



Apr 1st, 8:00 AM

## Aloft: A Language Oriented to Flight Engineering & Testing

W. F. Kamsler

*Martin Marietta Corporation Denver, Colorado*

J. Gyure

*Martin Marietta Corporation Denver, Colorado*

Follow this and additional works at: <https://commons.erau.edu/space-congress-proceedings>

---

### Scholarly Commons Citation

Kamsler, W. F. and Gyure, J., "Aloft: A Language Oriented to Flight Engineering & Testing" (1971). *The Space Congress® Proceedings*. 4.

<https://commons.erau.edu/space-congress-proceedings/proceedings-1971-8th/session-2/4>

This Event is brought to you for free and open access by the Conferences at Scholarly Commons. It has been accepted for inclusion in The Space Congress® Proceedings by an authorized administrator of Scholarly Commons. For more information, please contact [commons@erau.edu](mailto:commons@erau.edu).

## ALOFT: A LANGUAGE ORIENTED TO FLIGHT ENGINEERING & TESTING

W. F. Kamsler and J. Gyure  
Martin Marietta Corporation  
Denver, Colorado

### ABSTRACT

A high order computer language called ALOFT has been developed for the checkout and operation of complex space oriented equipment such as the proposed NASA Space Shuttle. The flexibility of the language makes it equally suited for use with existing launch vehicles such as Saturn, Titan, etc. and space systems such as Space Station, Viking, etc. With such flexibility it can be assumed that the language will be equally acceptable to future vehicles, space experiments, etc.

Flexibility is obtained by making the language independent of any test system and providing for the user to define a wide variety of words and functions. This later capability also makes it independent of the device it is testing.

The paper describes the language and its syntax. It also shows its ability to operate in a multidiscipline environment independent of the test system.

### LANGUAGE CHARACTERISTICS

ALOFT has been designed to be independent of the test system and of any particular test article. The language does, however, contain capabilities to enable it to cope with Space Shuttle peculiar features and requirements.

A study of these features and requirements, along with the general test and checkout problem, resulted in the determination that the capabilities described below should be included in the language.

#### Test oriented capabilities:

- Test initiation;
- Application of stimulus;
- Measurement of output;
- Comparison of results;
- Man/machine interfaces;
- Records and logs with time tags;
- Monitoring;
- Clock and time controlled actions;
- System, subsystem, and unit testing.

- Independence with respect to testing equipment via:
  - Dictionary data banks;
  - Common character set;
  - Statements which are free form with respect to input media;
  - Minimum interaction with operating system;
  - Test writer-created safing features,

#### Flexibility provided by:

- Full arithmetic and relational operator set;
- Thirty-two character data names;
- List and table capability;
- Simple loop capability;
- Subroutines;
- Integer, fixed point, Boolean, text, binary, and time data;
- Simple numeric and Boolean assignment statements;
- Unconditional and simple conditional transfers;
- Interrupt initiated routines.

#### Engineering reader orientation with:

- English words for primitives;
- Natural English forms as delimiters;
- Natural statement structure;
- Generalized commenting capability.

#### Concurrent test execution provisions:

- Initiated via language primitives;
- Synchronization capability;
- Interrupt capability;
- Meaning dependent on language processor implementation.

#### Self-extension through:

- Macro definition capability;
- Other language capability;

#### Special communications requirements:

- Computer to computer;
- Computer to data bus.

## PART I

### A. Why another new language?

A study was undertaken to determine the characteristics and capabilities of several higher-order, test oriented languages with respect to advanced space-oriented applications. The languages investigated included: ATOLL, ATLAS, CLASP, ATOLL II, MOLTOL, CTL, VTL, TOOL, ADAP, and ASEP. This study included the investigation of the languages themselves and also a study of existing language applications. The results provided a background and understanding of the role that a test-oriented language plays in the acceptance and implementation of automation. Other results include a greater appreciation of the degree to which test system characteristics and limitations have affected the development of "test-oriented" languages, and the degree to which test system characteristics have dictated test philosophy.

It is interesting to note that practically all of the test-oriented languages (TOL's) established the same objectives to direct the design of a language which would be useful in accomplishing automatic checkout tasks.

However, few TOL's have been able to accomplish their stated objectives without compromise. Developing a TOL for a specific test article and utilizing existing equipment affected the resulting language design. In all examples studied, with one exception, this has been the case.

A common tendency with all of the TOL's studied (perhaps ATLAS excepted) is for the language to be writer oriented. The common reader oriented objectives seem to be subverted by the writer's natural desire to reduce the number of characters to be written on the coding form. This results in abbreviations, mnemonics, fixed formats, unnatural (but shorter) word usage, and other forms of coding that are non-English like and require study by engineers who should be able to understand the test programs but don't really have the time. The writer is generally supported by the compiler designer because of the simplifications possible in recognizing and analyzing source language primitives and statements.

As a result the most common complaint about any TOL is that it is too difficult to learn to read and understand. This is usually brought about by the use of mnemonics, fixed fields to distinguish parameters, etc. and by the use of terms that the language designers erroneously considered to be generally understood.

The facilities of the language, such as declarations, specifications, and definitions, can usually be used by the writer to simplify his writing task at the expense of readability. Since these same facilities can be used to enhance readability (and sometimes are so used) the result becomes more a function of the writer's motivation than of language definitions and rules.

The absence of arithmetic capabilities has also been noted for several of the languages. These have had to be implemented through the use of machine language subroutines.

The absence of digital data transfer and computer intercommunications capability is a special problem with regard to advanced space vehicles.

TOLs developed to this date have (with the exception of ATLAS) been designed for a specific test program using identifiable test equipment. The languages are test system/test article dependent. As a result, they are not readily adaptable to new test systems or test articles.

None of the TOLs considered would fully satisfy the broad test-oriented applications area as envisioned by the authors. Work would need to be done to further the goal of test system independence of the languages other than ATLAS. ATLAS itself would need additions for system-oriented functions since it is primarily for use in a bench type unit testing operation.

Modification of a language is not impossible; however, the structure of the languages studied and their related processors are restrictive enough that it is more efficient to start over, rather than to accept limitations on the capabilities of a language which result from historical factors and are not related to its projected use.

