

IEEE Guide to the USE of the ATLAS Specification

Sponsor

ATLAS 771 Sub Committee
of the
IEEE Standards Coordinating Committee 20

Reaffirmed 9 December 2009

Approved 19 March 1998

IEEE-SA Standards Board

Abstract: Guidance in the use of ATLAS test languages is provided. ATLAS may be used to describe test requirements independent of any specific test equipment, and examples of best practice in the use of ATLAS are given.

Keywords: ATE, ATLAS, automatic test equipment, C/ATLAS, guide, test language, test requirement unit under test, UUT

The Institute of Electrical and Electronics Engineers, Inc.
345 Park Avenue, New York, NY 10017-2394, USA

Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 30 November 1998. Printed in the United States of America.

Print: ISBN 0-7381-0195-8 STD94524
PDF: ISBN 973-07381-5860-0 STDPD94624

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS**.”

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

(This introduction is not part of IEEE Std 771-1998, IEEE Guide to the Use of the ATLAS Specification.)

IEEE Std 716-1995, IEEE Standard Test Language for All Systems—Common Abbreviated Test Language for All Systems (C/ATLAS), and ARINC Specification 626-3, Standard ATLAS Language for Modular Test, define ATLAS, a standardized test language for expressing test specifications and procedures. This guide supplements IEEE Std 716-1995 and ARINC Specification 626-3 and is part of the documentation set for the language.

The need for a guide to the use of ATLAS arose from the experience of users in the preparation of test procedures in the language. This experience showed that differences of interpretation existed for technique and method, if not definition. The purpose of this guide is to provide guidance in the matter of technique and usage; it is seen as particularly important in promoting the use of the language envisaged by its originators. This document is not intended as an instruction manual for ATLAS nor as a substitute for some formal training, but by its nature it should find some applicability in the training environment.

IEEE Std 771-1998 corresponds with IEEE Std 716-1995 and ARINC Specification 626-3. It is the intention of the IEEE ATLAS Committee that there shall be a continued correspondence between these standards and IEEE Std 771.

The members of the ATLAS 771 Subcommittee during the preparation of this guide were as follows:

Ashley M. B. Hulme, *Chair*

Jean-Yves Barbier
Antonius Bunsen
Barry Burnage
Anthony Fitzpatrick

Arthur Hann
Bill Heather
Linda O. Lund
Neil A. McKinnon

Goran Nordstrom
Michael Seavey
Joseph Stanco
Anthony A. F. Vogelpoel

In addition, acknowledgment is made of the valuable individual contributions made by the following:

Michael Blair
William Borah
Patrice Brard

George J. Diab
Hans Hopf
Hans Lecluyse

Larry Lusz
William E. Reeves
Helmut Scheibenzuber

The following members of the balloting committee voted on this standard:

Guy Adam
Jean-Yves Barbier
Christos Bezirtzoglou
Michael Blair
Antonius Bunsen
Larry D. Carpenter
Richard D. Churcher
James E. Cottrell
Michael T. Ellis

Anthony Fitzpatrick
Arnold M. Greenspan
Arthur Hann
Michael G. Harrison
John E. Heiser
Ashley M. B. Hulme
Frank Kearns
Jean Pouilly

William E. Reeves
Michael L. Seavey
John W. Sheppard
William R. Simpson
Joseph J. Stanco
Kirk D. Thompson
Louis Y. Ungar
Richard J. Weger
Will W. Young

When the IEEE-SA Standards Board approved this standard on 19 March 1998, it had the following membership:

Richard J. Holleman, *Chair*

Donald N. Heirman, *Vice Chair*

Judith Gorman, *Secretary*

Satish K. Aggarwal
Clyde R. Camp
James T. Carlo
Gary R. Engmann
Harold E. Epstein
Jay Forster*
Thomas F. Garrity
Ruben D. Garzon

James H. Gurney
Jim D. Isaak
Lowell G. Johnson
Robert Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
Stephen R. Lambert
Jim Logothetis
Donald C. Loughry

L. Bruce McClung
Louis-François Pau
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Hans E. Weinrich
Donald W. Zipse

*Member Emeritus

Kim Breitfelder
IEEE Standards Project Editor

Contents

1.	Overview.....	1
1.1	Scope.....	1
1.2	Purpose.....	1
1.3	Related documents	1
2.	References.....	2
3.	Definitions.....	2
4.	The need for a test language standard.....	4
4.1	The importance of unambiguous communication.....	4
4.2	Types of languages	4
4.3	Need for unambiguous language	6
4.4	The portability of ATLAS	6
4.5	ATLAS development	7
4.6	Continued regulated growth of ATLAS	11
4.7	ATLAS as an aid to the management of testing	12
4.8	Benefits of a standard.....	12
5.	The current ATLAS standards	12
5.1	Definition of ATLAS	12
5.2	IEEE Std 716-1995	12
5.3	ARINC Specification 626-3.....	13
5.4	IEEE Std 771-1998	13
6.	Structure and characteristics of ATLAS.....	13
6.1	General definition	13
6.2	General characteristics of ATLAS.....	13
6.3	Levels of ATLAS.....	13
6.4	ATLAS test requirement structures	14
6.5	FSD	19
6.6	Non-ATLAS usage	20
7.	The elements of ATLAS.....	20
7.1	The ATLAS character set	20
7.2	ATLAS keywords	23
7.3	ATLAS number conventions	23
7.4	User-defined labels	26
7.5	Types of ATLAS data.....	27
7.6	Constants.....	29
7.7	Variables	29
7.8	Initializing variables.....	30
7.9	Connections.....	31
7.10	Classification of the language elements.....	31
7.11	Verbs	31
7.12	Nouns and modifiers	37

8.	Writing a test requirement in ATLAS	37
8.1	Introduction.....	37
8.2	Analysis of the test problem	37
8.3	Segmenting the test requirement.....	38
8.4	Flowcharting	38
8.5	Logical branching	40
8.6	Test resources.....	42
8.7	Arrangement of test requirement	42
8.8	Using procedures	44
8.9	Instructions to and from the operator.....	45
8.10	Applying power to the UUT	48
8.11	Resource allocation.....	49
8.12	Identifying signals with resources	50
8.13	Applying analog test signals	50
8.14	Applying digital signals	52
8.15	Sensing UUT responses	52
8.16	Statement execution rate	54
8.17	Postponing statement execution.....	54
8.18	Branching.....	58
8.19	Entry points.....	58
8.20	Iteration in testing	59
8.21	Multiple-action verbs	65
8.22	Time-out fields.....	65
8.23	Test problems not explicitly covered by ATLAS vocabulary	66
8.24	Use of comments.....	67
8.25	Shutting down a UUT	68
8.26	Recommended practices for good ATLAS.....	69
9.	Documentation and structuring of test requirements	72
9.1	Documentation of test requirements	72
9.2	UUT	72
9.3	Design requirement.....	73
9.4	Production test procedure	74
9.5	ATLAS test specification.....	77
10.	ATLAS elements	102
10.1	Introduction.....	102
10.2	ATLAS verbs	102
10.3	ATLAS nouns	103
10.4	ATLAS noun modifiers	105
10.5	ATLAS units.....	112
11.	Glossary of ATLAS keywords.....	119
12.	ATLAS problems and proposals.....	164
12.1	Submitting an ATLAS problem.....	164
12.2	Submitting an ATLAS proposal	165
12.3	Submitting an ATLAS extension.....	166
12.4	Involvement with ATLAS standards activities.....	167

Annex A (informative) ATLAS in analog testing	168
Annex B (informative) ATLAS in digital testing.....	180
Annex C (informative) ATLAS in air navigation system testing.....	196
Annex D (informative) Timing and synchronization in ATLAS	223
Annex E (informative) ATLAS in data-bus testing.....	242
Annex F (informative) The ATLAS escape mechanism	254
Annex G (informative) Complex signals in ATLAS	260

IEEE Guide to the Use of the ATLAS Specification

1. Overview

This guide is divided into clauses that introduce the history, use, and structure of the Abbreviated Test Language for All Systems (ATLAS) language followed by general guidance on good usage practice. The main clauses also contain information and language elements that apply to all test disciplines.

The guide also contains annexes that provide more specific examples of ATLAS usage as applied to particular test disciplines. The information in each annex includes language elements and structures used in a particular test discipline with some examples and descriptions of usage.

1.1 Scope

This guide has been written as an applications manual for engineers and others who write test requirements in ATLAS. The initial clauses are intended to provide managers with a brief introduction to the application of ATLAS. This guide is in no way intended to be a substitute for an instruction manual in the ATLAS language; rather it is intended to provide practical support to such a manual.

1.2 Purpose

The purpose of this guide is to provide principles of good practice in the use of ATLAS for writing test requirements. This advice has been drawn from practical applications in many organizations and a number of companies that generate test requirements in ATLAS.

Since ATLAS must be capable of communicating complex test requirements, the language has a large vocabulary and appropriate syntax. The guide concentrates on the proper use of the ATLAS syntax and semantics for the generation of unambiguous test requirements. The guide assists in preparing test requirements as well as in writing computer software that will efficiently process ATLAS test requirements for automatic implementation.

1.3 Related documents

The material in this guide is applicable to IEEE Std 716-1995 and ARINC Specification 626-3. Supplements to these standards will be issued periodically and the users guide updated accordingly.

2. References

This guide shall be used in conjunction with the following publications. When the following standards are superseded by a revision, the revision shall apply.

ARINC Specification 626-3, Standard ATLAS Language for Modular Test.¹

IEEE Std 716-1995, IEEE Standard Test Language for All Systems—Common/Abbreviated Test Language for All Systems (C/ATLAS).²

ISO 646:1991, Information technology—ISO 7-bit coded character set for information interchange.³

ISO 1000:1992, SI units and recommendations for the use of their multiples and of certain other units.

3. Definitions

This clause contains key terms as they are used in this guide.

3.1 argand plane: A graphical representation of a vector used in complex number notation.

3.2 ATLAS compiler: Software that converts ATLAS statements into executable machine code which may involve more than one operation.

3.3 ATLAS translator: Software that converts ATLAS statements into another language. This language may be a computer language or a restricted and structured version of a natural language.

3.4 ATLAS vocabulary: The range of words and symbols used in standard ATLAS.

3.5 automatic test equipment (ATE): Equipment on which an implementation of the tests, test methods, and test sequences to be performed on a unit under test (UUT) to verify conformance with its test specification with or without fault diagnosis may be executed with minimum reliance on human intervention.

3.6 Backus-Naur format (BNF): A particular metalanguage developed by Backus and Naur.

3.7 Backus-Naur notation (BNN): A general term relating to metalanguages that use the concepts developed by Backus and Naur.

3.8 boolean: A variable that can assume only two states, true or false.

3.9 delimiter: A character that provides punctuation in an ATLAS statement.

3.10 diagnostic test: A test applied to a unit under test UUT for the purpose of isolating a fault to a lower level of assembly.

3.11 field: A defined subdivision of an ATLAS statement.

¹ARINC publications are available from ARINC Research Corporation, Document Section, 2551 Riva Rd., Annapolis, MD 21401, USA.

²IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

³ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse. ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

3.12 flag: The first character of an ATLAS statement used to mark that statement as having a special purpose or capability.

3.13 functional test: A sequence of tests applied to a unit under test (UUT) to establish whether it is functioning correctly.

3.14 global: Relating to the whole of an ATLAS test requirement.

3.15 local: Relating to a bounded part of an ATLAS test requirement.

3.16 macro: A defined procedure or sequence of operations or characters that is inserted in the procedure each time its name is invoked.

3.17 metalanguage: A form of notation used to rigorously define the syntax, and sometimes the semantics, of another language.

3.18 semantics: The connotative meaning of words within an ATLAS statement.

3.19 SI units: Units belonging to the International System of Units. See also IEEE/ASTM Std SI 10-1997.

3.20 syntax: The grammatical rules pertaining to the structure of an ATLAS statement.

3.21 test: An action or group of actions performed which evaluate a parameter of a unit under test (UUT).

3.22 test method: A testing approach, philosophy, or strategy.

3.23 test procedure: A description of the tests, test methods, and test sequences to be performed on a unit under test (UUT) to verify conformance with its test specification with or without fault diagnosis and without reference to specific test equipment.

3.24 test program: An implementation of the tests, test methods, and test sequences to be performed on a unit under test (UUT) to verify conformance with its test specification with or without fault diagnosis and designed for execution on a specific test system.

3.25 test requirement: A definition of the tests and test conditions required to be performed on a unit under test (UUT) to verify conformance with its performance specification.

3.26 test sequence: A specified order of related tests.

3.27 test specification: A definition of the tests to be performed on a unit under test (UUT) to verify conformance with its performance specification and without reference to any specific test equipment or test method.

3.28 unit under test (UUT): The entity to be tested. It may range from a simple component to a complete system.

3.29 variable: A quantity, the value of which is assigned at program run-time.

3.30 virtual resource: A notional test resource, the performance characteristics of which conform to a summary of related test requirements.

4. The need for a test language standard

4.1 The importance of unambiguous communication

Communication is a process of transferring data from one source to another. In human terms it involves the transfer of ideas, commands, and requests for data (questions) from one human to another. It takes many forms by virtue of a wide variety of languages and an even wider variety of dialects within any one language. It is an observed and easily proven fact that human communications, regardless of the language used, are imperfect.

Human communication is best accomplished in a face-to-face environment where each word and phrase can fluctuate in its meaning and implication based on an almost unlimited variety of other factors. Environment, mannerisms, and intonation all play a key part in human communications. The one phrase that best typifies human communications is that it is a “do what I mean” type of communication rather than a “do what I say” version. In the world of industry almost all problems that occur in the day-to-day operation of a company can ultimately be traced back to a lack of communication. Nowhere is this relationship more clearly seen than in the world of automatic test equipment. This world revolves around a series of data transfers starting with a design specification for the unit under test (UUT), progressing through a variety of test specifications for that unit and ending in a test program to verify the unit’s operability and assist it in its return to operation if it has failed. This entire process can take place over a significant period of time and can embrace several different organizations. Given this environment, it should come as no surprise that automatic testing requires an explicit communications medium. It should be even less surprising that much emphasis has been placed on improving communications within this environment. These attempts have been centered on test-language standardization and have led to a focus on the use of the standard language designated ATLAS. This clause is intended to show the reasoning behind the utilization of this language and the steps that were taken to bring it to its present point of development.

