

Standard Test Language for All Systems—Common/Abbreviated Test Language for All Systems (C/ATLAS)

Sponsor
716 C/ATLAS Subcommittee
of the
IEEE Standards Coordinating Committee 20

Approved March 16, 1995
Reaffirmed December 7, 2011

IEEE-SA Standards Board

Abstract: A high order language for testing is defined. This language is designed to describe tests in terms that are independent of any specific test system, and has been constrained to ensure that it can be implemented on automatic test equipment.

Keywords: abbreviated, ATLAS, C/ATLAS, common, language, test language, unit under test, UUT

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394 USA

Copyright © 1995 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1995. Printed in the United States of America.

ISBN 1-55937-518-3

No part of this publication may 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 Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. 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.

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.

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 all 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 technical 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 on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

IEEE Standards documents may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user to such technology to obtain all necessary permissions.

Introduction

[This introduction is not part of IEEE Std 716-1995, IEEE Standard Test Language for All Systems—Common/Abbreviated Test Language for All Systems (C/ATLAS).]

IEEE Std 716 (C/ATLAS) originally was a companion language to IEEE Std 416 (ATLAS). This revision of the C/ATLAS standard incorporates 16 separately balloted changes. It was approved by the 716 Subcommittee in November, 1993, and by the IEEE Standards Board in 1995.

C/ATLAS has been conceived as a widely acceptable ATLAS set, with a single and unique construct to express each language facility selected to support structured programming. Major emphasis was placed on long-term stability between C/ATLAS updates.

This publication represents the culmination of numerous reviews, analyses, and assessments by representatives of the major users of C/ATLAS, as well as representatives of other corporations. The work was conducted by the IEEE SCC20 ATLAS Committee and its 716 C/ATLAS Subcommittee. Major users of C/ATLAS participating in this effort include the United States Department of Defense, the German Ministry of Defense, the French Ministry of Defense, the United Kingdom Ministry of Defense, the Swedish Ministry of Defense, and the airline industry as represented by Aeronautical Radio, Inc.

At the time this revised standard was completed, the 716 Subcommittee had the following membership:

Richard Churcher, *Chair*

Guy Adam	Bernd Eichenauer	Rolf Lundin
Christine Augéard	Wolf-Dieter Erdmann	Michael Martin
Michel Bantegnie	Horst Ey	Jan Meeusen
Jean-Yves Barbier	Anthony Fitzpatrick	Goran Nordstrom
Eric Bergaud	Jacques Flandrois	Hans Obermeier
Michael Blair	Ken Fox	Catherine Ozenfant
Patrice Brard	George Gohagan	Richard Patrick
Barry Burnage	Alf Gustafsson	Jean Pouilly
Antonius Bunsen	Arthur Hann	Brian Raffaelli
Paul Charbonnier	Joachim Hannemann	Johannes Reh
Patrice Chassard	Michael Harrison	Albert Sanson
Michael Cosslett	Raymond Heather	Helmut Scheibenzuber
James Cottrell	Ashley Hulme	Bruno Schlecht
Michel Courtois	Ulla Jacobsson	Michael Seavey
Bernard Dathy	Helmuth Kaunzinger	Joseph Stanco
Bernd Dinklage	Sonia Kval	Jack Taylor
Pascal Dormal	Michel Lacroix	Jacques Tete
Bernard Dumont	Lars Olav Larsson	Will Young

Review and comments were contributed by the following individuals:

Aldo Bernal	Chris Gorringe	Harry McGuckin
K. Ellis	Brian Kett	David Morris
Christine Fisher	Graham Matchett	David Pawson
John Goddard		Robert Rolfe

The following persons were on the balloting committee:

Guy Adam	Jacques Flandrois	Michael Martin
Michel Bantegnie	John Goddard	Robert McGarvey
Eric Bergaud	George Gohagan	Neil McKinnon
Michael Blair	Michael Gooding	David Morris
Patrice Brard	Chris Gorringe	Thomas Murphy
Alan Bridges	James Graves	Goran Nordstrom
Ronald Brooks	Arnold Greenspan	Hans Obermeier
Antonius Bunsen	Winston Griffin	Leslie Orlidge
William Byrnes	Alf Gustafsson	M. C. Oser
Paul Charbonnier	Arthur Hahn	Catherine Ozenfant
Richard Churcher	Damon Hart	Richard Patrick
Frank Conforti	Raymond Hasentab	John Pflaz
James Cottrell	Robert Hayes	Jean Pouilly
Michel Courtois	Raymond Heather	Brian Raffaelli
Robert Crawford	John Heiser	Narayanan Ramachandran
Jack Cross	Ashley Hulme	James Reeder
Bernard Dathy	Ulla Jacobsson	Robert Rolfe
Warren DeCamp	Gary Jenkins	Albert Sanson
George Diab	Helmut Kaunzinger	Helmut Scheibenzuber
Bernd Dinklage	Kerry Kugler	Michael Seavey
Pascal Dormal	Michel Lacroix	Philip Smith
Bernard Dumont	Lars Olov Larsson	Joseph Stanco
Marcie Emas	Fred Liguori	William Taylor
Wolf-Dieter Erdmann	Alfred Lowenstein	Vince Terpstra
Horst Ey	Linda Lund	Richard Weger
Anthony Fitzpatrick	Rolf Lundin	Kenneth Wilkinson
Christine Fisher		Will Young

When the IEEE Standards Board approved this standard on March 16, 1995, it had the following membership:

E. G. “Al” Kiener, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril
Clyde R. Camp
Joseph A. Cannatelli
Stephen L. Diamond
Harold E. Epstein
Donald C. Fleckenstein
Jay Forster*
Donald N. Heirman

Richard J. Holleman
Jim Isaak
Ben C. Johnson
Sonny Kasturi
Lorraine C. Kevra
Ivor N. Knight
Joseph L. Koepfinger*
D. N. “Jim” Logothetis
L. Bruce McClung

Marco W. Migliaro
Mary Lou Padgett
John W. Pope
Arthur K. Reilly
Gary S. Robinson
Ingo Rusch
Chee Kiow Tan
Leonard L. Tripp

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
Richard B. Engelman
Robert E. Hebner
Chester C. Taylor

Stephen J. Huffman
IEEE Standards Project Editor

Table of Contents

CLAUSE	PAGE
1.0 Scope and purpose	1
1.1 Language processors	1
1.2 Document control	1
2.0 Reference information	2
2.1 References	2
2.2 Document precedence	3
2.3 Document organization and conventions	3
2.3.1 Extensibility	3
2.3.2 Organization of syntax specification	3
2.3.3 Guide to the use of the C/ATLAS language	4
3.0 Complete C/ATLAS test program	5
3.1 <complete atlas test program structure>	5
3.1.1 <atlas program structure>	5
3.1.2 <atlas module structure>	6
3.1.3 <non-atlas module structure>	6
3.2 Basic statement elements	7
3.2.1 Flag field	7
3.2.2 Statement number field	7
3.2.3 Verb field	7
3.2.4 Field separator	7
3.2.5 Remainder of statement	8
3.2.6 Statement terminator (\$)	8
4.0 Structure delimiter statements	9
4.1 BEGIN/TERMINATE statements	9
4.1.1 BEGIN, ATLAS PROGRAM statement	9
4.1.2 TERMINATE, ATLAS PROGRAM statement	9
4.1.3 BEGIN, ATLAS MODULE statement	10
4.1.4 TERMINATE, ATLAS MODULE statement	11
5.0 Reserved for future use	12
6.0 Preamble statements	13
6.1 Main preamble structure	13
6.1.1 <program preamble structure>	13
6.1.2 <module preamble structure>	14
6.2 <local preamble structure>	15
6.3 DECLARE statement	16
6.4 DEFINE statements definition	27
6.5 DEFINE <signal> statement	28

CLAUSE	PAGE
6.6	PROCEDURE definition 32
6.6.1	<define procedure structure> 32
6.6.2	DEFINE PROCEDURE statement 33
6.6.3	PROCEDURE body 34
6.6.4	LEAVE <procedure> statement 35
6.6.5	END PROCEDURE statement 36
6.7	REQUIRE statement 36
6.8	INCLUDE statement 41
6.9	IDENTIFY statements 42
6.10	IDENTIFY TIMER statement 43
6.11	IDENTIFY SIGNAL BASED EVENT statement 43
6.12	IDENTIFY EVENT BASED EVENT statement 46
6.13	IDENTIFY EVENT INTERVAL statement 47
6.14	IDENTIFY EVENT INDICATOR statement 49
6.15	IDENTIFY TIME BASED EVENT statement 50
6.16	DIGITAL CONFIGURATION definition 50
6.16.1	<define digital configuration structure> 50
6.16.2	DEFINE DIGITAL CONFIGURATION statement 51
6.16.3	DEFINE DIGITAL SOURCE statement 52
6.16.4	DEFINE DIGITAL SENSOR statement 61
6.16.5	END DIGITAL CONFIGURATION statement 66
6.17	EXTEND statement 67
6.18	ESTABLISH PROTOCOL statement 73
6.19	DEFINE EXCHANGE statement 80
6.20	DEFINE DIGITAL TIMING statement 84
6.21	COMPLEX SIGNAL definition 86
6.21.1	DEFINE COMPLEX SIGNAL structure 86
6.21.2	DEFINE COMPLEX SIGNAL statement 92
6.21.3	SPECIFY COMPLEX SIGNAL CHARACTERISTICS statement 92
6.21.4	SPECIFY COMPLEX FUNCTION statement 93
6.21.5	SPECIFY COMPONENT SIGNAL statement 97
6.21.6	SPECIFY SIGNAL CONDITIONING statement 98
6.21.7	END COMPLEX SIGNAL statement 100
6.22	DEFINE EXCHANGE-CONFIGURATION statement 101
7.0	Procedural structure 102
7.1	<main procedural structure> 102
7.2	<main procedural statements> 102
8.0	Procedural statements, data processing 104
8.1	CALCULATE statement 104
8.2	COMPARE statement 117