### B. Who will use this new language?

A test and flight engineer oriented language implies a language specifically fitted to the education, technical vocabulary, experience and training of test and flight engineers. There are others who will also use the language including test and flight equipment designers and programmers. Each potential user will be identified and their characteristics and language requirements discussed in the following paragraphs.

#### 1. The Test Writer

The test writer prepares a step-by-step sequence of events to program the test system based on the design characteristics and performance requirements of the equipment to be tested.

The test writer should have a thorough knowledge of the prime equipment design, control, test and operation. The latter requirements make him a system applications specialist, requiring engineering training and test experience.

To the extent that he has to learn internal details of the operation and peculiarities of the test system and unfamiliar programming language features, his time is diverted from the study of the equipment to be tested.

His requirements call for a test language that is easy to learn and is as independent of the specific test system as possible.

When the test writer becomes proficient in the use of the language, he generally desires means of abbreviation of the language elements, statements, and routines in order to save writing time and to avoid inadvertent errors. He needs a language which allows the writer to predefine terms and abbreviations which can subsequently be used to shorten the writing task.

## 2. Design Engineer

The test writer may or may not be a designer or representative of the system or unit being tested or controlled. If he is not, it is generally considered necessary that the test programs be reviewed and approved by design engineering personnel.

The design engineers for advanced space-oriented projects include capabilities in a broad range of disciplines such as electronics (including digital systems), hydraulics, pneumatics, propulsion, RF systems, and life support.

The design engineer's requirement for a test language is that it consist of familiar technical terms and a logical, readily understood format. The prime equipment design engineer does not want to learn the details of the test system and its internal operations in order to understand the test and control interfaces with his equipment.

## 3. Operating Personnel

The eventual execution of the program will involve test and control operators. They must be intimately familiar with all operator interface hardware and with the system level operating characteristics of all hardware elements of the equipment under test and the test (or control) system. They are customarily test engineers at higher equipment operation and test levels and advanced technicians at lower (LRU) test levels. In the ultimate system operation level, they may be flight engineers, astronauts, or pilots.

These language users will review and approve the test procedures. Of prime concern will be the human engineering aspects of operator interactions, instructions, decisions, holds, emergency routines, displays, etc., and any related aids that will be made available. This user requires a language that is easily read and understood and one that defines operator involvement clearly and non-ambiguously.

## 4. Quality Assurance

The quality assurance personnel are involved in several areas of program preparation and execution. One role is to assure that the test procedure meets or exceeds all documented test requirements and indeed verifies the required performance capabilities of the system or unit under test. Another is to verify, either during or after program execution, that the tests and/or control activities were indeed performed and that acceptable results were obtained. This user requires a language that is non-ambiguous, one that has the

ability to clearly state evaluations and decisions, and a means to clearly specify how the system is to display and record the results of test evaluations, significant branches, and test completion.

## 5. Safety Engineering

Verification that test and control procedures contain satisfactory emergency safing routines and precautions is usually assigned to a specific organizational group or panel. Due to the potentially hazardous activities involved in all phases of advanced and current space programs, there are customarily many checks and rechecks of the integrity of test and control programs as well as equipment. The performance and integrity of equipment at all levels of testing is of concern, because it will eventually be used at higher levels and in hazardous operations.

This user requires that the language be easily read and understood, and facilities for clearly defining and presenting warnings, precautions, safing routines and monitoring be provided.

## 6. Customer

The term "customer" as used herein includes all of the generally higher levels of program or project management such as those implied by headquarters, project management, integration, coordination, etc., that are normally attributes or roles of customers.

The background and training of involved personnel may span several technologies. Here, the language requirement is for readability without extensive training. It should not be necessary to acquire a detailed knowledge of the test or control system in order to understand the test program.

This survey of the various users of a higher-level test-oriented language helps to establish the objectives such a language must meet in order to successfully accomplish the task for which it is designed.

## C. Language Objectives

The ALOFT language was designed with the primary emphasis placed on meeting the following objectives.

### 1. Independence with Respect to Testing Equipment

The nature of the Space Shuttle and other advanced space systems makes it necessary that the test language be developed to be independent of any particular set of equipment. There are expected to be several differently configured test systems which would use the language. These include numerous test system configurations at contractor and vendor facilities.

The advantages of this approach are that the test system need not be completely designed at the time that the writing of test programs is initiated. The language can work in several test systems, thereby allowing for easier communication of test

requirements between vendor, contractor, and customer. Finally, a longer life expectancy of the language and its associated processors can be expected since the obsolescence of a particular test system will have no direct language effect. The same language will be capable of being used from one program to another.

## 2. Flexibility

A test language must provide flexibility to meet both the anticipated and unanticipated needs of future space-oriented programs. The advantages of flexibility are that it gives a language a much better chance to meet the necessary requirements which arise as a result of changing technologies. Flexibility also contributes to the greater usefulness of the language to a particular project and extends the useable life of a language over a number of projects.

## 3. Engineering Reader Orientation

Two approaches to the definition of a language with respect to the users of that language can be identified. One approach is to define a language with maximum ease of writing (which generally results in degraded readability). Another approach is to define a language with maximum readability (which puts a heavier burden on the writer).

In space vehicle checkout applications it has been historically true that the writing task is a relatively smaller portion of the overall programming cycle, while the resulting tests must be read and validated by a number of people. Therefore, the emphasis is placed on maximizing readability and providing aids within the language to assist a reader in understanding tests written in the language.

## 4. Self-extension Capability

A self-extension capability is necessary to enable the language to keep up with new developments in space vehicle checkout without resorting to language and compiler modification which is a time consuming process requiring professional programmers. An important consideration is the constraint of such a capability so that difficulties are not introduced for those who must read and interpret the resulting language extensions.

## 5. Computer/Computer and Computer/Digital Interface Unit Communication Capabilities

Present Space Shuttle concepts require multiple computer configurations in a central computer complex linked by multiple data buses to other computers and special digital interface units. These computers and special digital interface units in turn interface with the line replaceable units. Test and checkout of the installed line replaceable units requires communication between the computers and the digital interface units via the data buses.