4.2 Types of languages

4.2.1 Person-to-person communication language

In 4.1 the imperfections surrounding human communications have been discussed. Of prime importance is the realization that the “do what I mean” approach to communication is only viable in a direct contact situation between two individuals with sufficient common background to permit an interpretation of data elements outside the communications language as used.

Languages used for human communications are known as “natural” languages. This term means that the language functions within a set of rules and is defined by its usage rather than its pure grammatical structure. Examples of this phenomenon are seen in all elements of day-to-day communication. The use of slang, idioms, and fashionable expressions is a part of our everyday lives, and interpretation of these elements, which bear no resemblance to the pure language grammar, is also part of normal human communications. This loose structuring of the communications language becomes the chief reason for ambiguity when the communications vehicle is the written rather than the spoken word. The external elements of intonation, fashion, expression, and environment are now removed, and the recipient of the communication is left to make his or her own interpretation of the precise meaning of the communiqué. In an attempt to compensate for this ambiguity, sections of society with a need for precise communications have evolved their own particular languages that serve their needs, but which may be almost useless when viewed by members of the society outside of the particular user communities.

It is well established that automatic testing also has this need. Communications are not restricted to the human level, but the automatic test equipment (ATE) engineer and manager must also operate with the person-to-machine interface. This environment forms the backdrop against which ATLAS has been developed.

4.2.2 Person-to-machine communication language

Despite their growing complexity, machines represent the very epitome of simple-mindedness. The “do what I mean” approach has no place in the machine world; a machine will universally obey the “do what I say” ground rule, even to the point of destroying itself or its operator. All elements of communication with a machine, whether they originate from a human source or from another machine, are singly interpretable, and consequently, totally without ambiguity. This factor, of course, greatly simplifies machine-to-machine communication, but serves as a complicating factor in person-to-machine communication, since the person is not used to functioning in this environment. In an attempt to smooth the person’s transition into this world of literal translation, languages have been constructed that closely resemble human communications languages, but in fact represent meaningful statements to the machine. The prime difference now is that while the statements are meaningful, they convey only a single meaning to the machine.

All person-to-machine communication languages fall into one of two categories. These categories are frequently given different names, but here they will be called high-order language and low-order language.

A high-order language is sometimes referred to as a problem-oriented language or, in the automatic testing world, a test-oriented language. It is characterized by the fact that it restricts itself to simple statements of a problem and avoids specifying methods of solution to that problem. For example, the statement shown in Table 1 as a high-order solution states that solving the equation $a = b^3$ is desired. No mention is made of any method of solution, and consequently, the method is left entirely up to the machine. Since the statement is also nonambiguous and since a machine is a repetitive environment, the method of solution will always be the same, regardless of its adequacy, for a particular situation.

Table 1—High- and low-order languages

Problem: Calculate $a=b^3$	
High-order solution:	Low-order solution:
A=B**3	LL*B (load accumulator with B)
	MM*B (multiply accumulator by B)
	MM*B (multiply accumulator by B)
	SL*A (store accumulator in A)
A high-order language addresses machine inputs and outputs.	A low-order language addresses the machine, machine element, or both.

A low-order language takes the reverse approach and ignores the problem itself, choosing instead to focus on the elements that make up a solution. By doing this, it allows the user to construct a wide variety of solutions to a particular problem, but at the same time it offers no guarantee that any of those solutions will be correct. It also restricts the solution of the problem to a particular machine. Regardless of the correctness of the solution, the machine will faithfully implement the instructions given and generate an answer. In short, the flexibility to implement more than one solution to a problem (high-order solution) has been traded for the guarantee that the problem itself will be solved by the machine (low-order solution). Table 1 also shows a low-order representation of the $a = b^3$ problem. In the low-order solution, no mention is made of the problem and the language is entirely oriented toward the solution on a particular machine.

In the world of automatic testing, standardization has focused on high-order languages. When examined in detail, it becomes obvious that this focus is indeed the correct one. The desire is to communicate a sequence of events that must occur, and it is left to the implementer, whether it be a person or a machine, to ensure that those events are faithfully reproduced. Any attempt to communicate specific solutions will fail for a multitude of reasons. The solution itself may be incorrect or incomplete, or both. The solution may require access to equipment or expertise that does not exist or is not available to the recipient. In general, low-order solutions are considerably more complex and correspondingly more voluminous than their high-order alternatives. The loss of flexibility suffered by moving to a high-order language is more than justified by the improvement in communications and the freedom from depending on specific tools to do the job.

4.3 Need for unambiguous language

The major stumbling block in test information communication is the need to express testing functions concisely and unambiguously. This need cannot be met by the use of a “natural” language with its many problems and ambiguities. Although the specialist terms required for testing exist in the natural language and may be well defined, the meanings of everyday words are too wide. Too many imperfect synonyms exist to guarantee that total information transfer will be effected.

The obvious solution was to suggest the use of a natural language that already had wide usage and to subset and redefine this language for the specific purpose of test information communications. This approach was the driving force behind the birth of ATLAS.

At the time ATLAS was conceived, many artificial languages existed for machine communications, and many special languages existed for particular automatic test equipment. All these languages lacked the general-purpose approach applicable to the testing domain. Only by addressing the general-purpose issue could the required international acceptance of the language be achieved. A stylized language was required based on a subset of a natural language. This subset was oriented toward the testing problem.

By virtue of its international acceptance, the natural language chosen was English. The initial area of focus was that of avionics testing due to the interest of industry in solving testing problems in this area. The obvious area of subset focus was that of avionics testing, since it was in that area that the bulk of the testing problems occurred. Therefore, ATLAS was created as an avionics test language based on a subset of the English language. Subsequently, the language has been adopted for use on a much wider range of electronic systems and equipment. ATLAS has continued to be developed into a robust test language for *all* electronic systems as the name ATLAS implies.

ATLAS was developed as both a person-to-person communication language and a person-to-machine communication language. As a person-to-person communication language it provides precise constructs, grammar, and terminology to communicate test requirement information unambiguously and concisely. It permits the exact description of a test requirement in terms and expressions sufficient to enable understanding by humans. ATLAS is also a high-order language for person-to-machine communications. It utilizes constructs that are carefully selected to be independent of particular solutions or particular test equipment. This independence allows faithful implementation of an ATLAS test requirement in a variety of ways and on a variety of test equipments, both manual and automatic. The realization of these goals is the key to the understanding of the ATLAS language, and the establishment of an ATLAS standard is the key to the continued use of the ATLAS language.

4.4 The portability of ATLAS

When ATLAS is used as a person-to-machine communication language, it does not necessarily provide all the information required to execute a test program on a specific test system. An ATLAS test *specification* may contain all the information relating to the UUT; but in order to execute a test *program* on an ATE some additional information will be required, including information about the ATE and its resources, limitations,

and interface to the UUT. This process usually makes use of an ATLAS compiler or translator that uses system-specific information to convert the ATLAS into an executable program.

The ATLAS language describes the power requirements and signals that pass over the UUT connection interface with no adjustments to accommodate test equipment functional characteristics. Therefore, the UUT connection interface is the ATLAS interface.

Note from the definitions in Clause 3 that ATLAS test specifications and ATLAS test procedures are independent of test equipment. ATLAS test programs are related to a particular test system. In a perfect world with a perfect compiler and a perfect test system, the test procedure and test program would be identical. In the real world, the test program will be developed from the test procedure after consideration of the limitations of the test equipment, the implementation of the compiler, and the measurement inaccuracies of the instruments. Most ATLAS-based test systems use an extended subset of standard ATLAS as their programming language. Changing a test specification to satisfy the limitations of a particular test system is not acceptable. The test system should be upgraded to satisfy the test requirements of the UUT.

An ATLAS test *specification* is, therefore, portable and may be implemented on any suitable test system. An ATLAS test *program* is unlikely to be portable, except to an identical test system. The information and examples in this guide, therefore, refer to ATLAS requirements that use standard ATLAS.

4.5 ATLAS development

4.5.1 General

At the request of the airlines, Aeronautical Radio Inc., (ARINC) began in 1965 to formulate a standard test language based on English. Sponsored by ARINC this activity was regulated by the Airlines Electronic Engineering Committee (AEEC) whose principal function is to formulate standards for electrical equipment and systems for the airlines.

In 1976 ARINC transferred the control of the development of the ATLAS language to the IEEE because ATLAS could be used for testing beyond that applied to the airline industry. The IEEE became the body to assume the further enhancement and standardization of the ATLAS language.

The historical development of the ATLAS language is summarized in Tables 2 and 3.

Table 2 summarizes the development of IEEE Std 416. Each published version of IEEE Std 416 is identified by a hyphenated number, which usually includes the year of approval. Revision levels are included in the table and are numbered sequentially. Table 3 summarizes the development of IEEE Std 716.

4.5.2 Constraints

In the selection of subset elements of English for inclusion in ATLAS, constraints had to be considered. The first required that the general syntax be restrictive, and to this end the imperative or command tense was selected. This imperative format is

DO, SOMETHING, WITH SOMETHING ELSE, SOMEWHERE \$

While the format itself is at least English-like, the punctuation differs considerably. The continual insertion of commas between statement segments or fields reflects a desire to make the statement more machine readable. The selection of the American currency symbol (\$) rather than the period (full stop) as a terminator is equally practical; the single-minded machine would have no way of readily differentiating between the ubiquitous decimal point used in engineering and the statement terminator, or period, used in normal everyday English. The dollar symbol was selected as a nonengineering element that could be a reasonable terminator for a statement.

Table 2—The historic development of ATLAS

IEEE Std 416 Issue/revision level	Month and year	Development
	1965-1966	A study of the need for a standard test language was initiated as the airlines were considering using automatic test equipment for maintenance testing.
	January 1967	The first AEEC subcommittee meeting was held to formulate a test language.
	August 1967- September 1968	Seven preliminary drafts of ATLAS Specification 416 were generated.
ARINC 416-1	October 1968	The final draft was approved by AEEC for issue as ARINC Specification 416-1.
2-5	1969-1972	Miscellaneous changes and refinements were introduced in the language that previously only covered analog testing requirements.
6	October 1972	A digital testing capability was introduced.
7	July 1973	Some minor changes were introduced.
8	July 1973	A digital macro construct was introduced to augment the handling of digital testing.
9	May 1974	Miscellaneous refinements were incorporated.
10	May 1975	The most significant changes were the introduction of decision table and tasking capabilities. Decision table defines and initiates branching a test requirement that is dependent upon a complex relation between a number of variables. Tasking allows for the use of separate test sequences to be performed simultaneously.
11	January 1976	A structured programming capability was introduced that allowed a test requirement to be divided into self-contained blocks of test routines.
12	April 1976	Miscellaneous refinements (e.g., relating to complex numbers and waveform handling) were incorporated.
13	August 1976	Miscellaneous changes were introduced.
13A	August 1976	The ARINC Specification 416-13 was reformatted (with no changes) to improve the interpretation and the management of the document.
IEEE Std 416-1976	November 1976	The reformatted version, ARINC Specification 416-13A, was adopted by the IEEE as their launching version of the ATLAS language standard.
14	November 1978	A capability for defining test resources as a requirement for testing was incorporated.
15	October 1979	Miscellaneous refinements were introduced.
16	July 1980	Miscellaneous refinements were incorporated.

Table 2—The historic development of ATLAS (continued)

IEEE Std 416 Issue/revision level	Month and year	Development
17	October 1980	New syntax and semantics were introduced to provide for — Definition of signals — Inclusion of ATLAS and non-ATLAS program modules — Definition of protocol procedures and data exchanges allowing databus testing to be handled New vocabulary was introduced to provide for — Defining electromagnetic fields — Radar signals — Ambient conditions — Fluid signals — Turbine engine — Vibration testing
18	April 1981	A number of ATLAS statements were introduced to identify events during testing to improve the handling of time-related testing. Statements were introduced to augment the handling of tasking to suspend, continue, request, and release tasks. Vocabulary related to input and output devices was added. A statement was added to separate the preamble and procedural structures in ATLAS test requirements. New constructs for handling inputs and outputs of test information were introduced.
19	October 1981	The IF THEN ELSE structure was enhanced to include ELSE IF. User-defined functions were introduced. New digital constructs were introduced to provide more flexibility for handling modern digital testing techniques. An escape mechanism was introduced to facilitate error handling.
20	May 1982	Miscellaneous refinements were incorporated.
21	November 1982	State diagrams to correlate single-action and multiple-action verbs were introduced. Miscellaneous refinements were incorporated.
22	May 1983	Vocabulary was introduced for Scorsby-like motion, visible light, infrared radiation. Requirements to the bus testing constructs and new vocabulary for video signal testing were introduced.
23	November 1983	Miscellaneous refinements were incorporated.
24	May 1984	Quantities and dimensions conforming to the ISO 1000:1992 were introduced for use on nonaerospace and nondefense projects (although they are equally useful in the aerospace and defense industries). New vocabulary related to target signals was introduced.
25	November 1984	Refinements to the old and new digital testing constructs were incorporated.
IEEE Std 416-1984	December 1984	Published version was aligned to development revision level 21.