CLAUSE	PAGE
9.0 Procedural statements, input/output	118
9.1 INPUT statement	118
9.2 OUTPUT statement	123
9.3 ENABLE FILE ACCESS statement	128
9.4 DISABLE FILE ACCESS statement	129
10.0 Procedural statements, control	130
10.1 IF THEN ELSE capability	131
10.1.1 IF THEN ELSE structure	131
10.1.2 IF THEN statement	133
10.1.3 <procedural segment if then else>	134
10.1.4 LEAVE, IF statement	135
10.1.5 ELSE statement	135
10.1.6 END, IF statement	136
10.2 WHILE THEN capability	137
10.2.1 WHILE THEN structure	137
10.2.2 WHILE THEN statement	138
10.2.3 <procedural segment while then>	139
10.2.4 LEAVE, WHILE statement	139
10.2.5 END, WHILE statement	140
10.3 FOR THEN capability	141
10.3.1 FOR THEN structure	141
10.3.2 FOR THEN statement	142
10.3.3 <procedural segment for then>	144
10.3.4 LEAVE, FOR statement	145
10.3.5 END, FOR statement	145
10.4 GO TO statement	146
10.5 PERFORM statement	147
10.6 FINISH statement	148
10.7 ENABLE DIGITAL CONFIGURATION statement	148
10.8 DISABLE DIGITAL CONFIGURATION statement	149
10.9 Escape structure	150
10.9.0 <escape structure>	150
10.9.1 ENABLE ESCAPE TO PROCEDURE statement	150
10.9.2 DISABLE ESCAPE TO PROCEDURE statement	152
11.0 Signal oriented statements	154
11.1 <procedural statements signal>	154
11.2 Single-action statements	163
11.2.1 General description	163
11.2.2 SETUP statement	168
11.2.3 CONNECT statement	169
11.2.4 DISCONNECT statement	171
11.2.5 ARM statement	172
11.2.6 FETCH statement	174
11.2.7 CHANGE statement	175

CLAUSE	PAGE
11.2.8 ENABLE EVENT statement	177
11.2.9 DISABLE EVENT statement	177
11.2.10 ENABLE COMPLEX SIGNAL statement	178
11.2.11 DISABLE COMPLEX SIGNAL statement	179
11.2.12 RESET statement	179
11.3 Multiple-action statements	181
11.3.1 General description	181
11.3.2 APPLY statement	181
11.3.3 REMOVE statement	182
11.3.4 MEASURE statement	184
11.3.5 MONITOR statement	185
11.3.6 VERIFY statement	186
11.3.7 READ statement	187
11.3.8 INITIATE statement	188
11.4 Digital statements	189
11.4.1 General description	189
11.4.2 STIMULATE statement	190
11.4.3 SENSE statement	192
11.4.4 PROVE statement	194
12.0 Procedural statements, timing	198
12.1 Timing statements (general)	198
12.2 READ TIMER statement	198
12.3 WAIT FOR statement	199
12.4 RESET TIMER statement	200
12.5 DO/END DO capability	201
12.5.1 <do simultaneous structure>	201
12.5.2 DO SIMULTANEOUS statement	202
12.5.3 <do simultaneous body>	203
12.5.4 END DO statement	204
12.5.5 <do timed digital structure>	204
12.5.6 DO TIMED DIGITAL statement	204
12.5.7 <do timed digital body>	207
13.0 Procedural statements, databus	211
13.1 <procedural statement databus>	212
13.2 DO EXCHANGE statement	213
13.3 UPDATE EXCHANGE-CONFIGURATION statement	221
13.4 FETCH EXCHANGE-CONFIGURATION statement	223
13.5 ENABLE EXCHANGE-CONFIGURATION statement	225
13.6 CONNECT EXCHANGE-CONFIGURATION statement	227
13.7 DISCONNECT EXCHANGE-CONFIGURATION statement	227
13.8 DISABLE EXCHANGE-CONFIGURATION statement	228

CLAUSE	PAGE
14.0 Field and subfield definition	230
14.1 <statement characteristics>	230
14.2 <real characteristic subfield>	231
14.3 <digital characteristic subfield>	234
14.4 <sync subfield>	234
14.5 <real errlim> (real error limit)	236
14.6 <measured characteristic>	238
14.7 <evaluation field>	240
14.8 <eval statement characteristics> (evaluation statement characteristics)	244
14.9 <max time>	244
14.10 <time quantity>	245
14.11 <condition>	245
14.12 <conn> (connection field)	246
14.13 <conn set> (connection set)	246
14.14 <signal value>	253
14.15 <real quantity>	254
14.16 <index>	255
14.17 <noun field>	255
14.18 <dimensioned number>	256
14.19 <require control>	256
14.20 <require capability>	257
14.21 <require limit>	258
14.22 <require cnx>	259
14.23 <digital quantity>	259
14.24 <data store>	260
14.25 <array range>	260
14.26 <parameter>	262
14.27 <gate field>	262
14.28 <stim rate>	264
14.29 <sense rate>	264
14.30 <sense delay>	264
14.31 <stim event>	265
14.32 <sense event>	266
14.33 <when field>	266
14.34 <digital source characteristics>	267
14.35 <digital sensor characteristics>	268
14.36 <on field>	269
14.37 <exchange expression>	269
14.38 <exchange frame>	271
14.39 <bus parameter>	273
14.40 <set protocol parameter>	274
14.41 <role field>	276
14.42 <command field>	277
14.43 <data field>	278
14.44 <status field>	279
14.45 <mark descriptor subfield>	280
14.46 <proportionality subfield>	283
14.47 <sweep configuration subfield>	285

CLAUSE	PAGE
14.48 <fetch protocol parameter>	289
14.49 <databus fetch data>	291
14.50 <file>	292
15.0 Language component specification	293
15.1 Authorized characters	293
15.2 Number representations	293
15.2.1 <decimal number>	294
15.2.2 <unsigned decimal number>	294
15.2.3 <unsigned integer number>	295
15.2.4 <digital number>	295
15.2.5 <long decimal number>	295
15.3 Flag and statement numbers	295
15.3.1 <fstatno>	295
15.3.2 Flag	296
15.3.3 <statement number>	297
15.4 Blank space requirements	298
15.5 <label>	298
15.6 Retention of labelled information	300
15.6.1 Scope of labels and identifiers	300
15.6.2 Retention of measured sensor values	300
15.6.3 <label> for DEFINE and IDENTIFY statement	300
15.6.4 <parameter>	301
15.6.5 <data store>	301
15.6.6 Condition identifiers	301
15.6.7 Other variables ()	302
15.6.8 <requirement>	302
15.6.9 <exchange>	302
15.6.10 <complex signal>	302
15.7 Range subfield	302
15.7.1 Range requirement for <data store> or <parameter>	302
15.8 Dimensions	303
15.8.1 Units of rate of change	303
15.8.2 Units	308
15.8.3 Power dimensions	308
15.8.4 ISO characters	308
15.8.5 <dim>	308
15.8.6 <freq dimension>	308
15.8.7 <time dimension>	308
15.9 Character code definitions	308
15.10 Special character sets	313
15.10.1 <character string>	313
15.10.2 <message text>	313
15.10.3 <letter>	313
15.10.4 <digit>	313

CLAUSE	PAGE
15.10.5<modifier descriptor>	313
15.10.6<module name>	313
15.10.7<program name>	313
15.10.8<extend label>	313
15.11 Miscellaneous language elements	314
16.0 Nouns and their modifiers	320
16.1 AC SIGNAL (alternating current signal)	321
16.2 ADF (automatic direction finder)	323
16.3 AM SIGNAL (amplitude modulation signal)	323
16.4 AMBIENT CONDITIONS	324
16.5 ATC (air traffic control)	326
16.6 COMMON	328
16.7 COMPLEX SIGNAL <complex signal>	330
16.8 DC SIGNAL (direct current signal)	332
16.9 DISPLACEMENT	333
16.10 DME (distance measuring equipment)	333
16.11 DOPPLER	334
16.12 EARTH	335
16.13 EM FIELD (electro magnetic field)	335
16.14 EVENTS	336
16.15 FLUID SIGNAL	336
16.16 FM SIGNAL (frequency modulation signal)	338
16.17 HEAT	338
16.18 IFF (identification, friend or foe)	339
16.19 ILS (instrument landing system)	341
16.20 IMPEDANCE	343
16.21 LIGHT	344
16.22 LOGIC CONTROL	344
16.23 LOGIC DATA	347
16.24 LOGIC LOAD	354
16.25 LOGIC REFERENCE	354
16.26 MANOMETRIC	356
16.27 PAM (pulse amplitude modulation)	356
16.28 PM SIGNAL (phase modulated signal)	357
16.29 PULSED AC (pulsed alternating current signal)	358
16.30 PULSED AC TRAIN	359
16.31 PULSED DC (pulse direct current signal)	362
16.32 PULSED DC TRAIN	362
16.33 PULSED DOPPLER	365
16.34 RADAR SIGNAL	366
16.35 RAMP SIGNAL	367
16.36 RANDOM NOISE	367
16.37 RESOLVER	368
16.38 ROTATION	368
16.39 SHORT	373
16.40 SQUARE WAVE	374
16.41 STEP SIGNAL	374