## 6. Maximum Use of Past Language Development Efforts

Many test-oriented higher-level languages have been developed for specific application areas. Since it is desirable that a new language have a longer and more useful life than previous efforts, it is necessary to take into consideration the advantages and disadvantages of these predecessor languages. Utilizing this information enables the insightful development of a language which will be able to effectively handle past and current requirements and yet be capable of effective use for some time to come.

### D. General Language Characteristics

The ALOFT language incorporates the following characteristics in order to achieve the objectives defined previously:

#### 1. Test Orientation

The following discussion identifies the test oriented functions which are implemented in ALOFT.

##### 1.1 The General Nature of Testing

Testing involves the initiation of activity with controlled predetermined conditions and then analysis of the resulting activity. The predetermined conditions are in the nature of applied stimuli while analysis involves the measuring and comparing of the responses. It is with this activity and analysis that a test-oriented language must concern itself.

##### 1.2 Initiation of Test Execution Via the Language

To initiate the action of a test from within another test, the language must be able to call or perform a test sequence. Such a request for action permits a test to commence.

##### 1.3 Application of Stimulus

The first test function usually performed is the application of a specific stimulus or control signal to a specific unit under test.

The application of stimulus signals may take many forms. Major categories include DC signals and AC signals, normally classified as analog signals. Application of single level DC signals usually falls into the discrete category. A third category consists of digital stimulus. The nature of the Space Shuttle (with its integrated avionics) indicates that a built-in stimulus (contained in an interface unit (IU)) will have to be programmed. As far as the test writer is concerned, he must request the application of the stimulus just as he would in any other test situation. Where the natural or operating stimulus cannot be called into use, an artificial stimulus is applied which produces a known output for a known input.

#### 1.4 Measurement of Output Signal

Once a stimulus is applied to a unit under test, an output is expected in response to the input stimulus. The language provides for acquiring that output and retaining it for further manipulations. In the Space Shuttle application, outputs will be sensed by IUs attached to Line Replaceable Units (LRU) and the data then is placed on the data bus to be received by the central computer complex.

#### 1.5 Comparison of Results

It is generally necessary to determine if the output acquired as a result of a measurement function is satisfactory with respect to some expected value. This output value is then compared to some predetermined value, with appropriate tolerances, and the results are used to indicate some further action.

#### 2. Naturalness of Statement Structure

The statement structure of ALOFT is based on an engineering oriented English format.

The English-like format of the language enhances the capability of a varied class of readers to understand the tests written in the language. The potential for error on the part of the test writer is reduced due to the familiar and natural way of using the language. Ease of learning on the part of all users is enhanced by an English-like format.

The selected approach is to use a limited number of explicit English-like statements, with a minimum of abbreviations on the final test output.

The ALOFT language is, as a result, understandable with little training and has few special rules which need to be learned by users, yet it is manageable by practical language processors.

#### 3. Self-extension Capability

A self-extension capability is implemented in the language. This self-extension capability is primarily provided for the use of the sophisticated test programmer who takes the time required to study how the language may provide powerful assistance in the accomplishment of his particular task. It is not intended that this capability be used by the less sophisticated test writer and in no way should detract from his ability to use the more straight forward portions of the language. Some project control of the use of language extension capabilities may be desirable.

The selected approach is to make the language extensible through subroutines, macros, and declarations, with all extensions using existing capabilities as elements. If necessary, another language can also be inserted.

This approach provides for a language processor that can be fixed but still be capable of reuse on different projects. All extensions are defined in terms of the basic language, so that retraining

is not required for the use of extensions. The language can, therefore, accommodate project changes, system evolution, new programs, etc.

#### 4. Self-documenting Capability

Programs written in the language are explicit and invariable as to the intent of all actions. As such, they are useful and sufficient as test definition documents.

This provides a single source of documentation, without the possibility of deviation of the actual program from the specification and/or commentary. As a result, fewer documents are subject to configuration control, review, approval, etc.

This capability is accomplished through the syntax design and the language elements themselves. Comments are also allowed in any statement where multiple blanks may appear. The use of comments in this way will allow the writer to clarify any statement that may not be completely clear as a result of its elements and syntax.

#### 5. Safing Features

A capability within the language is provided which allows the test writer to create his own safing features.

Three approaches with respect to safing features can be identified. One is the inclusion in the language of the necessary capabilities to enable a test writer to create his own safing procedures which would be attached to the test he is currently writing versus the inclusion of a standard set of safing procedures either in the language itself, or as part of an operating system. Inclusion of a standard set of safing procedures in either the language or an operating system is difficult to do prior to the establishment of the actual operating hardware of the checkout system. Since the language is independent of any particular test system, it is necessary to provide to the test writer the capability to create his own safing procedures. Another advantage to this approach is that safing procedures can easily be modified when necessary by the creation of new procedures.

Safing procedures might be called into execution by the operator, by branches within a program, or by interrupts (which are discussed later).

This approach provides for flexibility and visibility of all safing routines, with any degree of control that the project may direct.

It also enables writers to optimize routines for specific applications and to understand safing routines prepared by others.

#### 6. User Program Maintenance

User program maintenance is facilitated by the naturalness of statement structure and the self-documentation capability of the language.

This function is generally not the responsibility of the test writer but the responsibility of the users of the tests. In any case any changes which are initiated to a test are subject to considerable review by a number of affected parties. This requires that such changes and the test itself be readily understood by all concerned. The engineering reader orientation and the self-documenting capability of the language are of primary assistance in this capacity.

### E. Selected Specific Language Characteristics

#### 1. Format

The ALOFT statement format is free form with respect to input media, and consists of fixed but natural English statement structures.

The meaning of language elements depends solely on their alphanumeric configuration and not on any specific orientation with respect to input media. Neither the writer nor the reader is required to recognize meaning based on the position of a language element. All meaning is, therefore, explicit in the statement.

This results in more easily understood print-outs and documentation.

Also, this approach does not restrict the language to any specific input/output media (cards, printers, etc.) or special coding sheets.

Fixed statement structures enable language processing simplification and decrease potential misinterpretations (ambiguities) in statement meanings. It is also more efficient than variable field order structures where each field must be self-identifying.

With respect to natural English statement structures, readers will be most comfortable with statements that appear in as natural a form as possible. The writer is also prone to error when he is required to write in an arbitrary format or an unstructured format. The latter, especially, can be prone to inadvertent omissions.