Table 2—The historic development of ATLAS (continued)

IEEE Std 416 Issue/revision level	Month and year	Development
26	May 1985	Structured data types were introduced to provide more rigorous handling of basic data types, constants, variables, and parameters. Refinements to the vocabulary relating to waveforms were incorporated.
27	November 1985	Refinements to sensor statements were incorporated to allow for sampled measurements. An EXTEND mechanism was introduced to facilitate the inclusion of new vocabulary into ATLAS test requirements. Some nonpreferred ATLAS statements were deleted. Some infrequently used and superseded statements were made nonpreferred.
28	May 1986	Refinements for bit synchronization were added to the new digital testing constructs. The first refinements to the definitions of the ATLAS vocabulary review were incorporated.
31	November 1987	Further refinements to the definitions of the ATLAS vocabulary resulting from the vocabulary review were incorporated.
32-33	May 1988- November 1988	Further refinements to the syntax and semantics of ATLAS arising from the creation of the subsets IEEE Std 716-1989 and ARINC Specification 626-1. Further refinements to the definitions of the ATLAS vocabulary arising from the vocabulary review were incorporated.
34-36	May 1989- November 1990	Miscellaneous refinements were completed. Decision was made to freeze IEEE Std 416 at revision level 36 with no further development to take place. The revision level 36 development document was archived for future reference.
(36)	June 1993	IEEE Std 416-1984 was withdrawn.

The early versions of ATLAS were directed toward the testing of analog electronic equipment, although some content was included for logic signals. There were 34 verbs, 30 nouns, and approximately 200 other words in the vocabulary of ATLAS to ARINC Specification 416-1. The number of verbs, nouns, and other words has increased considerably; additionally, many of these may be prefixed, suffixed, and sometimes doubly suffixed to produce distinct compound words.

Additional verbs cover macro-functions or multi-actions for digital test purpose, multiple decisions, and “structured” programming techniques. Additional nouns provide further definition of digital signals and air navigation signals, as well as nonelectronic terms, which are reflected in the change of meaning of the second A in the acronym ATLAS from “Avionic” to “All.” In the full mathematical sense, logic as well as arithmetic operations can be specified.

Another constraint surrounding ATLAS development was that, in line with the need for a clear definition by a computer as opposed to a complex interpretation by a human being, strict yet simple language rules were required. The technique adopted to lay down these rules is one that is well accepted in the computer world, and yet almost unknown outside of this sphere. It involves the use of a formal definition language or, as it is frequently termed, a metalanguage.

Table 3—The development of IEEE Std 716 (C/ATLAS)

IEEE Std 716	Year and month	Development
1982	March 1982	A common ATLAS subset (C/ATLAS) for use in the defense industries was created from revision level 17 of IEEE Std 416. This document contained ATLAS constructs for analog testing and digital testing using the old digital constructs.
1985	September 1985	Refinements to the syntax and semantics were incorporated. Document was equivalent to IEEE Std 416 revision level 18.
1989	May 1989	The subset was revised to include — SI units — Structured data types — Identification of events — New digital constructs — Databus testing constructs — Requirement for test resources — Extensibility features for semantics Document was equivalent to IEEE Std 416 revision level 30.
1995	October 1995	The standard was revised to include — Escape mechanism — Complex signals constructs — Dynamic digital constructs — Upgraded data-bus testing facilities

The particular metalanguage used in the definition of ATLAS is known as the DeRemer and Frank form. The ability to understand this form and to use it is a totally unnecessary skill for almost all ATLAS users, but it is helpful for the computer specialist faced with the problem of designing an ATLAS compiler or translator for a machine. The first formal definition, or metalinguistic definition, of the ATLAS language occurred simultaneously with the 13th language issue, known as ARINC Specification 416-13A. Since that time this definition has been refined and expanded and is now given in the formal syntax definition (FSD) of IEEE Std 716-1995 and ARINC Specification 626-3.

4.6 Continued regulated growth of ATLAS

Rapidly evolving technologies can cause obsolescence in a language before the language has even established a mature foothold. Nowhere is this more true than in the rapidly expanding and ever-evolving field of electronics. The continued invention of new devices and new techniques requires a well reasoned but simple approach to developing language elements that will allow the transmission of new technology information. To meet this challenge, ATLAS is controlled and developed through the auspices of an international committee of test language experts. These experts serve this committee on a voluntary basis and work together to produce new language elements as needed. Users of the language, regardless of nationality, affiliation, or interest, may generate ATLAS problem sheets for consideration by this committee. These same users are encouraged to attend committee meetings to assist with the problem resolution and ATLAS development.

Meetings of the ATLAS Committee at which these problems and their proposed solutions are considered are organized at least once a year by IEEE. Similar meetings are arranged by ARINC.

4.7 ATLAS as an aid to the management of testing

ATLAS is structured to enforce rules not only related to syntax and semantics, but also to overall test requirement presentation. ATLAS will control the format of the test information, the sequence of the test information, the implementation of testing decisions, and the identification of test elements.

By controlling these elements and by eliminating the redundancy in information communication, ATLAS can improve efficiency and add clarity in communications, whether person-to-person or person-to-machine. The benefits of clarity are well-known and can lead to improvement in performance and reduction in testing costs.

4.8 Benefits of a standard

Quantifying the benefits of adopting a standard, in particular ATLAS, is difficult because few suppliers have either the time or the resources to conduct an experiment to measure the total effect. More often than not the early benefits accrue directly for the purchasers and users. The supplier initially sees only disadvantages. Formal training, on-the-job learning, and the acquisition of more detailed information increase the cost and time scale of the task. In the longer term, especially when the testing has been planned from the outset, the use of a standard test language in a formalized documentation system will show savings over a conventional free-range text, as well as providing a more complete test document. The introduction of ATLAS does not just impinge on the authors of test documentation, but also on the designers and in-house test departments. In many companies tradition, prejudice, and demarcation have hindered the communication process, wherein each department has tried to retain its autonomy. Because ATLAS is impersonal, it has helped to remove some barriers that had existed previously.

An advantage of a standard test language, especially an international one, is that the test information remains a function of the UUT only and is available to any test system implementer. It is not subject to the technical and proprietorial limitations and restrictions of a language owned by a particular ATE vendor.

The ultimate benefit of a standard can only be seen when it becomes universally adopted. As more and more projects use ATLAS on an international scale, the language itself will be refined to meet a larger proportion of total applications. With this refinement and with this extended usage, the nonrecurring costs associated with training and learning curves will be minimized, and the basic cost reductions associated with simplified communications and improved data transfer will become clear. In this sense, ATLAS is no different from any other standard.

5. The current ATLAS standards

5.1 Definition of ATLAS

ATLAS, a standard abbreviated English language, is used in the preparation and documentation of test requirements that can be implemented either manually or with automatic or semiautomatic test equipment. The ATLAS language is defined in IEEE Std 716-1995 and ARINC Specification 626-3. These documents include a general or functional definition as well as a formal definition written in a metalanguage.

5.2 IEEE Std 716-1995

This standard is no longer a subset of IEEE Std 416 (now withdrawn). It is a full standard in its own right and contains those elements of the language that were deemed necessary by the 716 C/ATLAS Subcommittee of IEEE SCC20. This standard is published on a regular basis approximately every five years.

5.3 ARINC Specification 626-3

This specification is no longer a subset of IEEE Std 416 (now withdrawn). It contains those elements of ATLAS deemed necessary by the user community of airlines and manufacturers for the testing of avionics in commercial airline industry. This specification is updated and published by ARINC as required by the demands of its user community.

5.4 IEEE Std 771-1998

Wherever possible, the current version of this guide is aligned to the ATLAS standards described in 5.2 and 5.3. The general principles and programming practices described in this guide are relevant to most versions of ATLAS. If a user has any doubts or queries about the specific syntax or semantics of examples in this guide or of their own programs, then reference must be made to the appropriate standard language definition.

6. Structure and characteristics of ATLAS

This clause presents the user with information regarding the general definition, general characteristics of ATLAS, the levels of ATLAS, and the basic structure of an ATLAS test requirement. It also discusses FSD and non-ATLAS usage.

6.1 General definition

The general definition is provided by means of functional descriptions and rules applicable to ATLAS constructs. Cross references to the formal syntax definition are provided. The text is augmented in appropriate places by diagrammatic representations of valid syntax. These diagrams are called syntax diagrams and provide guidance in constructing valid ATLAS statements.

6.2 General characteristics of ATLAS

The following are the general characteristics of ATLAS:

- a) *UUT orientation.* The language is dedicated to defining the test requirements of the UUT without reference to or dependence upon the test equipment that may be used. The latter may be automatic, manual, or hybrid in design.
- b) *Unambiguous communication.* The selective use of English language terms, which are compatible with the description of test requirements, and a formal structure for their use constitute an environment that will ensure an unambiguous description of the requirements of a test requirement for the UUT designers, developers, users, and maintenance personnel.
- c) *Test equipment independence.* The specification of test requirements in terms of the UUT by ATLAS facilitates the transportability of test specifications from implementation on one set of test equipment to another, provided that all the test requirements can be satisfied by each set of the test equipment.

6.3 Levels of ATLAS

The following are the four levels of ATLAS:

- a) *Standard ATLAS.* The complete language, including vocabulary, syntax, and rules, is defined in IEEE Std 716-1995 or ARINC Specification 626-3.

- b) *Subset ATLAS*. ATLAS has evolved as a broad based test language from the collective requirements of a wide spectrum of users. The language was used primarily in the areas of avionic and electronic testing (both analog and digital), but has now been extended to include several areas of nonavionic and nonelectronic testing. Individual ATLAS users seldom require or implement the entire language (i.e., standard ATLAS) because of the broad spectrum of coverage, language redundancy, and economic constraints of implementation. Subset ATLAS is, therefore, a language of which every constituent construct is included within standard ATLAS (that is, no extensions) except that, for commercial or technical reasons, it does not include all the vocabulary or statements and syntactic options of standard ATLAS.
- c) *Subset ATLAS with extensions*. A version of ATLAS that is based upon an ATLAS subset, as defined in item b), and that includes ATLAS extensions appended to this subset. Specific rules for the extension of ATLAS semantics are defined in IEEE Std 716-1995 and ARINC Specification 626-3. Legally adopted extensions of this type may be considered as conforming to standard ATLAS.
- d) *Adapted ATLAS*. ATLAS is a precise language used to communicate test requirements among various users. ATLAS lends itself to adaptation for particular test environments when it would be advantageous to modify the ATLAS language to aid implementation. Any such modification is known as adapted ATLAS. Adapted ATLAS is the term used to refer to the ATLAS requirement after it has been modified to suit a local application.

Adapted ATLAS identifies a family of languages that conform in some manner to a published ATLAS standard, but that have modified vocabularies and possible syntax distortions. These languages are not conducive to the creation of portable ATLAS test specifications, but may be necessary to fulfill specific testing needs on a particular project.

An adapted ATLAS language is not controlled by either the IEEE or ARINC and its use is discouraged.

6.4 ATLAS test requirement structures

6.4.1 General

An ATLAS test requirement is structured into distinct groups that are defined in 6.4. Figures 1 through 3 illustrate the structure of an ATLAS test requirement, including an ATLAS program structure and optionally any number of ATLAS module structures and non-ATLAS module structures.

The ATLAS standards always use the term “program” and not “specification,” “procedure,” or “requirement” in the language definitions. When an ATLAS standard refers to “program,” it can be applied to test specifications, procedures, or requirements equally well.

ATLAS and non-ATLAS modules exist as entities separate from the ATLAS program structure and are linked to that structure by INCLUDE statements in the main preamble structure. ATLAS and non-ATLAS modules provide a facility for preparing a large ATLAS test program by using a number of smaller, manageable portions with a controlled interface. See Figure 1 for an ATLAS test program family tree.

A syntax diagram (see Figure 2) is a diagrammatic representation of a level in the hierarchical structure of ATLAS. The title of the diagram is boxed by solid lines at the left side of the diagram. Constituent elements and the manner in which they may be linked together are described in a notation similar to flowcharting and are written on the right side of the diagram. The constituent elements may be either ATLAS reserved words (in uppercase) or other reserved characters such as \$ or /, or elements that require definition by a syntax diagram at the next lower level of the hierarchy. These latter elements are shown as dashed boxes containing the name of the element written in lowercase letters. A number is also included in the box to identify the paragraph in the standard where that element is defined.

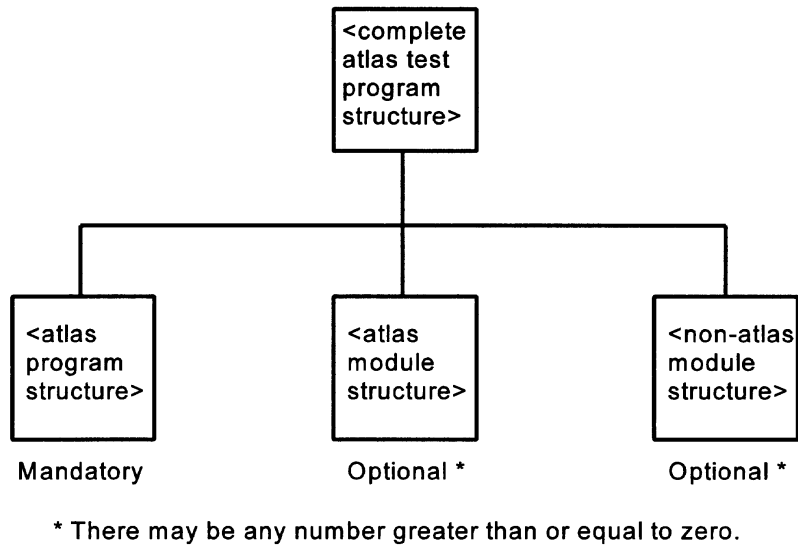


Figure 1—ATLAS test program family tree

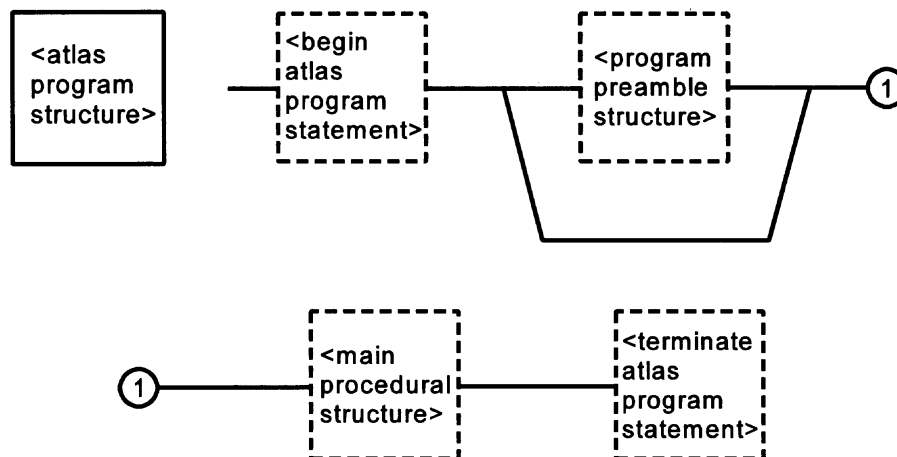


Figure 2—ATLAS program structure syntax diagram

Flow lines that link or by-pass the constituent elements must be read from left to right, except where an arrowhead on a line (see Figure 4) indicates that additional passes may be made through the part of the diagram looped by this line. Where options exist, these are shown by parallel paths.