CLAUSE	PAGE
16.42 SUP CAR SIGNAL (suppressed carrier signal)	375
16.43 SYNCHRO	376
16.44 TACAN (tactical air navigation)	376
16.45 TIME INTERVAL	382
16.46 TRIANGULAR WAVE SIGNAL	382
16.47 TURBINE ENGINE DATA	383
16.48 VIBRATION	384
16.49 VOR (VHF omnidirectional radio range)	385
16.50 WAVEFORM	387
17.0 Noun modifier definitions	391
17.1 Mnemonics for pulse-type signals	391
17.2 Modifier prefixes and suffixes	392
17.2.1 Pressure and temperature prefixes	392
17.2.2 Amplitude modifier suffixes	392
17.2.3 Phase identifier suffixes	394
17.2.4 Distance suffixes	395
17.2.5 Angle suffixes	395
17.2.6 Pulse identifier suffix	395
17.2.7 Reference identifier suffix	395
17.2.8 Composite vector suffixes	396
17.2.9 Quiescent signal suffix	396
17.2.10 Bi-phase digital signal suffixes	397
17.2.11 Bandwidth prefixes	397
17.3 Digital noun-modifiers	397
17.4 Noun modifiers	397
17.5 Function and function characteristics	447
17.6 Keyword definitions for the <specify signal conditioning statement>	451
17.7 Keyword definitions for the <mark descriptor subfield>	453
17.8 Keyword definitions for the <sweep configuration subfield>	454
17.9 Keyword definitions for the <step sweep descriptor>	455
18.0 Standard C/ATLAS syntax	457
18.1 Introduction	457
18.1.1 Purpose	457
18.1.2 Organization	457
18.1.3 References to the formal syntax definition	458
18.2 Syntax notation	459
18.2.1 Syntax language elements	459
18.2.2 Structure of a meta-program	460
18.2.3 Rule bodies	460
18.3 Program structure syntax	463
18.3.1 Range and order of the definition	464
18.3.2 Name conventions	464

CLAUSE	PAGE
18.4 Statements and field syntax	464
18.4.1 Range and order of the definition	464
18.4.2 Name conventions	464
18.4.3 Notes	465
18.5 Lexical symbols syntax	465
18.5.1 Range and order of the definition	465
18.5.2 Name conventions	466
18.5.3 Classes of lexical symbols	467
18.6 Syntax definition	468
18.6.1 Complete C/ATLAS test program	468
18.6.2 Structure delimiter statements	468
18.6.3 Preamble statements	469
18.6.4 Procedural structure	481
18.6.5 Procedural statements—data processing	481
18.6.6 Procedural statements—input/output	482
18.6.7 Procedural statements—control	483
18.6.8 Procedural statements—signal	486
18.6.9 Procedural statements—timing	490
18.6.10 Procedural statements—databus	492
18.6.11 Fields and subfields	494
18.6.12 Basic symbols and character sets	502
18.6.13 Identifiers and labels	504
18.6.14 Separators and statement-numbers	507
18.6.15 Operators	507
18.6.16 Descriptors	508
18.6.17 Nouns	509
18.6.18 Noun-modifiers	511
18.6.19 Noun-modifier prefixes and suffixes	515
18.6.20 Dimensions	516
18.6.21 Abbreviations	518
18.7 Cross-reference listings of syntax variables and symbols	519
18.7.1 Cross-reference list of rule names	519
18.7.2 Cross-reference list of literal symbols	539
19.0 Bibliography	592

1.0 Scope and purpose

IEEE Std 716-1995 defines a high order language for testing. This language is designed to describe tests in terms that are independent of any specific test system. It has been constrained to ensure that it can be implemented on automatic test equipment.

This document defines an operational standard that was originally based upon the language defined by the reference document for the ATLAS language, IEEE Std 416-1984, in that, for those cases where a capability was required that went beyond the existing reference document and working papers, or it was not possible to conveniently subset the 416 material, extensions were developed for IEEE Standard 716.

1.1 Language processors

Language processors conforming to this standard shall support all capabilities as specified within C/ATLAS down to the level of nouns, modifiers, dimensional units and pin descriptors. In addition, this support shall include all nouns, modifiers, dimensional units and pin descriptors that are necessary to support the target ATE.

1.2 Document control

All aspects of definition, creation, extensibility, management, maintenance, quality assurance and proposal handling shall be the responsibility of the Std 716 Subcommittee to the satisfaction of the Steering Committee of the IEEE Standards Coordination Committee 20 (SCC20)—Abbreviated Test Language for All Systems (ATLAS).

2.0 Reference information

2.1 References

This standard shall be used in conjunction with the publications listed below. Other documents cited in, and applicable to, this standard are listed in the Bibliography (clause 19).

ANSI X3.4–1986 (R1992), Coded Character Set—7-Bit American National Standard Code for Information Interchange (ASCII).¹

ARINC 570, Automatic Direction Finder.²

ARINC 572 and 711, Transponder and VOR Receiver.

ARINC 573, Aircraft Integrated Data Systems—Mark 2 (AIDS) for the NRZ Code.

ARINC 575, Sub Sonic Air Data Systems, Bi-Polar RZ Pulse Class.

ARINC 578, Instrument Landing System.

ARINC 579, VHF Omnidirectional, Radio Range.

IEEE Std 100-1992, The New IEEE Dictionary of Electrical and Electronics Terms (ANSI).³

IEEE Std 416-1984 (W1993), IEEE Standard ATLAS Test Language.⁴

IEEE Std 771-1989, IEEE Guide to the Use of the ATLAS Specification (ANSI).

ISO 1000:1992, SI units and recommendations for the use of their multiples and of certain other units.⁵

ISO/IEC 646:1991, Information technology—ISO coded character set for information interchange.

ITU-T Recommendation G. 702:1988, Digital hierarchy bit rates. Entry No. 8003, AMI Pulse Class, and Entry No. 8005, HDB Pulse Class.⁶

MIL-A-28826, Antenna System, Broadband Identification Friend or Foe, 3 May 1976.⁷

MIL-STD-291B, Standard Tactical Navigation (TACAN) Signal, 13 Dec 67 (change pages dated 12 Dec 1986).

1 ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

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

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

4 IEEE Std 416-1984 has been withdrawn; however, copies can be obtained from Global Engineering, 15 Inverness Way East, Englewood, CO 80112-5704, tel. (303) 792-2181

5 ISO and ISO/IEC publications are available from IEC Sales Department, Case Postale 131, 3, rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse. IEC 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.

6 ITU-T publications are available from the International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse. They are also available in the United States from the U.S. Department of Commerce, Technology Administration, National Technical Information Service (NTIS), Springfield, VA 22161, USA.

7 MIL publications are available from the Director, U.S. Navy Publications and Printing Service, Eastern Division, 700 Robbins Avenue, Philadelphia, PA 19111, USA.

2.2 Document precedence

In case of any conflict in the definition of C/ATLAS within IEEE Std 716, the definition follows the precedence in the order listed:

- a) Formal syntax
- b) Rules and descriptions
- c) Syntax diagrams
- d) Notes and examples

2.3 Document organization and conventions

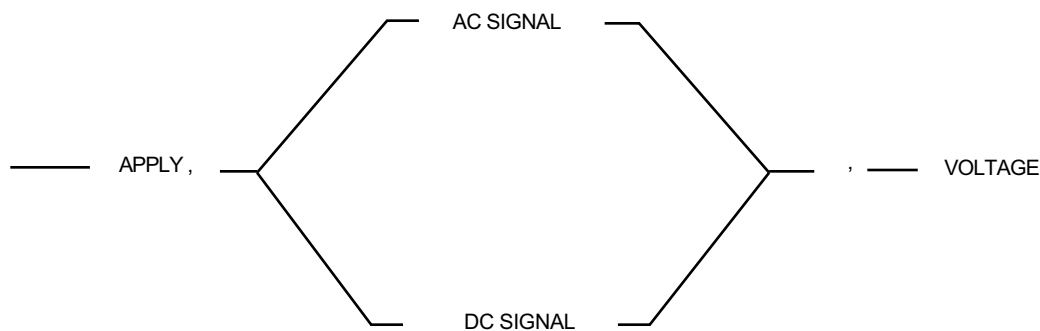
2.3.1 Extensibility

The use of the extend capability for adding nouns, noun modifiers, and pin descriptors shall be in accordance with the dictates of the management agency responsible for procurement of C/ATLAS capability, which incorporates the extensibility mechanism. This is applicable to test requirements which are peculiar to a restricted test discipline.

2.3.2 Organization of syntax specification

The syntax of C/ATLAS is described in syntax diagrams with associated rules and in formal syntax specifications. The syntax diagrams are organized for human readability as described below. The formal syntax is organized for machine analysis use in C/ATLAS compilers and syntax checkers. The formal syntax is fully described in clause 18. Names in subsections entitled "Formal syntax" refer to corresponding names in clause 18.