#### 2. Numeric and Relational Operators

The lack of an arithmetic calculation capability was identified as a deficiency in some of the test and checkout languages studied. In order to avoid this deficiency in the ALOFT language a capability for addition, subtraction, negation, multiplication, division, and exponentiation is provided.

The relational operators equal, not equal, greater than, not greater than, less than, not less than, between, and not between are provided.

These relational operators are necessary to aid in the expression of the various conditional statements, limit checks, and other forms of checks universally required in test and checkout languages.

### 3. Dictionary Data Banks

A dictionary data bank capability is available in the ALOFT language to provide the Line Replaceable Unit designers and the test equipment designers with the capability to declare the nouns and modifiers required to test a unit and to define the action of those nouns and modifiers with respect to the test system.

This requirement is necessary to provide the final link between the language and the test system. Such a link must be supplied in one way or another. The alternative to creating a language capability to define that link is to have a programmer generate machine language tables which provide the necessary information. These tables could be included in the language processor or operating system at the time the unit and test equipment have been designed. To avoid the use of a professional programmer to modify the language processor each time new LRUs (requiring new nouns and modifiers) and new test equipment are available for use, a language capability is provided.

This language capability provides for complete test system independence of both language and language processor. It will provide the capability required to interface tests written in the language with any test system.

A hierarchy of language users is necessary under the dictionary data bank concept. The LRU designer specifies the nouns and modifiers which are required to completely implement the test functions available in the language. The test equipment designer specifies the meaning of these nouns and modifiers with respect to the equipment which will actually test the device. In the case of Space Shuttle, for instance, a noun signifying pressure would have to be defined in terms of Interface Unit numbers and digital code words. This information is placed in a dictionary data bank, utilizing special language capabilities designed for this function.

When the test engineer writes his test he uses the functions available in the language along with the particular dictionary data bank he needs to provide him with all allowable nouns and modifiers which can be used in testing the particular device in which he is interested. He is in no way concerned with how the test system implements the meaning of these nouns and modifiers.

In short, the dictionary data bank provides the test writer with all necessary information with regard to the test article and also provides the interface between the language and any specific test system.

The advantages of this concept are:

The language is test system independent.

The language processors can be developed before any specific test system is defined.

Tests can be created before a specific test system is defined.

The dictionary data bank is created for the use of the test writers by test system designers.

The dictionary data bank provides common information to be used by a number of separately processed tests.

#### 4. Subroutine Structures

A subroutine capability is provided in the language.

This capability is a powerful aid for specifying those functions which are repeated many times. It is both a convenience to the writer in reducing his writing task and assists the reader by isolating and clearly specifying those functions which are of a repeatable nature.

A subroutine capability also allows the creation of a library of common sets of actions for use by all test writers.

This is a programming oriented capability provided for the use of test engineers. Most test oriented languages have some form of subroutine capability.

#### 5. Interrupt Initiated Routines

A capability is provided to initiate the execution of a subroutine as a result of an interrupt.

This includes an inhibit/enable interrupt capability which allows the test writer to control the action of those interrupts which affect the operations of his test.

This capability provides a test writer with the ability to respond to an interrupt which may affect the operation of his test. These interrupts may be interrupts that specify that certain error conditions or hardware status changes have been generated in the device under test, over and above those conditions which can be determined in the normal course of testing.

As a result, back out and safing subroutines can be established for execution when such hardware signals occur.

#### 6. Define-Type Capability

A define-type capability is provided as a writing aid. In essence, this capability provides a writer with the ability to create within the language a set of abbreviations for language elements and combinations of language elements and statements. The define statement will help the writer to both minimize the possibility of error in repeating long strings of language elements and will also ease the writing task. The task of the reader is not compromised however, since a compiler will produce full listings with proper substitutions for all abbreviated portions of statements.

#### 7. Concurrent Testing Capability

A special set of language elements to facilitate concurrent testing, along with simple rules for their use, is designed into the language. Such multiple programming features do not overly complicate the language or its compiler but provisions for concurrent testing must be included in the executive programs.

#### 8. Monitoring

A language capability is provided to enable a check to be utilized in a continuous monitor mode.

This capability is necessary to allow the continuous monitoring of systems. As long as no anomalies occur, little notice is attached to the monitored systems. However, if an anomaly is detected, a previously defined warning, alternate action or a backout routine provides corrective action.

With the capability for concurrent test execution existing in the language, monitor tests can be continuously executed while other tests are run on a noncontinuous basis.

A monitor test differs from a normal test only in that a way of specifying repeatable execution exists for the monitor test.

Continuous monitoring is a vital portion of most space system and booster test programs. As a test function, it belongs in the language to insure an integrated systems test approach.

#### 9. Special Discipline Provisions

Special discipline provisions within the language are confined to words which identify special characteristics which are attached to declared data items.

This approach removes special discipline provisions from the test function language elements which are designed for the general testing problem. It confines these characteristics to data which represent the subsystems and LRUs under test.

The language, therefore, provides the capability for specifying functions peculiar to each avionic discipline to insure that the test writer has terms to use with which he is familiar.

The names of the functions and characteristics of test article are specified by the people most closely related to the design of the system to be tested.



## 10. Test Level

The language is capable of defining tests at all levels; system, subsystem, unit and sub-unit.

The use of the same language at all levels will facilitate the preparation and verification of test programs because the writers and readers can directly use and compare performance parameters, etc. In addition, the separate programs can utilize common definitions, subroutines, and libraries when they are applicable. The subsystem test engineers can readily verify performance of the subsystem and units when involved in higher level tests. Common language processors can be used.

## 11. Program (Project) Orientation

The language described in this paper is capable of being used not only for Space Shuttle but for test and checkout of other advanced space vehicles and systems.

The language characteristics have been developed as a result of study of previously designed test languages and a knowledge of the current Space Shuttle configuration. Attention has been paid to the general test and checkout problem and the generalized needs identified as a result have been considered in establishing the characteristics of this language. The inherent flexibility and power of the language as currently envisioned, along with its self-extension capability, should enable it to be readily applied in test and checkout of other systems besides Space Shuttle.

## PART II

### A. Language Overview

A very flexible, yet unambiguous structure is provided for the ALOFT language. A minimum number of rules and restraints are imposed on the user.