The main preamble structure precedes the main procedural structure (see Figure 3) and contains all the statements that predefine functions or declare types of variables for the ATLAS test requirement. Preamble statements do not cause any tests to be executed, but the information described in the preamble structure may be repeatedly referenced in the procedural structure.

An ATLAS program structure may contain a single main preamble, and each DEFINE PROCEDURE statement may contain a single local preamble. In both cases the preamble serves to identify the name and attributes that are subsequently referenced within the appropriate part of the procedural structure.

000000	BEGIN, ATLAS PROGRAM 'PROGRAM-STRUCTURE'	\$
000100	INCLUDE, ATLAS MODULE 'ADDITIONAL-INFORMATION'	\$
000300	REQUIRE, 'RESOURCE-1'.....	\$
000400	ESTABLISH, BUS PROTOCOL 'SERIAL-BUS'.....	\$
000500	DECLARE, 'TEST-VARIABLE'.....	\$
000600	IDENTIFY, EVENT 'TRIGGER-POINT' AS.....	\$
000700	DEFINE, 'DC-1', SIGNAL, ...	\$
000800	DEFINE, 'TEST-SEQUENCE', PROCEDURE	\$
	APPLY, ...	\$
	APPLY, ...	\$
	VERIFY, ...	\$
	REMOVE, ...	\$
	REMOVE, ...	\$
000860	END, 'TEST-SEQUENCE'	\$
C	END OF PREAMBLE	\$
C	COMMENCE, MAIN PROCEDURE	\$
E100000	APPLY, ...	\$
10	APPLY, ...	\$
	MEASURE, ...	\$
	VERIFY, ...	\$
	REMOVE, ...	\$
120000	PERFORM, 'TEST-SEQUENCE'	\$
150000	PERFORM, 'ANOTHER-SEQUENCE'	\$
500400	IF, 'GOOD', THEN	\$
10	OUTPUT, C'UUT PASSED'	\$
20	ELSE	\$
30	OUTPUT, C'UUT FAILED'	\$
40	END IF	\$
C \$		
999997	REMOVE, ALL	\$
999998	FINISH	\$
999999	TERMINATE, ATLAS PROGRAM 'PROGRAM-STRUCTURE'	\$

Figure 3—ATLAS program structure

6.4.2 Program preamble structure

The program preamble structure, which appears immediately following the BEGIN ATLAS PROGRAM statement, establishes labels that are global to the entire ATLAS program structure unless an identical label is established, at a lower level structure, within a local preamble structure. The program preamble structure may contain INCLUDE, EXTEND, REQUIRE, ESTABLISH PROTOCOL, DEFINE, DECLARE, and IDENTIFY statements.

The syntax diagram of a program preamble structure is shown in Figure 4. The syntax diagram shows that all elements of the preamble are optional. All variables must be described in DECLARE statements before they can be used in procedural statements. Similarly, in ARINC Specification 626-3, all test resources must be described in REQUIRE statements. As a result, all practical ATLAS programs will include a program preamble structure.

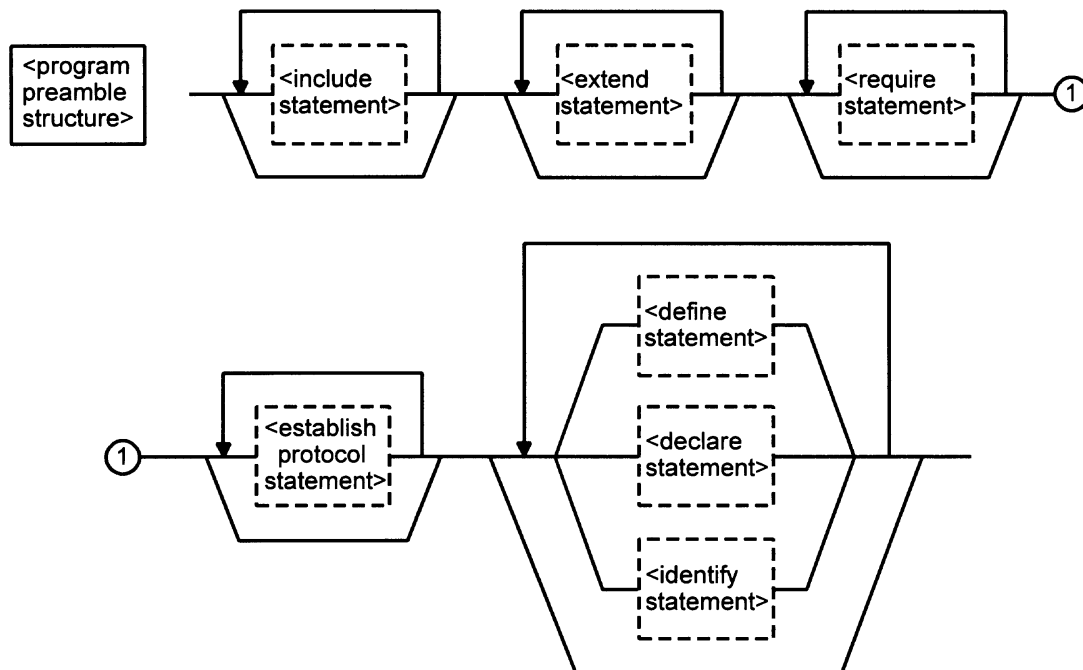


Figure 4—Program preamble structure syntax diagram

If the module facility is used, INCLUDE statements must be written to link the module documentation to the main program. The preamble linking does not cause execution of these modules; the execution of a module is dependent on an appropriate PERFORM statement within the main program procedural structure.

The EXTEND statement provides a facility to add nouns, noun modifiers, dimensions, and pin descriptors to the ATLAS language. Data-bus facilities may also be extended using this statement. The EXTEND statement may only be used to add semantics not available in standard ATLAS. It may not be used to define an alternative to existing capabilities in ATLAS.

The REQUIRE statement is used to describe the virtual test resources required to implement the test requirement. The *virtual* resources described in REQUIRE statements have to be satisfied by *real* resources in a test system when the test requirement is implemented.

The ESTABLISH PROTOCOL statement is used to describe a protocol for subsequent use in data-bus exchanges.

The DEFINE statement establishes and labels a section of the ATLAS test requirement that will be executed when the label is cited in a subsequent procedural statement. This section may be a series of statements, a single statement, or part of a statement.

The DECLARE statement establishes, labels, and optionally initializes data storage (i.e., data types, variables, constants) for subsequent use in an ATLAS program.

The IDENTIFY statement describes events, event intervals, event indicators, and timers that can be used to relate measurements and signals to precisely defined occurrences in the test requirement. Events are related to signals to or from the UUT or are related to time.

By using the facilities in the preamble structure, sources, loads, and sensors that are applied or used repeatedly can be defined once and assigned a label. They can then be referenced by these names whenever required, in the procedural structure.

For example, a dc power source may be defined once in the preamble:

```
001520 DEFINE, 'DC-1', SIGNAL,  
        DC  
        DC SIGNAL,  
        VOLTAGE 5V,  
        CNX HI J1-1 LO J1-2 $
```

This power source is then applied in the procedural structure as follows:

```
100421 APPLY, 'DC-1' $
```

which is equivalent to

```
100421 APPLY, DC SIGNAL,  
        VOLTAGE 5V,  
        CNX HI J1-1 LO J1-2 $
```

Defining signals in the preamble structure can be used to save repetition of detail in the procedural structure.

Complete tests, which are repeated several times, may be defined in a DEFINE PROCEDURE statement. In this case the test is written as several complete separate statements following the DEFINE PROCEDURE statement and is executed by the appropriate PERFORM statement in the procedural structure. Repeated messages can be handled in the same way.

Although no specific ATLAS symbol denotes the end of a preamble structure, indicating the end of a preamble structure with a comment is recommended (see Figure 3). In ARINC Specification 626-3 the procedural structure is introduced by a COMMENCE MAIN PROCEDURE statement, which indicates the end of the main preamble structure.

6.4.3 Main procedural structure

The main procedural structure consists of a series of statements describing a portion of the required test. Each statement implies an instruction to proceed to the following statement immediately after completion of the present one, unless directed otherwise in a branching statement.

6.4.4 Basic statement structure

The elements of a general ATLAS statement structure are shown in Figure 5.

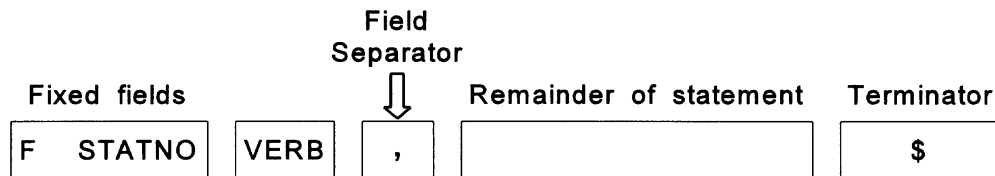


Figure 5—Basic statement structure

The detailed format for each verb type is specified in the appropriate clause of this guide. ATLAS statements have no length limits and need not be completed in one line.

- a) General ATLAS statement structure. The writer can begin a new line after any complete language element in the statement with subsequent lines indented, if desired. This flexibility allows the writer to highlight some portion of the statement (such as the label) by its location on the page. The first three fields (flag, statement number, and verb) must be written on the first line as required by the format. The flag and statement number must be written as the first 7 characters of a statement.
- b) Flag field. The flag field designates different types of ATLAS statement.
- c) Statement number field. The statement number (statno) field is used to uniquely designate an ATLAS statement.
- d) Verb field. The verb field identifies the test or data processing action for each statement. The ATLAS verbs are reserved words and must appear as the first word of a complete statement.
- e) Field separator. Since the fields following the verb are variable, a separator is required to identify the start of each new major field. A comma (,) followed by zero or more spaces is designated for this purpose. Within some major fields, blank spaces are used as subfield separators.
- f) Remainder of statement. This part of each particular type of ATLAS statement varies considerably. The detailed format following each verb type is specified in the appropriate clause of the ATLAS standards.
- g) Statement terminator. The last character of any ATLAS statement must be the US currency symbol (\$).

6.5 FSD

The FSD of IEEE Std 716-1995 was devised to give a rigorous definition of the syntax and vocabulary for each statement in an ATLAS test requirement. It employs an extension of the Backus-Naur notation to write syntax definitions in the form of 'equations' known as language production rules.

The Backus-Naur notation was invented to define exhaustively the permitted language structures of computer programming languages. It is known as a metalanguage because it is effectively a very limited language used to define more complex languages rigorously.

The FSD defines what character sequences are permitted. Its treatment is more precise than that of the syntax diagrams and rules, but it is not designed to give any indication as to the meaning of those character sequences (semantics). Making assumptions about the meaning of the metalinguistic names (tokens) from the English-language meanings is dangerous.

IEEE Std 716-1995 and ARINC Specification 626-3 define both the structure of the language and its meaning, technically known as the syntax and the semantics, respectively. Some syntax diagrams are a little imprecise to make the diagrams more readable; those diagrams are supplemented with rules to describe valid paths through the diagrams for particular circumstances. The FSD was added to present the permitted syntax in a formal machine-processable form. It is presented separately to not impair the readily assimilated form of the syntax diagrams and rules.

Most writers of ATLAS test requirements will find it unnecessary to refer to the FSD; but if a large number of requirements are to be written by a team, it is normally advantageous for one or two members of the team to understand the FSD to resolve any problems.

The material in the FSD takes precedence over that in the rest of the standard because the FSD is a more formal treatment of the syntactic structure of ATLAS. The developers of the latest versions of IEEE Std 716-1995 and ARINC Specification 626-3 generated the syntax diagrams from the FSD to greatly reduce the possibility of errors. However, errors are possible in the FSD. Anyone encountering any inconsistency should highlight the problem in the form of a change proposal to the IEEE ATLAS committee or the ARINC ATLAS committee. Clause 12 describes how this proposal may be presented.

6.6 Non-ATLAS usage

If, when writing an ATLAS test requirement, the ATLAS language as defined in IEEE Std 716-1995 or ARINC Specification 626-3 does not adequately cover a particular test discipline or problem, a non-ATLAS solution may be invoked. The non-ATLAS test routine will be provided as a non-ATLAS module. The non-ATLAS module is linked to the ATLAS program by the INCLUDE statement in the main preamble and executed by an appropriate PERFORM statement in the procedural structure.

The non-ATLAS module is likely to be written in another computer language such as 'C' or an assembly language. It could also be the native language of a particular test system. The only link between routines in a non-ATLAS module and the ATLAS program is via the PERFORM statement, so that any passing of parameters shall conform to the ATLAS standard. Also a defined procedure shall be in the ATLAS module or main program that contains the PERFORM statement. This procedure will include the keyword EXTERNAL and will not contain any procedural statements.