Syntax diagrams consist of C/ATLAS elements separated by flow lines. Statements are constructed by selecting a path of flow lines from the left of the syntax diagram to the right side and writing in sequence each C/ATLAS element which appears in that path. For example:



would have the following valid sequences:

APPLY, AC SIGNAL, VOLTAGE
APPLY, DC SIGNAL, VOLTAGE

All syntax diagram flow lines shall be traversed from left to right unless an arrowhead indicates differently. Such arrowheads are commonly used to allow repetition of an element as in the

following syntax diagram:



If 'HI-VAL', 'LO-VAL', and 'AVRG-VAL' are names that satisfy the variable <label>, then the following are valid sequences:

'HI-VAL '
'HI-VAL ' , 'LO-VAL '
'HI-VAL ' , 'LO-VAL ' , 'AVRG-VAL '

Three types of C/ATLAS elements included in C/ATLAS syntax diagrams.

- 1) Literals. Uppercase and special characters that are included in ATLAS sequences exactly as they appear.
- 2) Components. Lowercase letters enclosed by the characters "<" and ">" that are replaced by valid symbols and/or numbers as described in clause 15.
- 3) Subdiagrams. Broken bordered boxes containing a component-like name and the clause/subclause number in which the subdiagram is specified. The box is replaced by sequences valid according to that specified paragraph.

Subdiagrams being defined by a syntax diagram consist of a solid bordered box to the left of the defining syntax diagram and contain a component-like name and the subclause number in which the definition appears.

2.3.3 Guide to the use of the C/ATLAS language

For guidance on the proper use of C/ATLAS vocabulary, syntax, and structure for the generation of unambiguous test procedures, refer to IEEE Std 771-1989.

3.0 Complete C/ATLAS test program

3.1 <complete atlas test program structure>

A <complete atlas test program structure> (see figure 3-1) consists of an <atlas program structure> that provides control of the overall execution sequence. It may contain one or more <atlas module structure>s that are independently compiled and which provide additional procedures that are invoked at the appropriate points in the test process.

Execution of a <complete atlas test program structure> begins at an entry point (see 15.3.2) and proceeds sequentially by statement except where the sequence is modified by control structure statements.

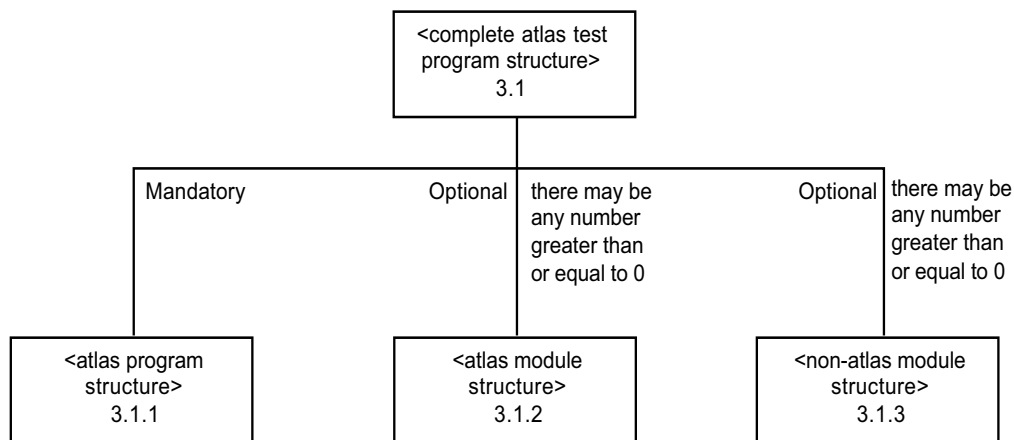


Figure 3-1. Complete C/ATLAS test program family tree

3.1.1 <atlas program structure>

An <atlas program structure> consists of four distinct elements: the <begin atlas program statement> which delimits the start of the structure, a <program preamble structure>, the <main procedural structure>, and the <terminate atlas program statement> which delimits the end of the structure.

The <program preamble structure> precedes the <main procedural structure>. Program preamble statements do not cause any tests to be executed. Information contained in program preamble statements may be referenced later in the <program preamble structure>, the <main procedural structure>, an <atlas module structure>, or a <non-atlas module structure> which have been associated with the <atlas program structure> by <include statement>s.

The <main procedural structure> is where the execution of the test procedure starts.

Figure 3-2 illustrates the structure of a C/ATLAS program and identifies the subclause where each structure or statement is located in this standard.

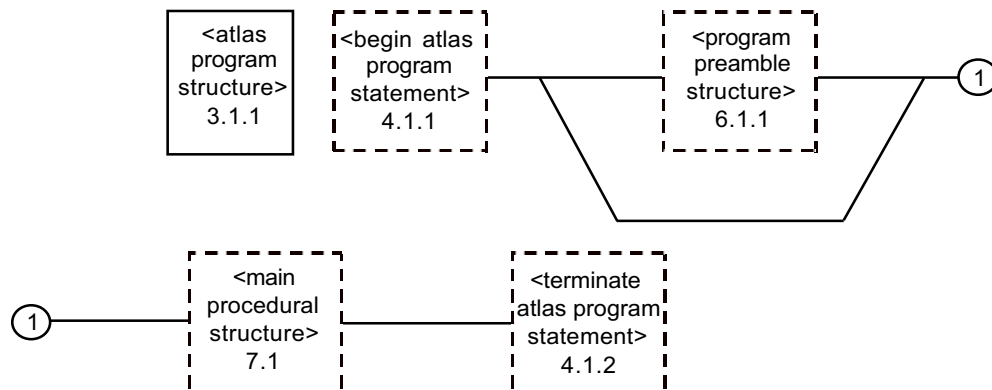


Figure 3-2. C/ATLAS program structure

3.1.2 <atlas module structure>

An <atlas module structure> consists of three distinct elements: the <begin atlas module statement> which delimits the start of the structure, the <module preamble structure>, and the <terminate atlas module statement> which delimits the end of the structure.

Statements within a <module preamble structure> do not cause any tests to be executed. Information contained in module preamble statements may be referenced later in the same <module preamble structure>, in the <atlas program structure>, in another <atlas module structure>, or in a <non-atlas module structure> when they have been associated by <include statement>s. The <atlas module structure> is referenced in the <program preamble structure> via an INCLUDE statement.

Figure 3-3 illustrates the structure of a C/ATLAS module and identifies the subclause where each structure or statement is located in this standard.

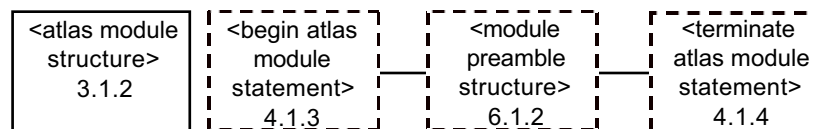


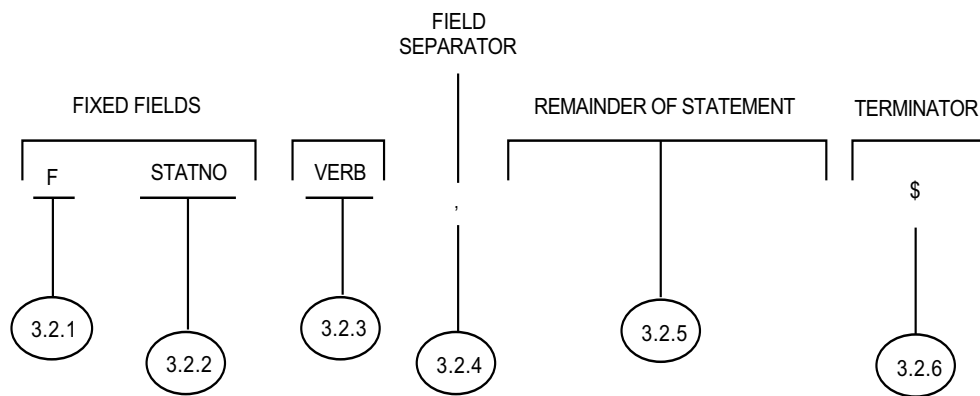
Figure 3-3. C/ATLAS module structure

3.1.3 <non-atlas module structure>

A <non-atlas module structure> is similar to an <atlas module structure> except that it is written in a language other than C/ATLAS. The <non-atlas module structure> is an independent entity and its implementation is dependent upon the run-time system. Information contained in a <non-atlas module structure> may be referenced in the <atlas program structure>, in an <atlas module structure> or another <non-atlas module structure> when they have been associated by <include statement>s.

3.2 Basic statement elements

The structure of a general C/ATLAS statement is shown in figure 3-4. The detailed format for each verb type is specified in the appropriate clause of this document. There is no limit to the length of a C/ATLAS statement. It is not necessary to complete the statement in one line. By inserting spaces between statement elements (as allowed by 15.4), the writer can begin a new line at any point 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 two fields (flag and statement number) shall be written on the first line of the statement.



NOTE—Numbers in circles refer to subclauses in this specification.

Figure 3-4. General C/ATLAS statement structure

3.2.1 Flag field

The flag field can be used to associate special meaning to a C/ATLAS statement or to designate the beginning of a line of text which is commentary (see 15.3.2).

3.2.2 Statement number field

The statement number field is used to uniquely designate C/ATLAS statements (see 15.3.3).

3.2.3 Verb field

The verb field identifies the action for each type of statement. The C/ATLAS verbs are reserved words in that they have unique and special meanings in the C/ATLAS language and shall not be used as variable names.

3.2.4 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 (see 15.4).

3.2.5 Remainder of statement

This part of each particular type of C/ATLAS statement varies considerably. The detailed format following each verb-type is specified in the appropriate clause/subclause of this document.