A basic English-like statement structure is used for test action statements. It has the form:

*When - do what - to what*

The *when* permits a time statement to define when the desired action is to take place.

The *do what* defines the action that is to take place. To meet the specific needs of the many disciplines involved in advanced space projects, a variety of action words are necessary. The language provides this capability. Typical are such verbs as measure, verify, apply, set, turn, send, display, print, etc. With this variety of verbs the user is able to select terms that most accurately describe the action.

The *to what* identifies the name of the unit under test function undergoing the action. The *name* is defined in the dictionary data bank. For any given test program and test system these *names* are defined in terms that are meaningful in relation

to the article under test. Provisions are included to enable these functions (which will appear in signal lists, schematics, etc.) to be so defined.

These defined *names* are placed in the dictionary using SPECIFY statements. The test writer is confined to the use of function *names* which must eventually appear in the dictionary.

The dictionary also facilitates the problem of identifying the calling addresses of Space Shuttle systems, subsystems, LRUs, etc. The redundant data bus concept of the Space Shuttle requires all addressable items to be identified by their data bus and interface unit (IU) numbers. The data bus, IU, and function codes are identified at the same time the function *name* is placed in the data dictionary.

Typical examples of the language, ready for compiling are:

```
STATEMENT 80 AFTER CDC 'COUNT DOWN CLOCK' IS  
-10MIN 50SEC,
```

```
MEASURE RIGHT AILERON 2 POSITION_ AND SAVE AS  
_AILERON POSITION_ .
```

```
IF AILERON POSITION_ IS BETWEEN 10PCT AND  
20PCT GO TO STATEMENT 120.
```

```
STATEMENT 120 DISPLAY TEST (RIGHT AILERON 2  
IN CORRECT POSITION) ON CRT 2, LINE 23_ .
```

As is readily seen, the language is very readable. All specially defined items are delimited by means of underscores. The compiler obtains the address for such information from the dictionary data bank.

Comments such as "COUNT DOWN CLOCK" which are not to be compiled, are delimited by dual apostrophes.

Test to be printed or displayed such as "RIGHT AILERON 2 IN CORRECT POSITION" is delimited by open and closed parentheses.

If the dictionary were not provided, it would become necessary for the test writer to define addresses while writing the test procedure. The procedure would then be more subject to error and the printed address data would impair readability.

### B. Language Specification Summary

#### 1. General

The specification of the ALOFT language<sup>1</sup> uses syntax diagrams to illustrate the construction of all legal elements of the language from the basic characters and symbols to complete programs. This technique was chosen because it is precise, minimizes the ambiguities associated with prose descriptions, is more condensed than prose descriptions, and facilitates rapid comprehension of alternative statement constructions.

## 2. Basic ALOFT Statements and Statement Prefixes

As with English and most higher level programming languages, the lowest meaningful and complete element of ALOFT is a statement. Within ALOFT statements, words and phrases are inserted to help readability and prevent misinterpretations and errors by users. These are generally verbs, articles, prepositions, etc., which make the statements English-like. They are required to be used precisely as shown in the syntax diagrams. The complete statement, rather than a single words, defines the action or purpose of the statement. In general, however, a *verb* or *operation code* in the statement is a very strong indication of the type of activity or purpose of the statement. In addition to basic statements, ALOFT has provisions for including optional *prefix phrases*, which may be either a condition for execution of the statement or an action to be performed at essentially the same time.

The basic statement and prefix phrase types, as indicated by their key words, are listed below. The parenthetical notes are included to further explain the associated actions.

### Send actions

APPLY	(Analog or digital function)
SET	(Discrete functions, valves, clocks)
TURN	(Discrete functions on or off)
SEND	(Digital data)

### Acquire actions

READ and SAVE	(Discretes, clocks, digital)
MEASURE and SAVE	(Analog, digital)
VERIFY	(Read or measure with conditional transfer)

### Invocations or calling statements

PERFORM	(Subroutine)
PERFORM PROGRAM	(Program)
EXECUTE	(For macros only)
USE	(Data bank)

### Delimiters

BEGIN	(Program, data bank, sub-routine)
MACRO	(Beginning of macro definition)
COMPLETE	(Program, data bank)
LEAVE ALOFT	(To use another language)
RESUME ALOFT	(To return from another language)

### Interrupt manipulation

WHEN INTERRUPT	(To identify the action to occur as a result of a named interrupt)
ENABLE	(Interrupt)
DISABLE	(Interrupt)

### Sequence control

GO TO	(Unconditional transfer)
IF	(Variable reference conditional transfer)
VERIFY	(Function conditional transfer)
WHEN INTERRUPT	(See above)
REPEAT	(Single statement)

### Assignment or arithmetic operation

LET	(Variable reference) = (Value or formula)
ASSIGN	(Variable reference) = (Discrete or Boolean state)

### Concurrent program implementation

CONCURRENTLY PERFORM	(For concurrent programs)
SYNCHRONIZE (n)	(Synchronization points in each program)

### Prefix phrases and timing control

WHEN (Clock=time)	(Precedes action statement)
SET (Clock=time), AND	(Precedes action statement)
AFTER (Clock=time),	(Precedes action statement)
STATEMENT (number)	(Statement label where required)
S (number)	(Statement label where required)

### Other time phrases

--WITHIN (Time value)	(To set a time limit for VERIFY)
--FOR (Time value)	(To generate a timed discrete or pulse)

### Operator interfaces and records

DISPLAY	(Messages)
INDICATE	(Lights or fixed states)
PRINT	(Variable messages)
RECORD	(Variable messages)
REQUEST	(DISPLAY message then READ and SAVE keyboard input)

### Definition statements

SPECIFY	(Function)
DECLARE	(Table, list, internal variable)
BEGIN	(Subroutine, program, data bank)
REPLACE	(Abbreviation, Substitution)
MACRO	(Macros)

### Miscellaneous

ACTIVATE --	(Acknowledge or honor a function in a table)
DEACTIVATE	(Ignore a function in a table)

## C. Sample Syntax Diagrams

### 1. Format of Language Syntax Diagrams

The syntax diagrams for ALOFT are modeled after the syntax diagrams found in the Abbreviated Test Language for Avionics Systems (ATLAS), ARINC Specification 416-1, June 1, 1969.