Non-ATLAS modules are by their very nature implementation specific and therefore should only be used when unavoidable. Programs using non-ATLAS modules are not necessarily portable.

The non-ATLAS text should be written in a statement format similar to ATLAS, and the writer should submit the problem and a possible solution to the ATLAS committees. In this way ATLAS will continue to be upgraded.

7. The elements of ATLAS

Like any other language, ATLAS has a set of symbols or characters that may be used singly or collectively in a prescribed order to convey meaning. This clause describes the characters and their uses and the words that may be formed using these characters.

7.1 The ATLAS character set

Characters from the seven-bit ASCII (American Standard Code for Information Interchange) code are available for use in ATLAS programs. Table 4 shows the current printing ATLAS character set.

Table 4—ATLAS character set

Characters	Usage
Uppercase letters: A through Z	Keywords, labels, character string data, and comments
Lowercase letters: a through z	Labels, character string data, and comments
Numerals: 0 through 9	Labels, variables, constants, and comments
Symbols: Blank space ! Exclamation point " Double quotes # Number, hash or pound sign \$ Statement terminator (dollar or currency character) % Percent sign & Ampersand ' Apostrophe or single quote (Left parenthesis) Right parenthesis * Asterisk, multiplication, and exponentiation signs + Plus sign , Comma - Minus sign or hyphen . Period or decimal point / Slash or division sign : Colon ; Semicolon < Less than = Equal sign > Greater than ? Query or question mark [Left bracket \ Backslash] Right bracket ^ Circumflex _ Underscore { Left brace # Bar or vertical line } Right brace ~ Tilde	Delimiters and punctuation

The nonprinting characters used in ATLAS programs are shown in Table 5. Not all characters are available in all situations; for example, non printing characters are only available in character strings and message texts and are represented by the mnemonic shown in Table 5 within slash characters. For the detailed rules applicable to the use of the character set, refer to the appropriate ATLAS standard.

Table 5—ATLAS nonprinting characters

Nonprinting characters				
NULL	Null character	VT	Vertical tab	SYN
SOH		FF	Form feed	ETB
STX		CR	Carriage return	CAN
ETX		SO		EM
EOT	End of tape	SI		SUB
ENQ		DLE		ESC Escape
ACK	Acknowledge	DC ₁		FS
BEL	Bell	DC ₂		GS
BS	Back space	DC ₃		RS
HT		DC ₄		US
LF	Line feed	NAK		DEL Delete

The following notes and guidelines should be considered when preparing ATLAS requirements:

- In handwritten ATLAS, O (the alphabetic character) and 0 (the number zero) are separate characters. A slash should be used to distinguish the zero character.
- Blank spaces are used in some ATLAS elements that comprise two or more words (mainly nouns, e.g. AC SIGNAL) and are part of that language element. There *must* be one and *only* one space between those elements.
- One or more spaces are used as a delimiter between ATLAS elements and keywords where confusion could occur.
- Blank spaces adjacent to all special characters that are not part of a language element (such as the comma, +, −, \$) and between alphabetic and numeric subfields are optional. One exception to this rule is in ARINC Specification 626-3 where a comma must be followed by at least one blank space.
- Spaces are used liberally in ATLAS requirements to improve human readability and to give the program a visual structure. Additional spaces are often used to indent statements and parts of statements to give added weight to the most important fields in statements and procedures.
- New-line characters are also used to improve human readability and to enable a maximum line length to be chosen to fit the paper width (or screen size) of the organization producing the ATLAS programs. ARINC Specification 626-3 allows a new line to be used following a comma or statement terminator. IEEE Std 716-1995 does not specify where new lines may or may not be used, but the accepted practice is to follow the method used in ARINC Specification 626-3.

- g) Where no lowercase alphabet is available in an ATLAS editing system and an alpha character in a manufacturer's pin designation is required to be used literally as lowercase, a slash is written immediately preceding each affected letter. For example, SKT1-bb would be written as SKT1-/B/B. This situation is only likely to arise with older installations.
- h) The use of the term "currency symbol" as the statement terminator originated in order to allow flexibility so that other international users of ATLAS could use a symbol other than the dollar (\$) when their keyboards did not include a dollar symbol. In practice all modern keyboards include the dollar symbol and it is used universally as the statement terminator.
- i) Nonprinting characters may be included in OUTPUT messages by enclosing them in backslashes (e.g., \CR).

7.2 ATLAS keywords

A keyword is any single character or sequence of characters recognized as being an element of ATLAS vocabulary by inclusion in the glossary of ATLAS words. Every verb, noun, modifier, dimension, and uppercase word in the ATLAS syntax diagrams is a keyword. See Clause 11 for a complete list of ATLAS keywords and their definitions.

When writing ATLAS requirements it is good practice to avoid using keywords as labels. Even though the compiler or syntax checking tools may be able to determine the difference between a keyword (*without* enclosing single quotes) and a label (*with* enclosing single quotes), it is most likely that other users will become confused when reading the requirements.

7.3 ATLAS number conventions

7.3.1 General

Test requirements may require the inclusion of several different number types for different purposes. ATLAS provides the following number types:

- a) *Decimal numbers*. Further subdivided into integer, fixed-point, and floating-point forms.
- b) *Digital numbers*. Bit strings that may be represented by binary, octal, hexadecimal, and binary-coded-decimal forms.
- c) *Statement numbers*. Integer numbers with a specified format.

7.3.2 Decimal numbers and decimal integers

Decimal numbers are the most familiar form of numbers in general use. Basic decimal numbers are written in ATLAS using the normal mathematical form, digits 0 through 9; they may or may not be preceded by a sign character (+ or −) (see Table 6). The absence of a sign character implies a positive sign (+). A period (full stop) is used to indicate the position of the decimal point. Good style requires that if a decimal point is written, a fractional part should also appear even if it is a zero, such as 34.0.

In other circumstances, such as the index of a list, it is essential to use an integer. As in normal mathematics, an integer is written without a decimal point.

Decimal numbers may be written in fixed-point or floating-point notation. In floating point notation the value is defined by a multiplier (the argument) and a power (the exponent) and take the form "(argument)E(exponent)." Thus 3E10 (or +3.0E10) expresses in ATLAS the value 3×10^{10} .

Table 6—Typical decimal numbers

Number	Signed/Unsigned	Type
16393 +16393	Signed or unsigned (positive)	Decimal integers
−16393	Signed (negative)	
1639.3	Signed or unsigned (positive)	Decimal fixed-point numbers
0.16393 .16393 ^a +0.16393	Signed or unsigned (positive)	
−0.16393	Signed (negative)	
16393.0 16393 ^a +16393.0	Signed or unsigned (positive)	
−16393.0	Signed (negative)	

^aThese are legal but nonpreferred examples of ATLAS numbers

Examples of the many floating-point representations of the (fixed-point) value 1234.5 include

1234.5E0
123.45E1
12345.0E-1
1.2345E3
+1.2345E+3
0.12345E+4

All these forms are equally acceptable in ATLAS. It is common practice to restrict usage so that all floating-point numbers have a common format. This practice is known as normalization. The most common normalization rules are as follows:

- | | |
|----------------|---|
| 1 < arg < 1000 | The argument is written so that the exponent is always a multiple of 3. This approach is effectively the basis of the SI system of units with standard prefixes (giga-, mega-, kilo-, milli-, micro-, and so on). It is good engineering practice to use this system. |
| 1 < arg < 10 | Commonly used in the scientific community. |
| 0.1 < arg < 1 | Commonly used by the computer community because it is normal to work in fractional arithmetic. |

In the items above, “arg” represents the argument.

The rules for number construction are detailed in IEEE Std 716-1995 and ARINC Specification 626-3. Deviations, even if they are common national practices, such as use of a comma for the decimal point or to separate integral and fractional parts into groups of three, are forbidden. Similarly, there shall be no spaces anywhere within a number or even between the sign and the digits of the number.

7.3.3 Digital numbers

Digital numbers are used in association with two-state circuits and thus based on numbers with a radix 2. Such numbers are called binary numbers. The digits of a binary number take the values 0 and 1.

Long strings of 1s and 0s tend to be incomprehensible, inconvenient, and error prone. The use of radices that are powers of 2 reduces the size of the numbers while retaining the ability to distinguish single bit values. Radices of 8 (octal system) and 16 (hexadecimal system) are used. In the hexadecimal system the representations for the decimal values 10 through 15 are the letters A to F.

The legal digits for each number type are as follows:

Binary	0, 1
Octal	0, 1, 2, 3, 4, 5, 6, 7
Decimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

The digits that occur in binary also occur (but with different significance) in decimal, octal, and hexadecimal systems. Therefore, distinguishing the various number types is necessary and is done by enclosing the digital numbers in single quotes and preceding them by a prefix letter to indicate the type of digital number. The letters used are

B for Binary
O for Octal
X for Hexadecimal

For example,

B'11110000110101101'
O'360655'
X'1E1AD'

all of which represent the same binary pattern with the decimal value 123309 if the bits are given integral binary weighting.

Bits in digital numbers can be given a variety of weightings. Some are commonly encountered, in particular binary weighting, with the radix point at any position in the binary string. Another is binary coded decimal (BCD). Here the binary bits are grouped into quadruplets with each group representing the value of a decade. Conversions between binary and decimal numbers are handled by mathematical functions. Standard functions are provided for binary and BCD conversion.

Many UUTs require that the digital words or fields received by them be self-checking by inclusion of a parity check bit. When the stimulus VALUE is a dependent variable, it is not possible at the time of writing the requirement to allocate a specific 1 or 0 as the parity check bit. Depending on whether odd or even parity checking is required, the function PO or PE may be used to generate the parity bit in a digital word stimulus.

7.3.4 Digital number descriptors

Because a binary digit has only two possible information states, 0 and 1, defining a convention that permits a logic bit to specify whether a number is positive or negative is necessary. If it is negative, defining the convention by which the numeric value is represented is also necessary. The number type and the negative convention are defined as parameters of the conversion function by digital number descriptors. The ATLAS digital number descriptors are

- BNR (binary number). Pattern with no sign bit and no negative form.

- BSM (binary sign and magnitude). The most significant bit is a 0 for positive and a 1 for negative. The remaining bits are effectively a BNR and are the same for both positive and negative.
- B1C (binary ones [1] complement). The most significant bit is a sign bit with the same convention as BSM. The remaining bits are all inverted for negative numbers so that a 1 in a positive number is a 0 in a negative number, and vice versa. It has one main disadvantage: there are two different forms for 0 (that is, 00000000 and its complement 11111111).
- B2C (binary twos [2] complement). The negative form is formed by adding a 1 in the least significant position to the B1C negative form. This convention results in only one form for 0 (00000000). It is the most common form for use in digital computers since it most conveniently fits in with simple arithmetic devices.
- BCD (binary coded decimal). The unsigned form has one group of four bits for each decade.
- SBCD (signed binary coded decimal). The signed form has one group of four bits for each decade plus one extra bit that is the most significant and is set to 0 for positive and 1 for negative.

7.3.5 Statement numbers

Statement numbers are number forms in their own right, namely, unsigned decimal integers of restricted length. These numbers are used solely for identifying statements in an ATLAS requirement. The number is six digits long. The leading four digits are called the test number, and the final two digits the step number. The test number may be replaced by four blank characters if the test number of a statement is the same as that of the previous statement; otherwise, when written, it must have four digits and include leading 0s as fillers when required. Using statement numbers is recommended but not mandatory. When the statement number is omitted, it must be replaced by six blank characters.

7.4 User-defined labels

ATLAS language rules allow users to create labels that are not listed as keywords for purposes that are determined by the context in which they are to be used. Data types (see 7.5), constants (see 7.6) and variables (see 7.7) are considered in this clause.

Any number of characters from a relevant subset of the ATLAS set can be linked to make a label. Characters are chosen to avoid delimiting problems in the context in which the string is used. Obviously a user should not introduce ambiguity by using identical strings with different meanings. A difference in one character within the first 16 characters of the string guarantees uniqueness (unless that difference is a blank space, which is considered to be insignificant in variables and defined labels).

Within the CNX field, a blank space is significant since it is the separator between connection designators.

Names created by the user, apart from connection designations, are called labels and must be enclosed within single quotes. See Table 7.

One reason for allowing user-defined names is to improve the readability of the test documentation. Meaningful labels, neither too abbreviated nor too lengthy, should be used. ATLAS module structures may be referenced by more than one program. Care is essential in the choice of labels and their categorization as EXTERNAL in module preamble structures to avoid duplication of labels that have global application.

7.4.1 Repeated test information

Where test information or data are used repeatedly throughout the test requirement, ATLAS provides a method of writing this information once in the preamble. The information is allocated a name in a preamble, and the name is recognized from then on as a label. Calling that label during the remainder of the test requirement accesses all the test information allocated to it.

Table 7—Examples of user designated names (labels)

ATLAS label (user-defined name)	Comment
'THREE PHASE POWER 115 V'	Identical up to the 17th significant character.
'THREE PHASE POWER 120 V'	
'THREE PHASE POWER 200 V'	Unique at the 16th significant character.
'AC POWER'	Considered identical as the space is not significant.
'ACPOWER'	
'AC-POWER'	The hyphen ensures this label is different from those above.
SKT-1-5	Connection identifier (Not a label). No spaces are allowed.
'INPUT PIN'	Most likely a connection label. A variable that contains a connection designation.
'115 V THREE PHASE POWER'	The uniqueness of the label should be established as early as possible in the string. (These labels differ at second character).
'120 V THREE PHASE POWER'	

7.4.2 Meaningful names (labels)

Characteristics associated with a label of a DEFINE statement must be applicable to the syntax and semantics of any statement that subsequently references that label. Although any arrangement of alphanumeric characters can be used as a label and a compiler or test system will recognize it, choosing labels that are meaningful to the human reader helps the reader interpret the requirement more easily. Examples of meaningful labels include 'GAIN-CHECK', 'POWER-ON', and 'DC-PSU-1'.