3.2.6 Statement terminator (\$)

The last character of every C/ATLAS statement shall be the currency symbol (\$).

4.0 Structure delimiter statements

4.1 BEGIN/TERMINATE statements

The BEGIN and TERMINATE statements indicate the first and last statement of an <atlas program structure> or <atlas module structure>.

4.1.1 BEGIN, ATLAS PROGRAM statement

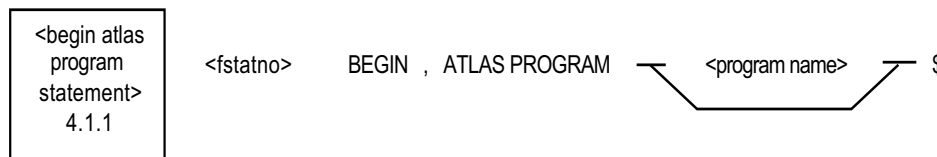
4.1.1.1 Function

To indicate the start of the text of an <atlas program structure>.

4.1.1.2 Formal syntax

Reference begin–atlas–program–statement

4.1.1.3 Syntax diagram



4.1.1.4 Rules

1. The first statement of every <atlas program structure> is a <begin atlas program statement>. The statement can be used only once within a single <atlas program structure>.
2. The optional <program name> is used to identify the <atlas program structure>.

4.1.1.5 Notes and examples

```
110000 BEGIN, ATLAS PROGRAM 'SERVO-AMPLIFIER' $
```

4.1.2 TERMINATE, ATLAS PROGRAM statement

4.1.2.1 Function

To indicate the end of the text of an <atlas program structure>.

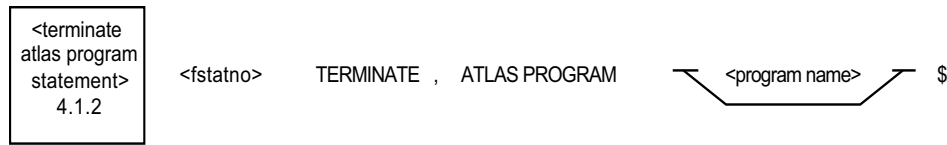
4.1.2.2 Formal syntax

Reference terminate–atlas–program–statement

4.1.2.3

Syntax

diagram



4.1.2.4 Rules

1. The last statement in every <atlas program structure> is a <terminate atlas program statement>. The statement can be used only once within a single <atlas program structure>.
2. The optional <program name> shall be identical to the <program name> in the <begin atlas program statement>.

4.1.2.5 Notes and examples

```
999999 TERMINATE, ATLAS PROGRAM 'SERVO-AMPLIFIER' $
```

4.1.3 BEGIN, ATLAS MODULE statement

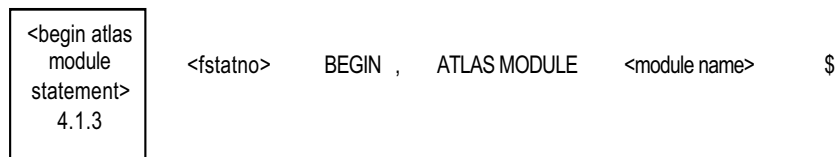
4.1.3.1 Function

To indicate the start of the text of an <atlas module structure>.

4.1.3.2 Formal syntax

Reference begin-atlas-module-statement

4.1.3.3 Syntax diagram



4.1.3.4 Rules

1. The first statement of every <atlas module structure> is a <begin atlas module statement>. The statement can be used only once within a single <atlas module structure>.
2. The <module name> is used to identify the <atlas module structure>, and is referenced by an INCLUDE statement within the <program preamble structure>.
3. The <module name> shall be unique within one <complete atlas test program structure>.

4.1.3.5 Notes and examples

```
120000 BEGIN, ATLAS MODULE 'CALIBRATION PROC' $
```

4.1.4 TERMINATE, ATLAS MODULE statement

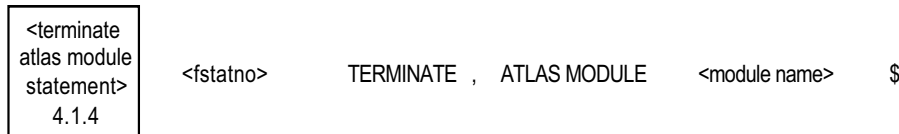
4.1.4.1 Function

To indicate the end of the text of an <atlas module structure>.

4.1.4.2 Formal syntax

Reference terminate–atlas–module–statement

4.1.4.3 Syntax diagram



4.1.4.4 Rules

1. The last statement in every <atlas module structure> is a <terminate atlas module statement>. The statement can be used only once within a single <atlas module structure>.
2. The <module name> shall be identical with the <module name> in the <begin atlas module statement> of the same <atlas module structure>.

5.0 This clause is reserved for future use.

6.0 Preamble statements

An <atlas program structure> may contain a single <program preamble structure>, and each <define procedure structure> may contain a single <local preamble structure>. In each case the preamble serves to identify the names and attributes of each <label> which is subsequently referenced within the appropriate structure.

Within the atlas preamble structures, labels may be established for use in the <atlas program structure>, an <atlas module structure> or a <non-atlas module structure>. These labels are generally classified as follows:

1. Global. A label is global in scope if it has been established with the GLOBAL attribute in one of the structures mentioned above and with the EXTERNAL attribute in the others in which it is to be referenced.
2. Local. A label is local in scope if it is established with neither the GLOBAL nor EXTERNAL attribute. It may then only be accessed within the structure in which it was established.

A full description of global and local attributes is located in 15.6.1.

6.1 Main preamble structure

6.1.1 <program preamble structure>

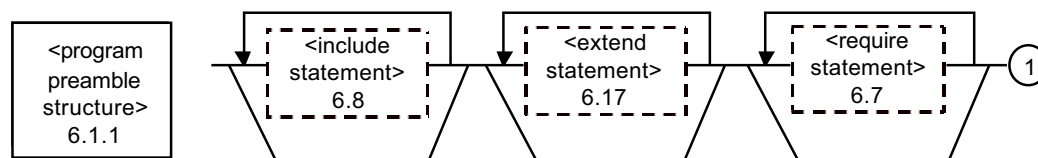
6.1.1.1 Function

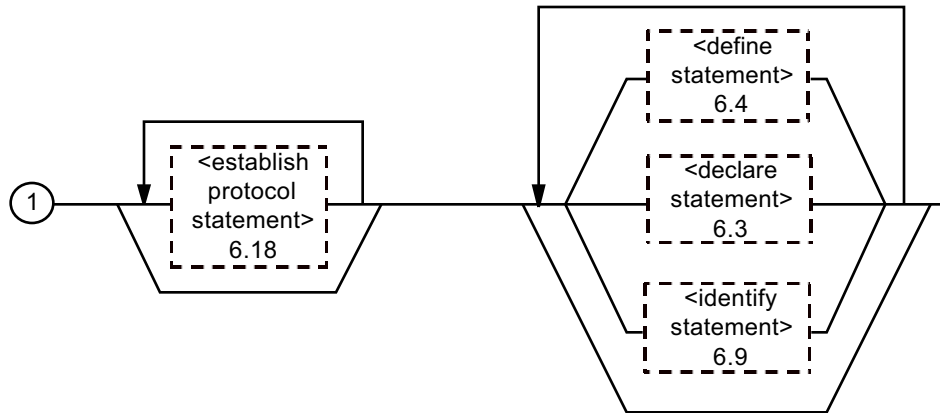
The <program preamble structure>, which appears immediately following the <begin atlas program statement>, establishes the name and attributes for labels whose scope includes the whole of the <atlas program structure>.

6.1.1.2 Formal syntax

Reference program-preamble-structure

6.1.1.3 Syntax diagram





6.1.1.4 Rules

1. There may be only one <program preamble structure> within an entire <atlas program structure>.
2. The <program preamble structure> shall appear immediately following the BEGIN, ATLAS PROGRAM statement.
3. Any <label> identified within the <program preamble structure> can be referenced within the entire <atlas program structure> unless an identical <label> is established at a lower level structure within a <local preamble structure>.

6.1.2 <module preamble structure>

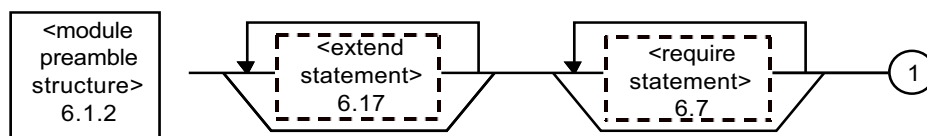
6.1.2.1 Function

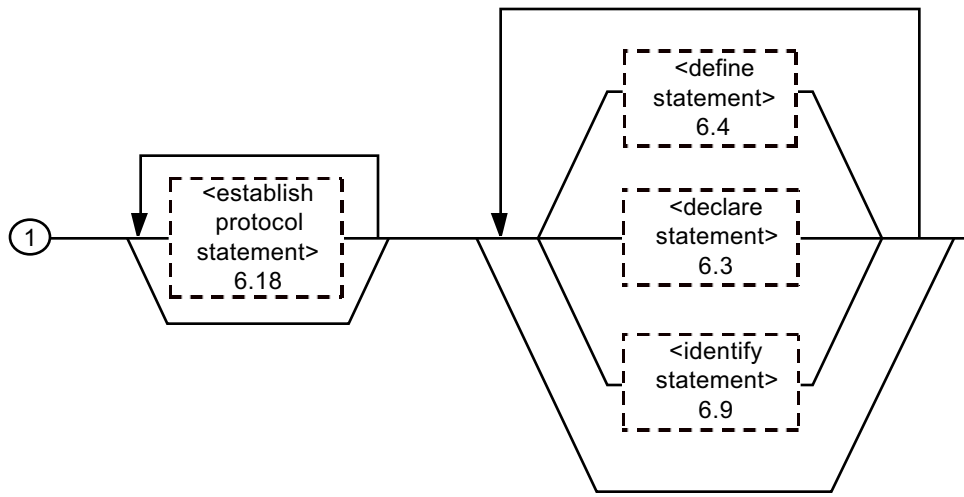
The <module preamble structure> that appears immediately following the BEGIN, ATLAS MODULE statement establishes the name and attributes for labels whose scope includes the whole of the <atlas module structure>.