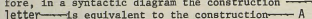
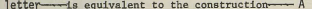
This form of syntax diagram was chosen over alternate forms due to its readability, clarity, and precision. It is a type of syntax diagram that can be easily learned by engineering personnel and has been proven in field use.

The format of presentation used in the ALOFT specification is the syntactic diagram followed by an explanation of the semantics of the illustrated diagram. This combination constitutes a full definition of the structure and meaning of a language form.

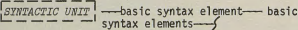
### 2. Explanation of Language Syntax Diagrams

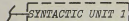
Syntax diagrams are made up of syntactic units and basic syntax elements that ultimately reduce to the allowable letters, numerals, and symbols which make up the character set of the language. The basic syntax elements appear in syntactic diagrams as themselves or as a name that is syntactically equivalent. The syntactically equivalent name appears in lower case type. For example:

letter :: = A

where "::=" means syntactic equivalence. Therefore, in a syntactic diagram the construction  is equivalent to the construction 

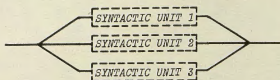
A name enclosed in a dashed box is a syntactic unit defined from basic syntax elements and/or other syntactic units. A definition consists of a name within a dashed box on the left and a syntax diagram on the right. For example:





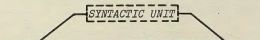
The syntax diagram in the example indicates that the syntactic unit being defined on the left is a concatenation of two basic syntax elements with a previously defined syntactic unit. The lines indicate the flow of the syntax diagram from left to right. The wavy lines indicate continuation of a syntax flow from one line on the page to the next lower line. Assume that the dashed box indicated by the name syntactic unit 1 has previously been defined as CD. Further assume that the first basic syntax element is syntactically equivalent to A and the second basic syntax element is syntactically equivalent to B. Therefore, the syntactic unit being defined on the left, is reducible to the basic syntax elements forming the character string ABCD.

Choice among syntactic units is indicated by a branching in the syntax diagram. For example:

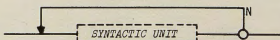


The flow of the syntax diagram illustrated allows only one branch to be taken, which results in a single syntactic unit being chosen from the three syntactic units available.

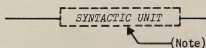
A choice between taking or omitting a syntactic unit is indicated by a branch in the syntax flow that contains no syntactic unit. For example:



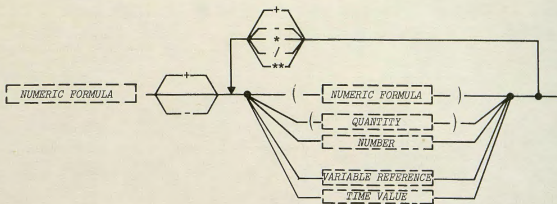
Repetition of syntactic units is indicated by a feedback loop with the maximum number of repetitions, if applicable, indicated on the loop arrow. Otherwise, the number of repetitions is undefined. A syntactic unit on a line that is part of a feedback loop must appear at least once in the corresponding statement for which the syntax diagram exists. For example:



Notes that give further information on a syntax diagram appear in parentheses beneath the diagram, with an arrow indicating where in the diagram the note is to be applied. For example:



To further illustrate these concepts, the following selected sample diagrams from the ALOFT specification are presented:

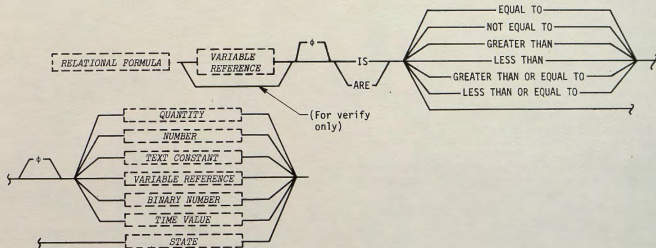


The syntactic unit "NUMERIC FORMULA" provides a syntactical structure for the expression of arithmetic calculations. The meaning of the symbols included in the syntax are:

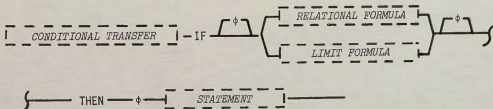
+, preceding the feedback loop, is unary positive;  
 -, preceding the feedback loop, is negation;  
 +, inside the feedback loop, is addition;  
 -, inside the feedback loop, is subtraction;

\* is multiplication;  
 / is division;  
 \*\* is exponentiation;

Parentheses enclose numeric formulas used within numeric formulas, where necessary, and also enclose quantities so as to delimit dimensional information to alleviate confusion of dimensional symbols and arithmetic symbols.

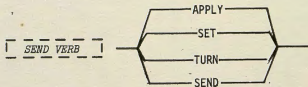


The syntactic unit "RELATIONAL FORMULA" provides a syntactical structure for the expression of relationships between variables or between variables and data constants.

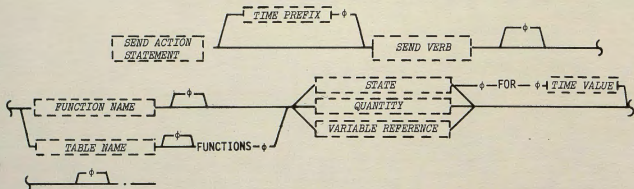


The syntactic unit "CONDITIONAL TRANSFER" provides a syntactical structure for the optional execution of a statement. The optional nature of the statement execution is provided by imbedding in the conditional transfer statement a relational formula or a limit formula. When the result of the evaluation of these imbedded syntactic units is "true", the statement following the "THEN" is executed. Otherwise, the statement is skipped and the next statement after the conditional transfer is executed. The statement following the "THEN" may often be an unconditional transfer.

#### Send Action Syntax



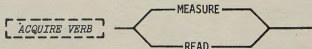
The syntactic unit "SEND VERB" provides terms for use in describing the send actions performed in send action statements, defined below.