7.4.3 Label control

Where the ATLAS test requirement is so extensive and may be generated by more than one user, overall label control should be exercised. When only uppercase letters are available in a subset, overall label control ensures that all sections of the final ATLAS test requirement use the same labels for the same variables. This consistency may be unimportant to a machine, but it can assist human interpretation. Ensuring label uniqueness is not difficult, even if it is achieved only by adding a prefix or suffix to an otherwise common label. When organizing overall label control, all labels introduced in the preamble structure should be covered.

7.5 Types of ATLAS data

7.5.1 Base types

Seven base types of data (six base types for ARINC Specification 626-3) exist, upon which all other data types are built. Table 8 lists the base types and the two most common structured types.

Table 8—ATLAS data types

Type	Contents
INTEGER	A base 10 numeric value that has no fractional part.
DECIMAL	A base 10 fixed point or floating point number that may have a fractional part.
LONG-DECIMAL	A base 10 fixed point or floating point number that is beyond the range or precision for DECIMAL (IEEE Std 716-1995 only).
BIT	A digital number specifying one bit.
STRING OF BIT ^a	A digital number specifying more than one bit.
CHAR	A character string specifying one character.
STRING OF CHAR ^a	A character string specifying more than one character.
ENUMERATION	Constants ordered by the sequence in which they are written into the declaration.
CONNECTION	A connection, COMMON, EARTH, or ATMOS preceded by the word CONNECTION.

^aThese are structured types, i.e., STRING OF <base type>.

7.5.2 Structured types

Four structured types are developments of base types: STRING OF <type>, ARRAY OF <type>, RECORD OF <type> and FILE OF <type>. A structured type is in itself a 'type' so that it is possible to define complex types. The most common structured types include STRING OF BIT and STRING OF CHAR as shown in Table 8, but many other common structured types exist, such as ARRAY OF ARRAY OF DECIMAL (a two dimensional array), RECORD OF... (containing several items of different types), and FILE OF RECORD OF... (which would be used for saving all the data for a UUT, such as date, operator, serial number, results, pass criteria, and so on).

STRING OF BIT is used for variables that represent digital numbers. It specifies the number of bits the digital number may occupy.

STRING OF CHAR is used for character strings, such as messages.

ARRAY OF is used to declare a number of memory locations. Declaring multidimensional arrays is possible.

RECORD OF is used to declare a collection of data items that need not be of the same type or structure. Individual fields within a record are given field labels.

FILE OF is used to declare a collection of data elements where each item has a position in the FILE beginning with zero and advancing by one with each subsequent data element. The three classes of file are

- Known type. A known type file consists of a sequence of data items all of which have the same type.
- Untyped. An untyped file consists of bytes of data that could represent anything. Input and output for this type of file requires a detailed knowledge of the data format, which may be machine dependent. (This class is not used in ARINC Specification 626-3.)

- c) Text. A text file consists of a stream of characters that equate to declared data types, after possible automatic input/output (I/O) conversion.

7.5.3 Predeclared enumeration types

The three predeclared enumeration types are BOOLEAN, DIG-CLASS, and CHAR-CLASS. These types exist as though each ATLAS program contained an enumeration type declaration.

For BOOLEAN the enumeration elements are TRUE and FALSE.

For DIG-CLASS the enumeration elements are BNR, BSM, B1C, B2C, BCD, and SB CD.

For CHAR-CLASS the enumeration elements are ASCII7 and ISO7 (not used in ARINC Specification 626-3).

Predeclared enumerations differ from user-declared enumerations in that they are not enclosed by single quotes.

7.6 Constants

A constant is the label chosen by the test program writer to describe an implementation-time location that holds numerical, literal, connection, or enumeration data for use in signal orientated, computational, logical, or I/O operations. The value of the constant does not change throughout the entire execution of the program.

Each constant has a specific type and may be used in expressions only as permitted by the type rules for variables. The constant type is determined by the form in which it is written. Constants must be declared in either the main preamble or a local preamble. Constants that are declared in different preambles can use identical labels. The scope during which a constant is valid is determined by the type of preamble. Constants declared in the main preamble are global, and access to them can be made from any subsequent statement except when the same label has been declared in a local preamble. In this case the local constant takes precedence in those procedural statements directly or indirectly associated with the local preamble containing the identical label. It is recommended that all constants and variables be given different labels to avoid any possibility of confusion.

The following shows some examples of declarations of constants:

```
DECLARE, CONSTANT, 'PI' IS 3.14159265 $
```

```
DECLARE, CONSTANT, 'MESSAGE' IS C'THIS IS A MESSAGE' $
```

```
DECLARE, CONSTANT, 'MASK' IS X'FF00' $
```

In the above examples, the type is determined by the literal value assigned to the constant label. 'PI' is DECIMAL, 'MESSAGE' is STRING (17) OF CHAR and 'MASK' is a STRING (16) OF BIT.

7.7 Variables

Similar to constants, variables are labels chosen by the user to describe implementation time locations holding numerical, literal, or logical data for use in signal orientated, computational, logical, or I/O operations. The value of variables may be updated during the course of the execution of the program.

Examples of declarations of variables are shown below:

```
DECLARE, VARIABLE, 'NUMBER' IS DECIMAL $
```

```
DECLARE, VARIABLE, 'PARITY' IS BIT $

DECLARE, VARIABLE, 'PIN1' IS CONNECTION (J1) $

DECLARE, VARIABLE, 'DAY' IS ENUMERATION
    ('MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN') $

DECLARE, VARIABLE, 'COMPARISON' IS BOOLEAN $

DECLARE, VARIABLE, 'ADDRESS' IS STRING (8) OF BIT $

DECLARE, VARIABLE, 'DATA' IS ARRAY (1 THRU 10) OF STRING (8) OF BIT $
```

The last statement would declare 10 memory locations of 8 bits each. A multidimensional array may be defined. The following statements illustrate two ways of declaring the same two dimensional 10 × 5 array of memory locations, each one 8 bits long:

```
DECLARE, VARIABLE, 'BIT-ARRAY' IS ARRAY (1 THRU 10, 1 THRU 5)
                                     OF STRING (8) OF BIT $

DECLARE, VARIABLE, 'BIT-ARRAY' IS ARRAY (1 THRU 10) OF ARRAY (1 THRU 5)
                                     OF STRING (8) OF BIT $
```

An example of a record declaration could be

```
DECLARE, VARIABLE, 'DATE' IS RECORD OF
    ['DAY', 'MONTH', 'YEAR' IS INTEGER] $
```

Type declarations may be used to avoid repetition of identical descriptions of data structures in VARIABLE declarations. For instance, suppose two similar digital variables 'ADDRESS' and 'DATA' are declared as follows:

```
DECLARE, VARIABLE, 'ADDRESS' IS ARRAY(1 THRU 10) OF STRING(8) OF BIT $
DECLARE, VARIABLE, 'DATA' IS ARRAY(1 THRU 10) OF STRING(8) OF BIT $
```

It would be better to declare a type 'DIG-NUMBER' as follows:

```
DECLARE, TYPE, 'DIG-NUMBER' IS ARRAY (1 THRU 10) OF STRING(8) OF BIT $
```

and then to declare 'ADDRESS' and 'DATA' as being of the type 'DIG-NUMBER':

```
DECLARE, VARIABLE, 'ADDRESS', 'DATA' IS 'DIG-NUMBER' $
```

7.8 Initializing variables

Values may be assigned to variables by two methods. During program execution, the CALCULATE statement is used to assign a new value to a variable at any time. New values may be assigned to a variable many times during the course of program execution. It is also possible to assign an initial value to a variable when it is established by the DECLARE statement.

For example:

```
DECLARE, VARIABLE, 'NUMBER' IS DECIMAL INITIAL = 0.0 $
```

```

DECLARE, VARIABLE, 'PARITY' IS BIT INITIAL = B'0' $

DECLARE, VARIABLE, 'PIN1' IS CONNECTION (J1,J2,J3) INITIAL = J1 $

DECLARE, VARIABLE, 'DAY' IS ENUMERATION
  ('MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN') INITIAL = 'SUN' $

DECLARE, VARIABLE, 'COMPARISON' IS BOOLEAN INITIAL = FALSE $

```

7.9 Connections

Connections on the UUT and on special hardware defined in DEFINE DRAWING statements (available in ARINC Specification 626-3) obviously have descriptions assigned by the manufacturer. In ATLAS statements connections are defined in the CNX field. Each description of a connection will be an alphanumeric string without a space between any of the characters. The apostrophe, currency symbol, comma, left and right parentheses, and equal sign are forbidden characters. When only uppercase letters are available in a system, lowercase letters are represented by preceding each uppercase letter by a slash '/'. The slash character must not be used for any other purpose in the CNX field. Separators between a connector identifier and individual connector pins should be hyphens, for example

J3-/A	is Jack 3, pin a.
PLA3-12	is Plug A3, pin 12.
SKT6-/B/B	is Socket 6, pin bb.

ATLAS keywords, in particular pin descriptors, are not permitted as connection identifiers.

7.10 Classification of the language elements

The syntax of ATLAS is very similar to that of any natural language in that it comprises verbs, nouns, modifiers (adjectives), connectives (conjunctions), and delimiters (punctuation). An ATLAS statement is very similar to a natural language sentence. The verb is the action word, especially when that verb is used in the imperative (command) mood. The majority of ATLAS statements are written in the imperative mood and require the verb-predicate format of the natural language. For uniformity of statement syntax, this order is maintained for all statements, even when there is no command or, in some cases, only a pseudo verb. When creating an ATLAS requirement, the user must select the appropriate verb for each statement. Selecting the verb will also determine the syntax of the statement. The connectives (keywords) and delimiters required in the statement will be apparent from following the syntax diagram. Clearly, selecting the correct verb, together with any appropriate nouns and noun modifiers, is essential to creating a meaningful test requirement. Writing ATLAS test requirements is no different from writing test requirements in English (or any other natural language) in that engineering skills (rather than programming skills) are paramount.

7.11 Verbs

In IEEE Std 716-1995 and ARINC Specification 626-3, the verbs are presented in groups according to their functions. This grouping is repeated here to present the scope and power of the language. Most of the verbs have a function related to test engineering, and only a few have a general programming role.

In the following lists

- Items marked ¶ are only in IEEE Std 716-1995.
- Items marked § are only in ARINC Specification 626-3.
- The parenthesized information refers to clauses in IEEE Std 716-1995 and chapters in ARINC Specification 626-3.

7.11.1 Structure delimiting (4.0)

BEGIN ATLAS MODULE
BEGIN ATLAS PROGRAM
BEGIN BLOCK §
COMMENCE MAIN PROCEDURE §
END BLOCK §
LEAVE BLOCK §
TERMINATE, ATLAS MODULE
TERMINATE ATLAS PROGRAM

7.11.2 Preamble (6.0)

DECLARE
DEFINE COMPLEX SIGNAL ¶
DEFINE DIGITAL CONFIGURATION
DEFINE DIGITAL SENSOR
DEFINE DIGITAL SOURCE
DEFINE DIGITAL TIMING ¶
DEFINE DRAWING §
DEFINE EXCHANGE
DEFINE EXCHANGE CONFIGURATION
DEFINE FUNCTION §
DEFINE PROCEDURE
DEFINE SIGNAL
DEFINE TIMED DIGITAL PROCEDURE
END COMPLEX SIGNAL ¶
END DIGITAL CONFIGURATION
END FUNCTION §
END PROCEDURE
ESTABLISH, BUS PROTOCOL
EXTEND ATLAS CONNECTIONS
EXTEND ATLAS MODIFIERS
EXTEND ATLAS NOUN ¶
EXTEND ATLAS NOUNS §
EXTEND ATLAS databus
IDENTIFY DIGITAL TIMER ¶
IDENTIFY EVENT BASED EVENT
IDENTIFY SIGNAL BASED EVENT
IDENTIFY TIME BASED EVENT
IDENTIFY EVENT INDICATOR
IDENTIFY EVENT INTERVAL
IDENTIFY TIMER
INCLUDE
LEAVE FUNCTION §
LEAVE PROCEDURE
REQUIRE
SPECIFY AS ¶
SPECIFY COMPLEX FUNCTION ¶
SPECIFY COMPLEX SIGNAL ¶
SPECIFY DELAY-MIN §
SPECIFY SIGNAL-CONDITIONING ¶

7.11.3 Data processing (8.0)

CALCULATE
COMPARE

7.11.4 Input/output (9.0)

CREATE FILE §
DELETE FILE §
DISABLE FILE ACCESS
ENABLE FILE ACCESS
INPUT
OUTPUT

7.11.5 Program Control (10.0)

DISABLE DIGITAL CONFIGURATION
DISABLE ESCAPE §
DISABLE ESCAPE TO PROCEDURE ¶
DISABLE MANUAL INTERVENTION §
ELSE
ELSE IF THEN §
ENABLE DIGITAL CONFIGURATION
ENABLE ESCAPE TO PROCEDURE
ENABLE MANUAL INTERVENTION §
END FOR
END IF
END WHILE
FINISH
FOR THEN
GO TO
IF THEN
LEAVE FOR
LEAVE IF
LEAVE, WHILE
LEAVE PROCEDURE'
PERFORM
RESUME FOR §
RESUME WHILE §
WAIT FOR MANUAL-INTERVENTION §
WHILE THEN

7.11.6 Signal-oriented (11.0)

APPLY
ARM
CHANGE
CONNECT
DISABLE COMPLEX SIGNAL ¶
DISABLE EVENT
DISCONNECT

ENABLE COMPLEX SIGNAL ¶
ENABLE EVENT
FETCH
INITIATE
MEASURE
MONITOR
PROVE
READ
REMOVE
RESET
SENSE
SETUP
STIMULATE
VERIFY

7.11.7 Timing (12.0)

DO SEQUENTIAL §
DO SIMULTANEOUS
DO TIMED DIGITAL
END DO
READ TIMER
RESET DIGITAL TIMER ¶
RESET TIMER
WAIT FOR DIGITAL TIMER ¶
WAIT FOR EVENT
WAIT FOR TIME
WAIT FOR TIMER

7.11.8 Macro statements related to data-bus testing (13)

CONNECT EXCHANGE-CONFIGURATION
DISABLE EXCHANGE-CONFIGURATION
DISCONNECT EXCHANGE-CONFIGURATION
DO EXCHANGE
ENABLE EXCHANGE-CONFIGURATION
FETCH EXCHANGE
FETCH PROTOCOL
UPDATE EXCHANGE
UPDATE PROTOCOL

7.11.9 Relationship between multiple- and single-action verbs

One of the ground rules for the development of ATLAS is “a procedure shall be written in a step-by-step manner.” Therefore, some verbs are described in terms of other simpler verbs. This grouping creates the terms multiple-action signal-oriented verbs and single-action signal-oriented verbs. Further subgrouping produces the terms source (and load) and sensor. The relationships between single-action and multiple-action verbs are shown in Table 9 (for IEEE Std 716-1995). The order in which the single-action verbs occur varies depending on whether the multiple-action verb is acting on a source, load, or sensor resource. In ARINC Specification 626-3, the single actions occur in the same order for sources, loads, and sensors. These relationships are specified for each verb in detail in the ATLAS standards.

