, the complete definition of the data bank will be possible only after the necessary detailed system information is available. However, the GOAL specification can be used for test procedure definition independently of this work.

The data bank is a separate software entity from the program. It contains a collection of specify statements. A data bank acts as a central file which provides the linkages between the test procedure and the system under test. GOAL allows the use of more than one data bank for compilation of a program. A data bank is required only if the test program is to access system subroutines or external test points.

The three secondary GOAL components are the SUBROUTINE, MACRO, and NON-GOAL.

A SUBROUTINE is a self contained set of statements which perform a specific task. It is defined once within a program or data bank and may be executed by an appropriate perform statement. The subroutine organization is the same as a program, with Declaration Statements preceding the Procedural Statements. The calling statement transfers control back to the main program at the next sequential step following the call. Subroutines may contain variable parameter locations which are specified by the call statement, or they may be completely independent of outside (main program) data.

A MACRO is a method of allowing the test writer to abbreviate character strings which must be repeated throughout his program. The character strings to be repeated may be in the program, data bank, or subroutines. Each MACRO is assigned a language label. The writer may call the MACRO in those locations where he wants the sequence to appear and the compiler will perform the task for him. The end result on the output listing is the same as if each step had been individually coded by hand.

A NON-GOAL component must be contained within a subroutine and that subroutine must be within a data bank. NON-GOAL components can be used to provide capabilities that are not inherent in the GOAL statement reportoire.

GOAL STATEMENTS

The general structure of a GOAL statement is the same as a simple imperative English sentence, with the subject understood to be the computer. The minimum requirement for a GOAL statement is a verb; however, most statements also contain an object to receive the action. An optional phrase may be used to modify the action. That is, tell when, how often, or how long to perform the action. Example:

Optional Phrase	Verb	Object
AFTER <GMT> IS 12 HRS 30 MIN,	OPEN	<INLET SUPPLY VALVE> ;

GOAL statements are written in free field format. The free format permits the writer to position elements on the page, as he desires, for clarity. It has greater flexibility since fixed fields can later be legislated if desired.

All procedural statements may have a statement number which consists of up to six numerals preceded by the word STATEMENT, STEP or S. The statement number uniquely identifies a statement for branching and reference purposes. For example, S14, STATEMENT 651, STEP 3141 could be statement numbers and need not occur in any particular order.

GOAL notation is in terms of the system under test (SUT) and is the notation of the test engineer. A GOAL statement is designed to accomplish a certain test function. Knowledge of the actual linkage between the computer and SUT is not required because it is obtained from the data bank.

IV. LANGUAGE CAPABILITIES

The GOAL Procedural statements provide the basic language capabilities. These statements have been grouped into six functional areas which are defined as follows:

External Test Action

These statements provide interaction with and control of the system under test (SUT). Commands or data may be sent to the test equipment external to the program and inputs to the program may be acquired.

Internal Sequence Control

These statements control the execution sequence of the program statements.

Arithmetic/Logical Operations

These statements provide the mathematical capabilities of GOAL. They contain the mathematical capability to add, subtract, multiply, divide and exponentiate using notation compatible with the current FORTRAN IV system.

Execution Control

These statements provide capabilities for concurrent program execution and also for serial execution of other programs and subroutines.

Interrupt Control

These statements control the action to be taken when an interrupt occurs.

Table Control

These statements enable selective processing of table entries.

V. SUMMARY

The necessity for a standard test language must be emphasized. Care should be exercised in selecting the scope of tasks that a language describes. The assertion that 'one language should be used for everything' sounds attractive, but under close examination this approach would defeat the objective for simplicity and readability. Many languages have been reviewed to determine if they would be sufficient to meet the requirements for a ground test and checkout language. Many (such as BASIC, CAGE, SPL, etc.) were rejected because of the lack of necessary testing capabilities. Others like FORTRAN required too many and too complicated statements to perform specified tasks. Test (Ground/Inflight) and ground operations procedures represent a logical subdivision of the total task, and the language supporting these areas should be capable of defining most of the required activities. While preserving the general readability such a capability would help minimize the tedious, costly, time-consuming traditional interface between the test engineer and the programmer.

For the Space Shuttle Program, we must make maximum use of the lessons learned through the years of design and launch experience. The very nature of the Space Shuttle design and the essence of the operational concept dictate that more be accomplished in a shorter period by fewer people than ever before. Automation, then, becomes a requirement for operations, not an elective. To effectively apply extensive automation, a test language has no suitable alternate.

REFERENCES

The following documents were used in preparing this overview.

1. Development of a Test and Flight Engineering Oriented Language, Phase III Report, Martin Marietta Corporation, NASA-KSC, MCR-70-424

2. Development of a Test and Flight Engineer Oriented Language, Final Report, Volumes I and II, M & S Computing, Inc., NASA-KSC, Report Number 70-0034

3. Development of a Test and Flight Engineering Oriented Language, Phase II, Report, Martin Marietta Corporation, NASA-KSC, MCR-70-365

4. Development of a Test and Flight Engineer Oriented Language, Phase II, M & S Computing, Inc., NASA-KSC, Report Number 70-0031

5. Development of a Test and Flight Engineer Oriented Computer Language, Technical Proposal, M & S Computing, Inc., NASA-KSC, Reference RFP 3-309-0

6. Requirements For a Standard Language For Test and Ground Operations, NASA-KSC, KSC-TR-1111