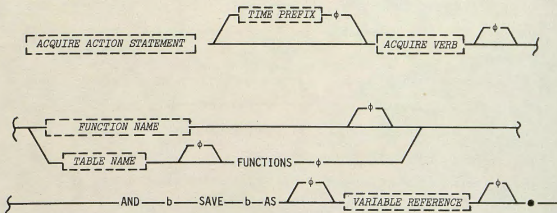
The syntactic unit "SEND ACTION STATEMENT" provides a syntactic structure for the performance of stimulus actions in a test. A time prefix may be attached to the send action statement. The function name is provided to the test writer from a dictionary data bank. The state of a discrete, a numeric quantity, or a variable reference identifying a numeric quantity to be sent to an LRU can be identified. In the case of a discrete state, a time limit for the application of that discrete state may be established. At the end of the time specified, the discrete state will be reversed.

If a table name is identified instead of a function name, the function of each row of the table is sent with the appropriate values as identified by the state, quantity, or variable reference.

#### Acquire Action Syntax



The syntactic unit "ACQUIRE VERB" provides terms for use in describing the acquire actions performed in acquire action statements, defined below.



The syntactic unit "ACQUIRE ACTION STATEMENT" provides a syntactic structure for the performance of measurement actions in a test. A time prefix may be attached to the acquire action statement. The function name is provided to the test writer from a dictionary data bank. The information acquired by the action of this statement is retained, for later use in the test, in the variable reference identified.

#### D. Sample ALOFT Statements

In the course of development of ALOFT a number of programs were written in the language. One of these programs was a routine called "KAF2 Flight Control Preps Program". This routine was originally written in ATOLL and used in the checkout of the Saturn launch vehicle. Statements from this program are included here to provide an example of ALOFT usage.

```
BEGIN PROGRAM_KAF2_**FLIGHT CONTROL PREPARATIONS FOR A5509**.
```

```
USE DICTIONARY DATA BANK_KAF2 DISC OUTPUTS TO VEH _*
    _ DDAS SIGNAL FUNCTIONS _*
    _ DISCRETE I/O FROM ESE PANELS_*
    _ INPUT/OUTPUT DEVICES _.
```

```
DECLARE _CSP POWER ON TIME_ TIME.                **SC0W**
DECLARE _GR-1 UP-TO-SPEED INDICATION TIME_ TIME.  **SC0X**
DECLARE _GR-2 UP-TO-SPEED INDICATION TIME_ TIME.  **SC0Y**
DECLARE _GR-3 UP-TO-SPEED INDICATION TIME_ TIME.  **SC0Z**
DECLARE _FCC POWER ON TIME_ TIME.                 **SC0V**
DECLARF _T2_ TIME.
```

⋮

DECLARE\_FC FLAG TABLE\_WITH 9 COLUMNS INDEXED BY\_SC\_AND LABELED  
 ROW NUMBER, FUNCTION, UNITS, \_ST1\_BOOLEAN, \_ST2\_BOOLEAN, \_ST3\_  
 BOOLEAN, \_ST4\_BOOLEAN, \_ST5\_BOOLEAN, \_ST6\_BOOLEAN, HAVING 8 ROWS  
 INDEXED BY\_FR\_WITH ENTRIES

**FR	FUNCTION	UNITS	ST1	ST2	ST3	ST4	ST5	ST6	**
1.	_FLAG 25_	ON/OFF,	ON,	OFF,	OFF,	OFF,	OFF,	OFF	AND
2.	_FLAG 26_	ON/OFF,	OFF,	ON,	OFF,	OFF,	OFF,	OFF	AND
3.	_FLAG 27_	ON/OFF,	OFF,	OFF,	OFF,	OFF,	OFF,	OFF	AND
4.	_FLAG 28_	ON/OFF,	OFF,	OFF,	OFF,	OFF,	OFF,	OFF	AND
5.	_FLAG 37_	ON/OFF,	OFF,	OFF,	OFF,	ON,	OFF,	OFF	AND
6.	_FLAG 38_	ON/OFF,	OFF,	OFF,	OFF,	OFF,	ON,	OFF	AND
7.	_FLAG 39_	ON/OFF,	OFF,	OFF,	OFF,	OFF,	OFF,	ON	AND
8.	_FLAG 47_	ON/OFF,	ON,	ON,	ON,	ON,	ON,	ON	.

⋮

S000100 BEGIN CRITICAL \_TERMINATION SUBROUTINE\_ WITH INPUT \_TERM TABLE\_  
 S000200 APPLY \_TERM TABLE\_ FUNCTIONS \_STATE\_  
 S000300 DISPLAY \_PROG NAME\_, TEXT (HAS BEEN FORCIBLY TERMINATED) ON  
 \_CONSOLE CODE\_  
 S000400 END CRITICAL \_TERMINATION SUBROUTINE\_

⋮

S100000 WHEN INTERRUPT \_TERMINATE\_ OCCURS THEN PERFORM \_TERMINATION SUBROUTINE\_  
 WITH INPUT \_KAF2 TERM FUNCTIONS\_

S100100 ENABLE \_TERMINATE\_

⋮

```

S300000 ACTIVATE_FC PREPS SCAN_ALL.
S300100 IF_FLAG 1_IS ON GO TO S300500.
        LET_RN_=1.
S300110 DEACTIVATE_FC PREPS SCAN_ROW(_RN_).
        LET_RN=_RN_*1.
        IF_RN_IS LESS THAN 36 THEN GO TO S300110.
        GO TO S300600.
S300500 IF_FLAG 2_IS ON GO TO S300600.
        LET_RN_=36.
S300510 DEACTIVATE_FCC PREPS SCAN_ROW(_RN_).
        LET_RN=_RN_*1.
        IF_RN_IS LESS THAN 52 GO TO S300510.
S300600 VERIFY_FC PREPS SCAN_FUNCTIONS ARE EQUAL TO_STATE_OTHERWISE GO TO
        S600000.
S300700 IF_FLAG 1_ IS OFF THEN GO TO S301500.
S300800 VERIFY _ FCC/ON/+6011 _ IS OFF OTHERWISE GOTO S327200.
S300900 READ GMT INTO _FCC POWER ON TIME_.
S301000 TURN _ IU FCC SYSTEM PWR _ ON.
S301100 DISPLAY _CRT 1 CLEAR_.
S301200 DISPLAY TEXT ( IU FCC SYSTEM PWR ON ) ON _CRT 1,LINE 1_.
S301300 ASSIGN _FLAG 6_ **FCC POWERED ON BY PROGRAM** ON.
S301400 IF_FLAG 2_ IS OFF THEN GOTO S302100.
S301500 VERIFY _ CSP/POWER/ON _ IS OFF OTHERWISE GOTO S327700.
S301600 READ GMT INTO _CSP POWER ON TIME_.
S301700 TURN _ IU EDS RG SYS POWER _ ON.
S301800 DISPLAY TEXT (IU EDS RG SYS POWER ON) ON _CRT 1,LINE 2_.
S301900 ASSIGN _FLAG 5_ **CSP POWERED ON BY PROGRAM** ON.
S302000 IF_FLAG 1_ IS OFF THEN GOTO S304500.