Table 9—Relationship between multiple-action and single-action verbs

Multiple-action verb	Used as a source	Used as a sensor	Used as a load
APPLY	CONNECT SETUP ENABLE EVENT ^a	n/a	SETUP CONNECT ENABLE EVENT ^a
REMOVE	DISABLE EVENT ^a RESET DISCONNECT ^a	DISABLE EVENT ^a DISCONNECT RESET	DISABLE EVENT ^a DISCONNECT RESET
MEASURE	n/a	SETUP CONNECT ARM ENABLE EVENT ^a FETCH DISABLE EVENT ^a DISCONNECT RESET	n/a
MONITOR	n/a	SETUP CONNECT ENABLE EVENT ^a ENABLE (manual intervention) WHILE (no manual intervention) ARM FETCH OUTPUT reading END WHILE DISABLE (manual intervention) DISABLE EVENT ^a DISCONNECT RESET	n/a
VERIFY	n/a	SETUP CONNECT ARM ENABLE EVENT ^a FETCH COMPARE DISABLE EVENT ^a DISCONNECT RESET	n/a
READ	n/a	ARM ENABLE EVENT ^a FETCH	n/a
INITIATE	n/a	ARM ENABLE EVENT ^a	n/a

^aRefers to resources with event-monitoring capabilities.

A source or load verb is one that is appropriate to use with source or load functioning of the equipment. For example, APPLY may be used to force a signal to the UUT or to place a load on the UUT. APPLY may not be used to perform a measurement function.

A sensor verb may be used for measurement functions, for example, MEASURE or VERIFY. These verbs acquire information about the UUT but do not affect it in any way.

The multiple-action verb REMOVE and several of the single-action verbs are relevant to source, load, and sensor resources.

7.11.10 State diagrams

IEEE Std 716-1995 and ARINC Specification 626-3 contain state diagrams to illustrate the relationship between single-action and multiple-action verbs. These diagrams show the states that can be adopted by a test resource and the usual transitions that occur. Any transitions that are not shown cannot occur when using multiple-action verbs. When using single-action verbs, forcing a resource to perform almost any low-level transition is possible, but could cause damage to a UUT or test system if the programmer does not exercise special care and consideration. For example, hot switching a relay that is carrying a high current could weld the relay contacts and damage the test system. Furthermore, these closed contacts could cause an incorrect or unexpected signal to be applied to the UUT and damage the UUT. Therefore, use of multiple-action state-ments is recommended wherever practicable. Figure 6 is the state diagram for an analog sensor resource with an event monitor. This example is taken from IEEE Std 716-1995.

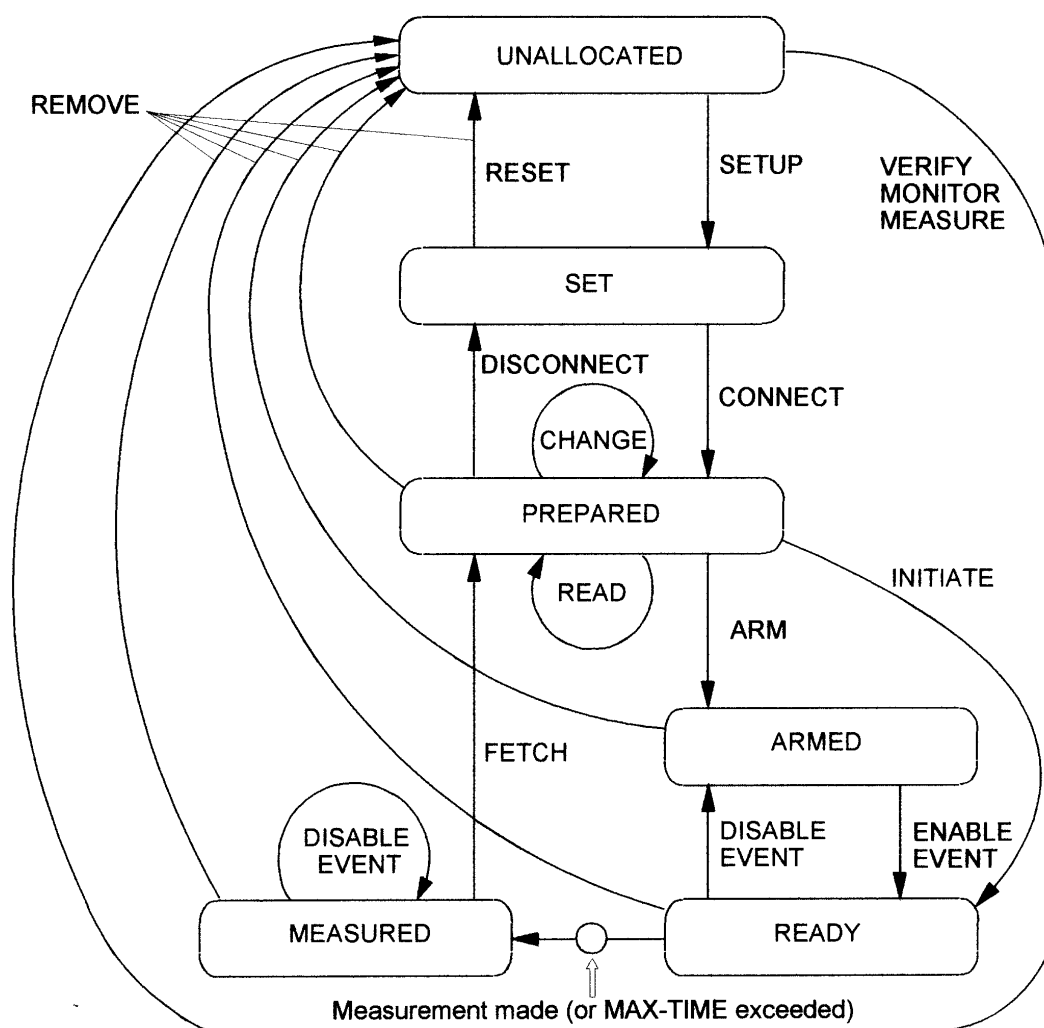


Figure 6—State diagram for an analog resource (with event monitor)

At the start of the test requirement, all resources are unallocated. The operation of the state diagram can be explained using the example of a MEASURE statement. The line for MEASURE runs completely around the diagram and indicates that the resource goes through each of the transitions and states necessary to complete the measurement and return to its original unallocated state. It first completes the SETUP transition to become set. This step puts the resource in the correct state (voltage range, for example) so that it will not be damaged by the output from the UUT. The next transition is CONNECT, and the resource is placed in the prepared state. The resource is now connected to the UUT. The next transition is ARM, and the resource is put in the armed state so that it can make a measurement. The next action is to ENABLE the EVENT that is going to trigger the measurement. The resource is then in the ready state. When the specified event occurs, the measurement is performed; the resource is then in the measured state. If the event did not occur, the resource (and test system) would be waiting indefinitely. Specifying a MAX-TIME value will prevent this situation; and, if this time expires before the measurement is made, the resource is automatically placed in the measured state so the program can continue. The result is then FETCHed from the resource (and the EVENT DISABLEd) returning it to the prepared state. The final two transitions (DISCONNECT and REMOVE) return the resource through the set state to the original unallocated state.

Note that READ and CHANGE do not perform the same function. The READ path is drawn below the prepared box and indicates that all states below the box are transited (see Table 9). The CHANGE line is above the prepared box and indicates that the resource remains in the prepared state (perhaps set to a different value).

7.12 Nouns and modifiers

In IEEE Std 716-1995 and ARINC Specification 626-3, each noun has a list of associated noun modifiers so that only semantically appropriate or sensical combinations can be formed. Modifiers, together with dimensional quantity or modifier descriptors, form statement characteristics to qualify the nouns. The complete alphabetic lists of nouns and noun modifiers are shown in Clause 11 of this guide. To determine which modifiers are available for use with any particular noun, the user must refer to the clause titled “Nouns and their modifiers” in the appropriate standard.

8. Writing a test requirement in ATLAS

8.1 Introduction

Test requirements can be complex. ATLAS has many different constructs to deal with these complexities. The syntax diagrams; the various charts showing valid combinations of nouns, modifiers, and dimensions; common sense; and, above all, a thorough knowledge of the test requirements are the tools available for writing ATLAS test requirements. It is impossible to present examples of every test situation. Newcomers to ATLAS will probably feel uncertain until they gain experience in using the language. If possible, newcomers should attend a formal training course to acquire basic knowledge. Planning the testing is still the most important part of writing a requirement.

8.2 Analysis of the test problem

The purpose and depth of testing are dependent on the total environment of the UUT. During design and development, the product is tested to see whether it meets design requirements. In production, the aim is repeatability at an economic cost. In service, the customer is monitoring continued satisfactory performance and effecting a repair if the testing shows up a fault condition. The means of effecting that repair are based on an agreed spare-part scheduling. The designers of the equipment should have considered all these factors. Feedback of failure information from production and service may produce short-term or long-term remedies to improve the equipment.

The value of all the test planning from design through in-service use should be assessed according to its effectiveness to prove the design, prove functioning in production and in use, and locate failures to scheduled replaceable parts. To do otherwise merely wastes the in-service life of the UUT.

Much has been written on these topics, and the reader is advised to consolidate his or her knowledge of the broader aspects of test planning and documentation.

8.3 Segmenting the test requirement

Several synonyms describe the breaking down of a major task into more manageable ones (e.g., modularization, structuring, partitioning, sectionalization, segmentation). Whether breakdown is determined by physical form, by signal flow, or by some other factor (such as the predicted or the actual mean time between failures of the parts) is not particularly important. Presenting the information logically is important. Programs for machines will be read by humans as well, especially when the equipment is modified at the same time. Therefore, writers should aim for simplicity by breaking down major tasks into smaller, more manageable ones.

The maintenance of test requirements is often ignored at the planning stage, but can be costly. Test requirement maintenance expenditures can be minimized by structuring the test requirement into independent blocks that ideally should not interact. Remember that maintenance segments will probably be written by someone other than the original writer(s) of the test requirement.

8.4 Flowcharting

Flowcharting is one method of documenting the design of a test program that has traditionally been used in the test industry. It is not necessarily the best method to use for any specific application and is not recommended as such. Several design methodologies should be considered by anyone creating test requirements. However, as the flowcharting method is still used in many military programs, the following comments concerning the use of flowcharts are included.

Flowcharting is committing to paper the visualization of the logical paths along which testing proceeds for both pass and fail conditions. For simple test problems, it can be omitted. For complex test problems, it may be used for the clarification of the objectives at each stage. The correct level of flowcharting should be selected at the outset by considering the eventual logistic situation.

The amount of detail included in the chart is a matter of common sense (see Figure 7). If the chart is written too high in the hierarchy, it can be meaningless; if it is too low, it may lead to frustration because the author could be writing the equivalent statements.

The benefits of flowcharts stem from their pictorial nature. These benefits are lost if the charts are not drawn properly.