6.1.2.2 Formal syntax

Reference module-preamble-structure

6.1.2.3 Syntax diagram





6.1.2.4 Rules

1. There may be only one <module preamble structure> within an entire <atlas module structure>.
2. Any <label> identified within the <module preamble structure> can be referenced within the entire <atlas module structure> unless an identical <label> is established at a lower level structure within a <local preamble structure>. Each label identified as being EXTERNAL shall have a corresponding statement in either the <program preamble structure> which identifies it, or another <module preamble structure> which identifies it as being GLOBAL. All references to the label shall be to the same object, except where an identical label is established at a lower structure within a <local preamble structure>.

6.2 <local preamble structure>

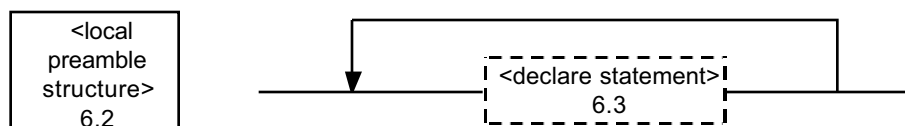
6.2.1 Function

A <local preamble structure> identifies the name and attributes of each <label> that is local to the <define procedure structure>.

6.2.2 Formal syntax

Reference local-preamble-structure

6.2.3 Syntax diagram



6.2.4 Rules

1. There may only be one <local preamble structure> within each <define procedure structure>.
2. The <local preamble structure> shall appear immediately following a DEFINE PROCEDURE statement within the appropriate structure.
3. A <local preamble structure> only contains DECLARE statements.
4. A <declare statement> within a <local preamble structure> is always local and consequently may not specify either a GLOBAL or EXTERNAL attribute.

6.3 DECLARE statement

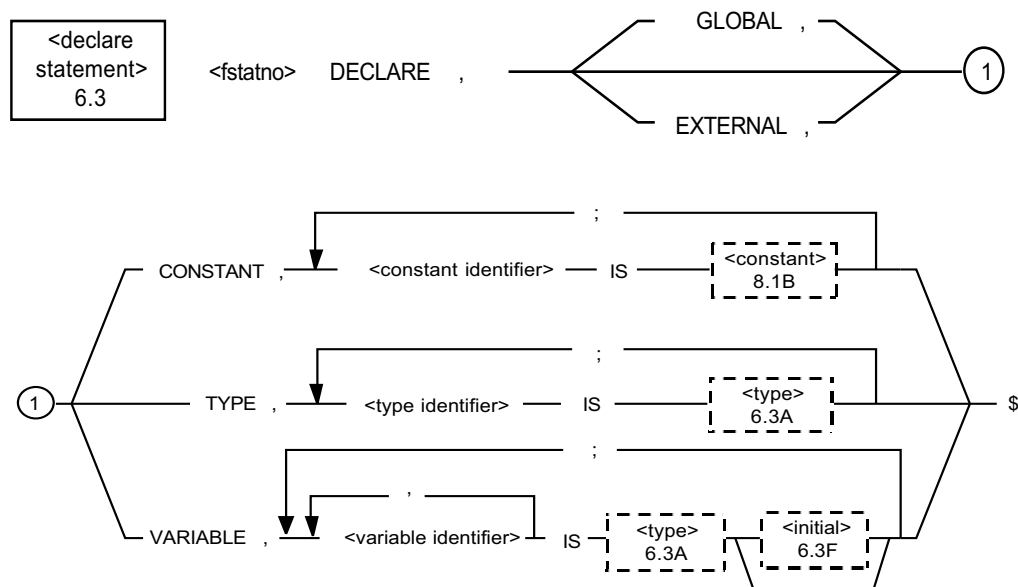
6.3.1 Function

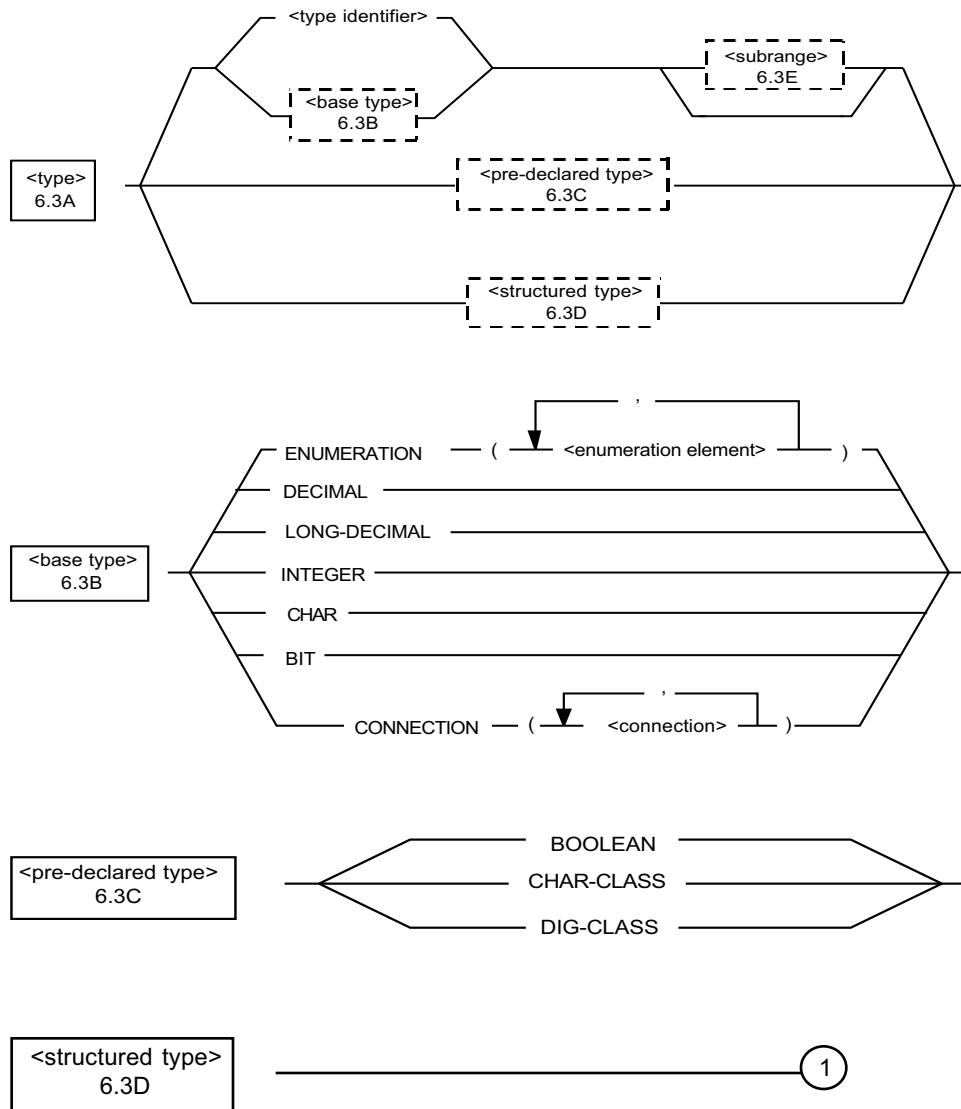
To establish, label, and identify data types, variables, and constants.