```

\*\*MD01823\*\*

\*\*MD01903\*\*

⋮



```

S321300 DISPLAY TEXT (GROUP 3,UP TO SPEED) ON _CRT 1,LINE 11_.
S321400 DISPLAY _GR-3 UP-TO-SPEED INDICATION TIME_ ON _CRT 1,LINE 12_.
S321500 PRINT TEXT (GROUP 3,UP-TO-SPEED TIME), _GR-3 UP-TO-SPEED INDICATION
      TIME_ ON _PRINTER_.
S321600 RECORD TEXT (GROUP 3 UP-TO-SPEED TIME), _GR-3 UP-TO-SPEED INDICATION
      TIME_ ON _MAG TAPE_.
S321700 TURN _ IU EDS RG ROLL AXIS SEL _ OFF.                **M001904**
S321800 TURN _ IU EDS RG YAW AXIS SEL _ OFF.                 **M001905**
S321900 TURN _ IU EDS RG PITCH AXIS SEL _ OFF.              **M001906**
S322000 TURN _ IU EDS RG REF GYRO SEL _ OFF.                 **M001907**
S322100 TURN _ IU EDS RG CMD GYRO SEL _ OFF.                 **M001909**
S322200 TURN _ IU EDS RG SPARE GYRO SEL _ OFF.              **M001910**
S322300 IF_FLAG 1_IS OFF THEN GOTO S323600 .**FCC OPTION NOT SELECTED**

```

⋮

A further example of ALOFT usage is provided by this example subroutine and its use.

⋮

```

BEGIN_ADJUST_WITH INPUTS_VALUE OF X_+_FINAL VALUE_+_ADJUST FUNCTION_
      AND_FUNCTION OF X_AND OUTPUT_RESULT_.

```

```

DECLARE_Y_NUMERIC.
DECLARE_VALUE OF X_NUMERIC.
DECLARE_FINAL VALUE_NUMERIC.
DECLARE_RESULT_NUMERIC.

```

```

LET_RESULT_EQUAL 0.
SET CLOCK 1 TO 0MSEC, AND
SEND_ADJUST FUNCTION_**THE**_VALUE OF X_.
AFTER CLOCK 1 IS 5MSEC,
MEASURE_FUNCTION OF X_AND SAVE AS_Y_+.
IF_Y_IS GREATER THAN OR EQUAL TO_FINAL VALUE_ THEN
LET_RESULT_EQUAL_VALUE OF X_.

```

```

END_ADJUST_.

```

⋮

\*\*THE FOLLOWING IS A PORTION OF THE PROGRAM USING THE PREVIOUSLY DEFINED SUBROUTINE\_ADJUST\_, AS IT WOULD BE WRITTEN AND AS IT WOULD APPEAR ON A FINAL LISTING. DECLARATIONS AND SPECIFICATIONS REQUIRED ARE ASSUMED.\*\*

```
      **OTHER
      STATEMENTS**
      LET_START_EQUAL 5.0V.
STATEMENT 100 PERFORM_ADJUST_WITH INPUTS_START_+55.0DEG+_POSITION DRIVER_
      AND_POSITION_AND OUTPUT_VOLTIN_.
      IF_VOLTIN_IS NOT EQUAL TO 0 THEN GOTO STATEMENT 101.
      LET_START_EQUAL_START_+1.0V.
      GOTO STATEMENT 100.
STATEMENT 101 **PROGRAM CONTINUES**
      **OTHER
      STATEMENTS**
      LET_VALUE_SENT_EQUAL 24.0INHG.
STATEMENT 200 PERFORM_ADJUST_WITH INPUTS_VALUE_SENT_+110.0DEGF+_PRESSURE_
      AND_TEMPERATURE_AND OUTPUT_TOTAL PRESS_.
      IF_TOTAL PRESS_IS NOT EQUAL TO 0 THEN GOTO STATEMENT 201.
      LET_VALUE_SENT_EQUAL_VALUE_SENT_+2.0INHG.
      GOTO STATEMENT 200.
STATEMENT 201 **PROGRAM CONTINUES**
**AT EACH PERFORM_ADJUST_STATEMENT CONTROL WOULD BE TRANSFERRED TO THE
PREVIOUSLY DEFINED_ADJUST_SUBROUTINE WITH THE APPROPRIATE INFORMATION
AS INDICATED IN THE PERFORM STATEMENT. WHEN THE SUBROUTINE IS COMPLETE,
CONTROL IS RETURNED TO THE STATEMENT FOLLOWING THE PERFORM STATEMENT.
THIS ACTIVITY OCCURS AT RUN TIME.**
```

#### ACKNOWLEDGEMENT

The authors would like to acknowledge the extensive contributions of C. W. Case and E. L. Kinney toward the development of the ALOFT language and especially for the contribution of many of the ideas which have been presented in this paper.

#### REFERENCE

ALOFT was developed for NASA, KSC under contract NAS10-7308. Mr. Henry Paul, LV-CAP was the Technical Monitor.

#### BIBLIOGRAPHY

- (1) Specification for ALOFT, A Language Oriented to Flight Engineering and Testing, MCR-70-450, Martin Marietta Corp., December, 1970.
- (2) Development of a KSC Test and Flight Engineering Oriented Language - Phase I Report, MCR-70-327, Martin Marietta Corp., August, 1970.
- (3) Development of a KSC Test and Flight Engineering Oriented Language - Phase II Report, MCR-70-365, Martin Marietta Corp., October, 1970.
- (4) Development of a KSC Test and Flight Engineering Oriented Language - Phase III Report, MCR-70-424, Martin Marietta Corp., December, 1970.