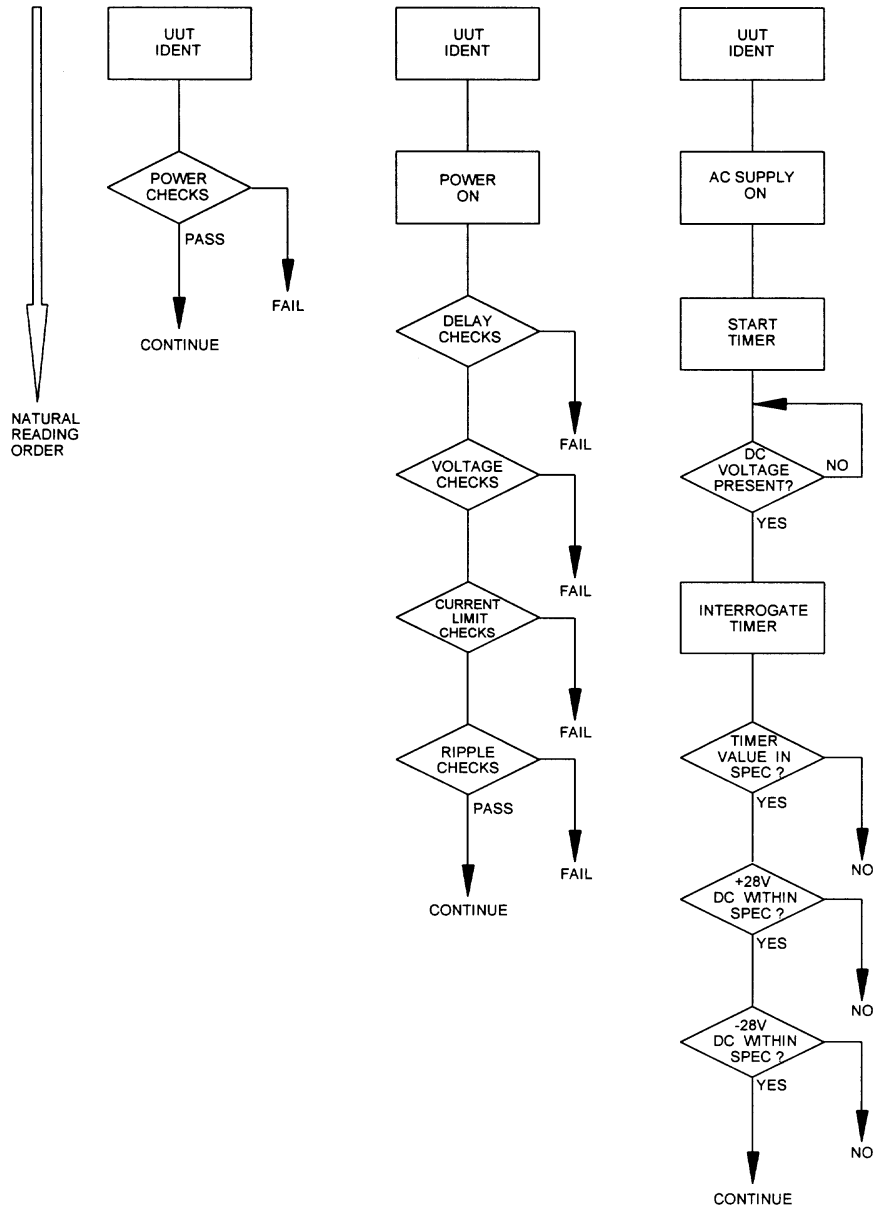


Figure 7—Examples of levels of flowcharts

8.5 Logical branching

Testing is based on one or more decisions. More than one decision implies logical branching. A test program is likely to assume the appearance of a many branched tree. When writing a test requirement, a straight line represents this topology, statement following statement. Two approaches are considered and shown diagrammatically in Figures 8 and 9.

Figure 8 represents a possible statement layout when the IF THEN ELSE structure is used, and Figure 9 represents a possible statement layout when the GO TO structure is used.

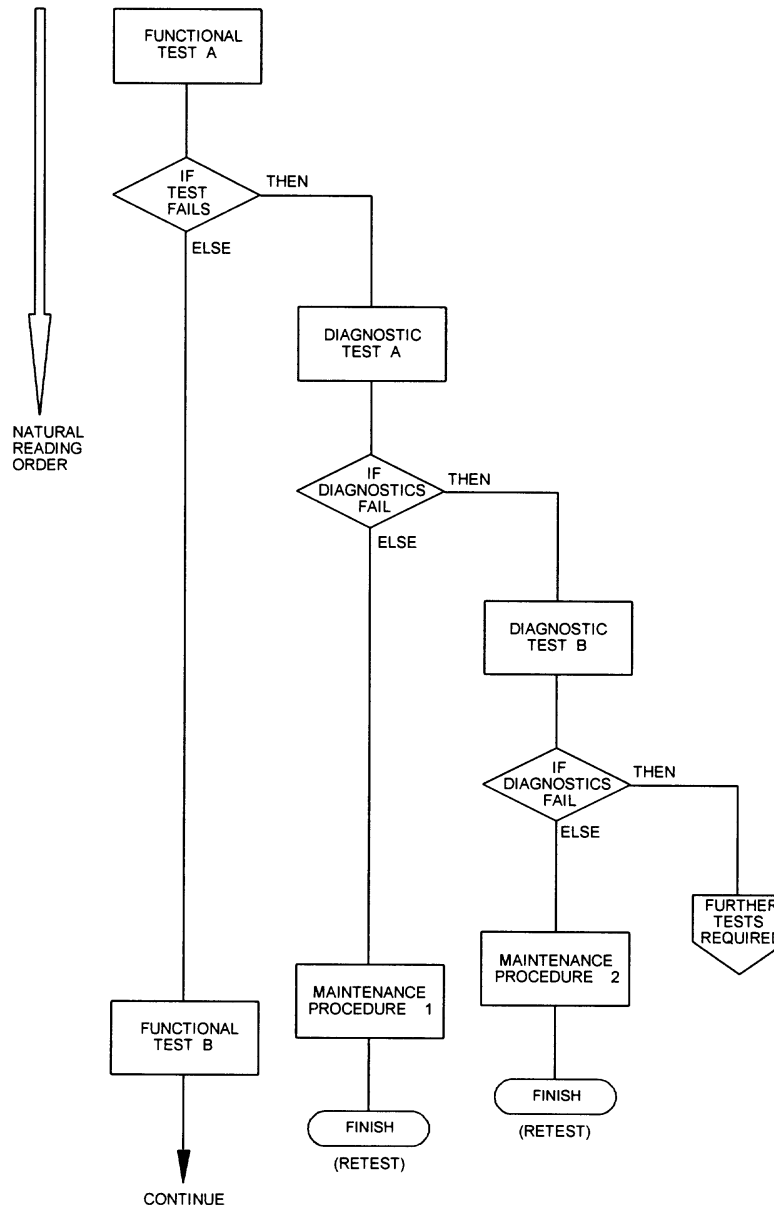


Figure 8—Example of program flowchart using IF THEN ELSE structure

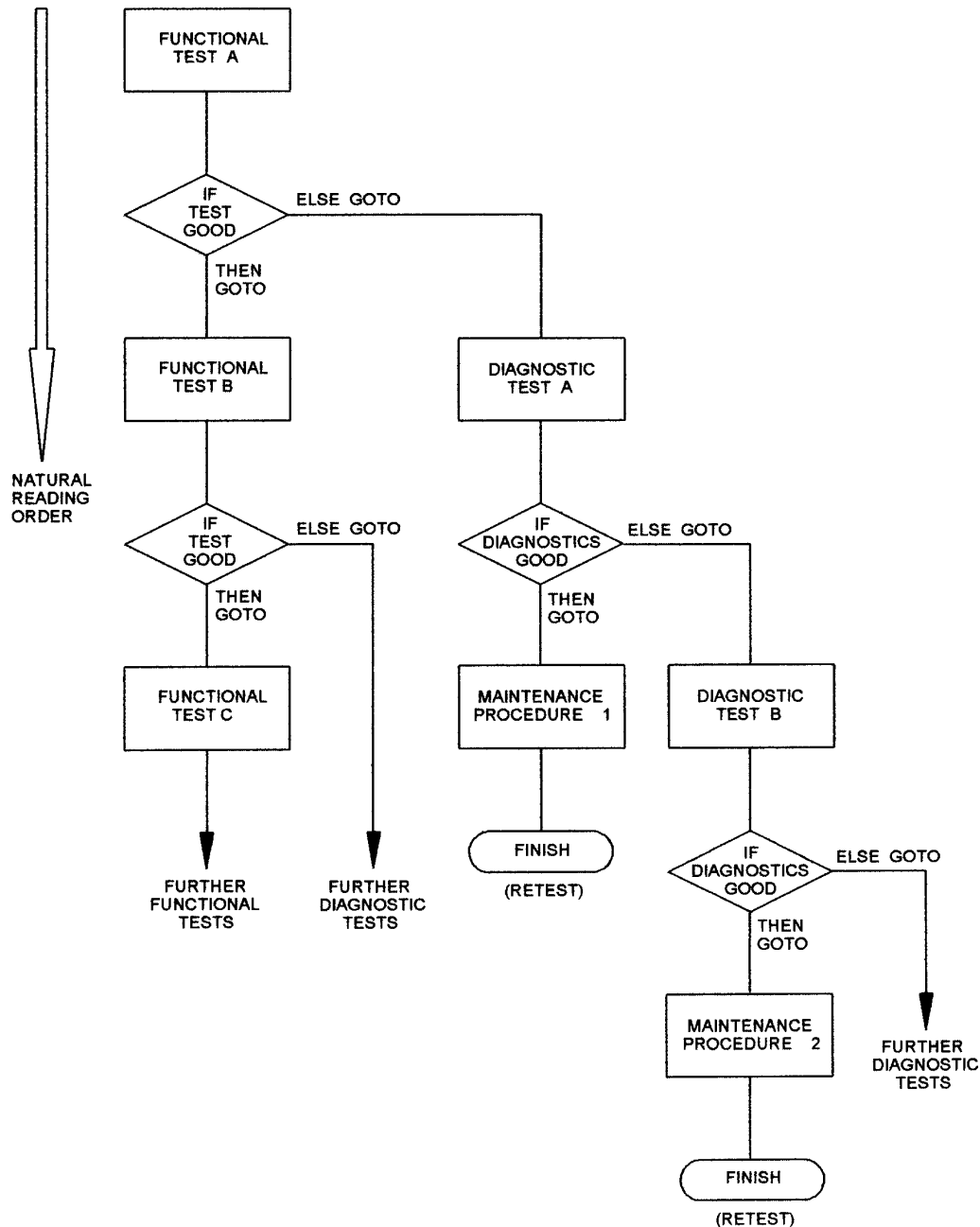


Figure 9—Example of program flowchart using GOTO structure

The conditions GO and NOGO (or THEN and ELSE) are not necessarily synonymous with the terms “pass” and “fail” respectively. Testing may, on occasion, be purposely written with a “reversed logical result” in mind. For example, an expected absence of a signal at a given time can be deduced from the failure to verify the presence of the signal at that time.

The GO TO statement remains in the ATLAS language to support legacy systems and programs. Many organizations list the GO TO statement as nonpreferred or do not allow its use in new programs. Usually an alternative method consistent with structured programming methods exists.

8.6 Test resources

When the test planning has been completed, assessing the minimal set of test resources required to execute the requirement will be possible. Frequently, the worst case condition will occur in a branch, and this possibility should not be overlooked. Additional resources may be required to cope with the parallel processing of tasks.

The total requirements can be listed in an optional group of REQUIRE statements (see 8.11 and 8.12).

8.7 Arrangement of test requirement

The major segmentation of any ATLAS requirement is into the main preamble and the procedural structures.

8.7.1 Preamble

The principle objectives of the preamble are

- a) To summarize and assign mnemonic labels to virtual resource requirements.
- b) To declare and assign mnemonic labels to test-data storage requirements.
- c) To define and assign mnemonic labels to procedural information of a global nature.
- d) To preload labelled storage spaces with data for use during the execution of the procedural structure.
- e) To list ATLAS and non-ATLAS modules that complement the program to form a complete test requirement.
- f) To describe and assign mnemonic labels to test and signal events and timing relationships for use in synchronizing and timing the execution of the procedural structure.
- g) To define and assign mnemonic labels to protocols for data exchange on buses.
- h) To define and assign a mnemonic label to a particular exchange (or transition) on a data bus.
- i) To define and assign mnemonic labels to digital configurations for use in digital tests.
- j) To define extensions (new nouns, modifiers, and connections) to the standard language required for new techniques.
- k) To identify and assign a mnemonic label to particular timing and synchronizing relationships.

8.7.2 Delimiting statements

Several statements in an ATLAS requirement appear redundant to a human operator but are essential for machine translation. Similarly, some of the punctuation is excessive by literary standards. Humans reading printed matter can assimilate at a glance enormous amounts of data from the page layout, chapter headings, paragraphing, and so on, that are not apparent when the same material is encoded. In the coded form, the delimiting statements provide the machine with this type of information, so that the machine can use the correct routines to translate each section to appropriate machine instructions. Delimiting statements always occur in pairs to mark the bounds between which items have meaning or validity.

For example,

```
000001 BEGIN, ATLAS PROGRAM 'EXAMPLE' $
.
001000 DEFINE, 'STTO', PROCEDURE $
.
.
001990 END, 'STTO' $
.
```

```

002000  DEFINE, 'POWER-ON', PROCEDURE $
.
.
002990  END, 'POWER-ON' $
.
003000  BEGIN, BLOCK 'SERVICEABILITY-TESTS' $
.
.
003950  END, BLOCK 'SERVICEABILITY-TESTS' $
.
004000  DO, TIMED DIGITAL, STIM-EVENT 'CLOCK' $
.
.
004810  END, DO $
.
100000  TERMINATE, ATLAS PROGRAM 'EXAMPLE' $

```

Although the above example does not illustrate all the delimiting statements that can be written, the paragraphing that can be achieved should be evident. Further emphasis can be given to the breakdown of a program by the allocation of statement numbers or blocks of statement numbers to sections of the requirement. For example, all segments of a requirement should start with step number 00. The BLOCK structure is only available in ARINC Specification 626-3.

People prefer textual layout that is easy on the eye, whereas machines are untroubled by layout but require markers to indicate the divisions of the requirement. The chapter and paragraph headings that provide a rapid overview in normal test are a distinct aid to human readers. In ATLAS programs the same facility can be provided by well-structured test requirements with meaningful indentations and comment statements.

8.7.3 Preamble labels

Labels are established in an ATLAS preamble structure to provide identifiers for subsequent reference within the executable structures. A label is established by the writer by using the DECLARE, DEFINE, ESTABLISH, IDENTIFY, REQUIRE, and SPECIFY statements.

8.7.3.1 DECLARE labels

In the preamble, storage of data can be established using labels that are meaningful for the classes of data to be stored. Although it is not mandatory to arrange DECLARE statements in the order of class or type, a test writer should always bear in mind the test requirement reader. The DECLARE statements shall be written in the following authorized locations:

- a) The main program preamble. Labels should be declared with:
 - 1) A GLOBAL attribute if the label is established within the preamble and referenced in other included modules.
 - 2) An EXTERNAL attribute if the label is declared within an included module as GLOBAL.
 - 3) If no attribute is assigned to the label, it is accessible from the whole main program but inaccessible from other modules.
- b) A module preamble. Labels should be declared with:
 - 1) A GLOBAL attribute if the label is established within the module preamble and referenced in other modules or the main program.
 - 2) An EXTERNAL attribute if the label has been declared within the main program preamble or another module.