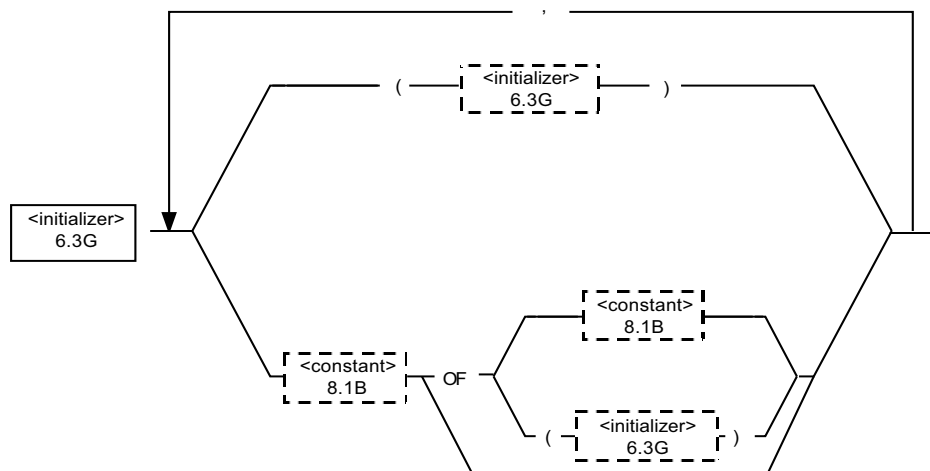
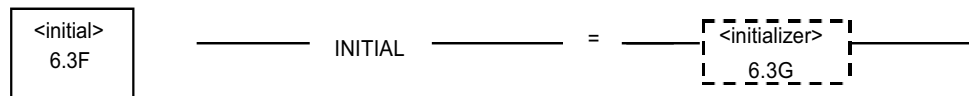
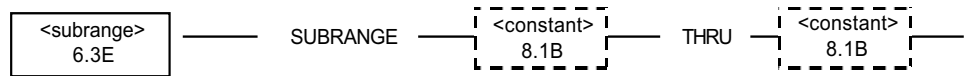
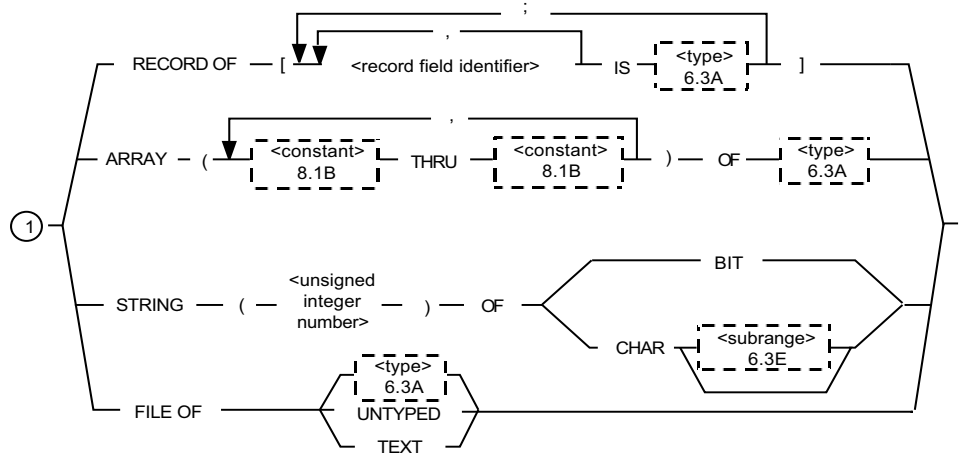
6.3.2 Formal syntax

Reference declare-statement

6.3.3 Syntax diagram







6.3.4 Rules

6.3.4.1 Rules for CONSTANT, TYPE and VARIABLE forms of <declare statement>

1. The CONSTANT, TYPE, or VARIABLE <label> created is accessible at the structural level where it is declared and at any inner nested structural level.
2. Repetition via the semicolon, ";", allows for the creation of multiple CONSTANT, TYPE, or VARIABLE <label>s in one DECLARE statement without keyword repetition.
3. References to CONSTANT, TYPE, and VARIABLE <label>s can only be made after they have been properly declared in a DECLARE statement.
4. A <label> that is declared as GLOBAL shall be unique within the <complete atlas test program structure>. A GLOBAL <label> declared in an <atlas program structure> may be accessed in an <atlas module structure> if that <atlas module structure> contains a DECLARE for that <label> as EXTERNAL. A GLOBAL <label> declared in an <atlas module structure> may be accessed in another <atlas module structure> or in the <atlas program structure> if that <label> is declared as EXTERNAL in those structures.
5. The attributes GLOBAL and EXTERNAL are not allowed in a <local preamble structure>.
6. Within any main preamble structure or any <local preamble structure> all declared labels shall be unique. However, <record field identifier> labels in different record types may be the same. Where nested preambles use the same labels, the innermost sense shall prevail following the redefinition until the structure in which it is redefined is exited.
7. If a <label> is declared as GLOBAL in one structure and as EXTERNAL in another then the two shall be of the same <type> and:
 - a) If they are ARRAYS, the corresponding dimensions shall have the same number of components with the same index number.
 - b) If they are STRINGS, they shall have the same number of components.
8. INITIAL cannot be used to initialize an EXTERNAL variable.

6.3.4.2 Rules for CONSTANT form of <declare statement>

The type of the <constant identifier> label created is determined by and is the same as that of the constant. This type depends on the form of the constant in accordance with 8.1.4.5.

6.3.4.3 Rules for TYPE form of <declare statement>

TYPE declarations are used to avoid repetition of identical descriptions of data structures in VARIABLE declarations.

6.3.4.4 Rules for VARIABLE form of <declare statement>

1. A VARIABLE declaration creates storage space for that variable according to its <type>. The <type> is expressed either explicitly in the VARIABLE declaration or by reference to a previously declared <type identifier>. The VARIABLE will be initialized if <initial> is invoked.
2. Any number of VARIABLES of the same <type> can be created using the path containing the comma.
3. VARIABLES declared in a <local preamble structure> cannot be relied on to retain their values between invocations of the structure containing that <local preamble structure>.
4. Initialization of VARIABLES is discussed in 6.3.4.10.

6.3.4.5 Rules for <type>

1. A DECLARE, TYPE statement establishes a <type identifier> label which is associated with the described <type>.
2. When used in a DECLARE, VARIABLE statement or in a DEFINE, PROCEDURE statement within a <parameter> subfield, the description of the <type> can be stated explicitly without reference to a <type identifier> label.
3. A value is assignment compatible with a <data store> provided that one of the following conditions is true:
 - a) They are not <structured type>s, are of the same <base type>, and if the <data store> has been declared with a sub-range, then the value lies within that sub-range.
 - b) Their <structured type> is STRING OF BIT and the current length of the value is less than or equal to the declared length of the <data store>.
 - c) Their <structured type> is STRING OF CHAR, the current length of the value is less than or equal to the declared length of the <data store>, and if the <data store> has been declared with a sub-range of CHAR, then the value of each character lies within that sub-range.
 - d) Their <structured type> is ARRAY, they have the same dimensions with the same number of components, and each component is assignment compatible with the corresponding component of the <data store>.
 - e) Their <structured type> is RECORD and their corresponding fields are assignment compatible.
 - f) The <data store> is of <base type> LONG-DECIMAL, the value is of <base type> DECIMAL or INTEGER, and if the <data store> has been declared with a sub-range, then the value, when coerced to type LONG-DECIMAL, lies within that sub-range.
 - g) The <data store> is of <base type> DECIMAL, the value is of <base type> INTEGER, and if the <data store> has been declared with a sub-range, then the value, when coerced to type DECIMAL, lies within that sub-range.
 - h) The <data store> is of <structured type> STRING OF CHAR, the value is of <base type> CHAR, and if the <data store> has been declared with a sub-range, then the value of the CHAR lies within that sub-range. Assignments according to this rule set the dynamic length of the STRING OF CHAR to one.

- i) The <data store> is of <structured type> STRING OF BIT and the value is of <base type> BIT. Assignments according to this rule set the dynamic length of the STRING OF BIT to one.

6.3.4.6 Rules for <base type>

1. A variable of <base type> shall assume a value whose type conforms to the declared <base type>. Thus, for example, a <variable identifier> of type DECIMAL may only assume a <decimal number>.
2. The <enumeration element> labels of a particular declared ENUMERATION type are <constant>s ordered by the sequence in which they are written into the declaration. They have the same format as other labels, including the requirement for enclosing them in apostrophes.
3. The <enumeration element> labels shall be unique within the <atlas module structure> or <atlas program structure> in which they are defined.
4. A CONNECTION variable identifies a UUT <connection> as defined in 14.12 <conn> (Connection Field).
5. A CONNECTION variable can only assume values taken from the given <connection> list. The <connection> elements in all CONNECTION type declarations shall be unique within the <atlas module structure> or <atlas program structure> in which they are defined.
6. LONG-DECIMAL <type> provides for an increased number of significant digits in comparison with DECIMAL <type> in order to gain high accuracy. See 15.2, item g.

6.3.4.7 Rules for <pre-declared type>

1. The <pre-declared type>s are commonly used ENUMERATIONS. The <pre-declared type>s exist as though each C/ATLAS program contained the following declaration:

```
DECLARE, GLOBAL, TYPE,
  'BOOLEAN' IS ENUMERATION ('FALSE', 'TRUE');
  'DIG-CLASS' IS ENUMERATION ('BNR', 'B1C', 'B2C', 'BSM', 'BCD', 'SBCD');
  'CHAR-CLASS' IS ENUMERATION ('ASCII7', 'ISO7') $
```

2. The <type identifier>s and <enumeration element>s that are enclosed by apostrophes in the above declaration defining <pre-declared type>s are written without apostrophes in an <atlas program structure> or <atlas module structure> in accordance with 15.5.
3. DIG-CLASS variables have values that are associated with digital data formats as follows:

BNR for unsigned positive binary numbers (no sign bit)

B1C for ones-complement binary numbers

B2C for twos-complement binary numbers

BSM for sign-plus-magnitude binary numbers

BCD for binary-coded-decimal numbers (4 bits per digit)

SBCD for sign-plus-magnitude BCD numbers
(4 bits per digit plus a sign bit)

BCD and SBCD numbers have an 8-4-2-1 coding.

Signed numbers have a sign bit in the most-significant-bit position with a binary zero representing a positive sign.

Other types of coding and sign conventions can be represented by BNR, with the value of each bit of a number being translated into and out of C/ATLAS number types by conversion routines defined by a <define procedure structure>.

4. CHAR-CLASS variables have values that are associated with character codes as follows:

ASCII7 for ASCII characters transmitted across the UUT/test equipment interface (7 bits per character) (see 15.9.1).

ISO7 for ISO characters transmitted across the UUT/test equipment interface (7 bits per character) (see 15.9.2).

6.3.4.7.1 Notes and examples

1. Values of DIG-CLASS are arguments to the <pre-defined function>s DIG and DEC.
2. Values of CHAR-CLASS are arguments to the <pre-defined function>s CHAR and BIT.

6.3.4.8 Rules for <structured type>

1. An ARRAY is a collection of data items of the same <type>. Any <type> except FILE may be specified. An ARRAY may have any number of dimensions. Each dimension has an associated index that shall be bounded by upper and lower limits. A multi-dimensional ARRAY declaration is structurally identical with, and is shorthand for a declaration of an ARRAY of an ARRAY. For example:

```
ARRAY (1 THRU 7, 1 THRU 9)
```

is shorthand for:

```
ARRAY (1 THRU 7) OF ARRAY (1 THRU 9)
```

The <constant>s that specify the boundary limits of a dimension of an ARRAY shall be of the same <type>. The <type> shall be either INTEGER, CHAR, or <enumeration element> of the same ENUMERATION type. The lower boundary limit is written first. Any element in an ARRAY can be addressed by its indices.

2. Array elements are assigned in row major order, such that the last written index varies most rapidly and the first written index varies more slowly in the conventional manner.
3. Operations between entire arrays require that the arrays involved are assignment compatible and have an equal number of elements. The only operations permitted between entire arrays are assignment and test for equality and inequality of all corresponding elements in both arrays.
4. A RECORD is a collection of data items that need not be of the same <type> or structure. Any type except FILE may be specified. Individual fields within a RECORD are given field names. Items of a RECORD are addressed by appending a period followed immediately by the <record field identifier> label to the RECORD label. The declaration of fields of records is the same as for variables and the rules of 6.3.4.4 apply.
5. The only operations permitted between entire records are assignment and test for equality and inequality of all corresponding fields in both records. Records shall be assignment compatible and have equal numbers of elements.
6. The constant specifying the length of a string shall be greater than zero and of <base type> INTEGER.
7. The <base type> of a STRING shall be either BIT or CHAR, or a subrange of CHAR.
8. A STRING has an associated dynamic length attribute that holds the current length of the STRING.
 - a) Initialization, input, parameter passing, and assignment (whether explicit as in CALCULATE or implicit as in such verbs as SENSE) set the dynamic length.
 - b) References to an individual BIT or CHAR of a STRING are allowed within the current dynamic length.
 - c) Operations between STRINGs of the same <base type> and different declared sizes are permitted.
 - d) It is an error if an operation on a STRING would cause the current dynamic length to exceed the declared length.
9. Conversion of a STRING OF BIT to and from a numerical value is accomplished by using the DEC and DIG functions respectively, in accordance with the DIG-CLASS parameter.
10. Conversion of a STRING OF BIT to and from a character is accomplished by using the CHAR and BIT functions respectively, in accordance with the CHAR-CLASS parameter.
11. Within RECORDs, any number of fields of the same <type> can be created by using the path with a comma.
12. Within RECORDs, use of the ";" repetition operator allows for the creation of any number of fields, which may be of different types.
13. A FILE is a collection of data items that may not contain another FILE. Each data item has a position in the FILE beginning with zero and advancing by one with each data item. Three classes of FILE are specified:
 - a) A known type file consists of a sequence of data items all of which have the same type.

Input and output shall be to and from a variable of identical type.

- b) An untyped file consists of bytes of data that may be representative of anything. Input and output for this type of file requires a detailed knowledge of the data format, which may be very machine dependent. For graphic applications, a graphic file containing all the needed graphic data should be declared as an untyped file.
- c) A text file consists of a stream of characters that equate to declared data types, after possible automatic I/O conversion.

6.3.4.9 Rules for <subrange>

1. A <subrange> is used to restrict the range of the values that may be assigned to a variable to lie within a sub-division of the range of values that may be assigned to its <base type>.
2. The type of <constant> shall conform to the <base type>.
3. The value of the <constant>s shall lie within the bounds of the <type> being ranged. The value of the first constant shall be less than or equal to the value of the second constant.
4. The <type> being ranged shall be ENUMERATION, DECIMAL, LONG-DECIMAL, INTEGER, or CHAR and shall not be any structure.
5. A <subrange> variable is of the same type as a variable of the associated <base type>.
6. Initialization, input, assignment, and parameter passing to subrange variables shall lie within the subrange bounds.

6.3.4.10 Rules for <initial>

1. The action of an <initial> field is to set the initial value of variables. Initialization of variables always takes place in the order in which the variables are declared.
 - a) For declarations in a <program preamble structure> or <module preamble structure> and not in a <local preamble structure>, initialization takes place prior to the execution of the first executable statement of the <main procedural structure>.
 - b) For declarations in a <local preamble structure>, initialization takes place each time the procedure is invoked prior to the execution of the first executable statement following the <local preamble structure> that contains the declaration.
2. Variables cannot be initialized if:
 - a) They are of type FILE.
 - b) They are EXTERNAL.
3. INITIAL can be used to initialize a single variable or a series of variables. The <initializer> following INITIAL is a comma delimited list of values having a one-to-one correspondence with the variables. Each value may be one of the following:
 - a) A <constant> that corresponds to a simple variable.
 - b) A <constant> number of <constant>s where the first <constant> shall evaluate to a positive integer that specifies the number of instances of the second <constant>. Each instance of the second <constant> corresponds to a simple variable.
 - c) A parenthesized list of values having a one-to-one correspondence with the elements of

an ARRAY or <record field identifier>s of a RECORD.

- d) A shorthand notation for a repeated occurrence of a parenthesized list of values defined by item c). This is in the form of "<constant> OF (list of values)", where <constant> shall be a positive integer that specifies the number of times the parenthesized list is to be repeated (e.g., "2 OF (0,1)" is equivalent to "(0,1), (0,1)").

4. The value of the <constant> supplied by the <initializer> shall be assignment compatible with the corresponding variable.

5. For ARRAYs, initial values are assigned starting with the lowest value of all indices and varying the last written index most rapidly.

6.3.5 Notes and examples

1. This example shows the multiple <variable identifier> use of DECLARE. In this example, three locations are set aside for pin designations.

```
DECLARE, VARIABLE,
    'PIN-1', 'PIN-2', 'PIN-3' IS
    CONNECTION (J1, J2, J3, J4) $
```

2. This example is used to illustrate how storage is set aside for a <variable identifier> of type STRING OF CHAR. The length of the character string stored in this location is equal to or less than 36 characters.

```
DECLARE, VARIABLE,
    'MESSAGE' IS STRING (36) OF CHAR $
```

3. This example illustrates how three storage locations, each 10 bits wide, are specified. The format of the data stored in these locations can be controlled by using the <pre-defined function> DIG for digital numbers in which a bit weight is to be associated with each bit position.

```
DECLARE, VARIABLE,
    'STIM', 'MASK', 'RESP' IS STRING (10) OF BIT $
```

4. This example is a use of <record field identifier> to specify storage for 'PATTERN' consisting of 22 bits composed of three individual parts 'PART1', 'PART2', and 'PART3' consisting of 4, 8, and 10 bits respectively.

```
DECLARE, VARIABLE,
    'PATTERN' IS
    RECORD OF ['PART1' IS STRING (4) OF BIT;
               'PART2' IS STRING (8) OF BIT;
               'PART3' IS STRING (10) OF BIT] $
```

5. This example shows how three one dimensional arrays, each containing five elements, are declared.

```
DECLARE, VARIABLE,
    'INDEX', 'WEIGHT FACTOR', 'COUNTER'
    IS ARRAY (1 THRU 5) OF INTEGER $
```

6. This example shows how storage may be established for an array of connection variables.

```
DECLARE, VARIABLE,
  'PINSET1', 'PINSET2' IS ARRAY (1 THRU 4) OF
    CONNECTION (J1, J3, J5, J7, J9, J11, J13, J15, J17) $
```

7. This example establishes storage for five bit-strings each of which may be up to 20 bits in length.

```
DECLARE, VARIABLE,
  'COMMAND' IS
    ARRAY (1 THRU 5) OF STRING (20) OF BIT $
```

8. This example illustrates how several structures may be declared and initialized within a single DECLARE statement.

```
DECLARE, VARIABLE,
  'SINGLE' IS DECIMAL INITIAL = 1.234;
  'LEFT', 'RIGHT', 'CENTER' IS
    BOOLEAN INITIAL = 3 OF TRUE;
  'PATTERN' IS ARRAY (1 THRU 1200)
    OF STRING (16) OF BIT
    INITIAL = (1200 OF X'0000');
  'WEEK-DAY' IS RECORD OF
    ['SUN', 'MON', 'TUE', 'WED',
     'THURS', 'FRI', 'SAT' IS BOOLEAN]
    INITIAL = (FALSE, 5 OF TRUE, FALSE) $
```

9. This example illustrates the declaration of connection data elements.

```
DECLARE, GLOBAL, VARIABLE,
  'PINSET1' IS CONNECTION (PIN1, PIN2, PIN3);
  'PINSET2' IS CONNECTION (PIN4, PIN5, PIN6);
  'PINSET3' IS CONNECTION (PIN7, PIN8, PIN9) $
```

10. This example illustrates the declaration and initialization of enumerated data elements and the record structure.

```
DECLARE, VARIABLE,
  'CHESS-BOARD' IS ARRAY (1 THRU 8, C'A' THRU C'H')
    OF ENUMERATION ('EMPTY', 'BLACK', 'WHITE')
    INITIAL = ((2 OF (8 OF 'BLACK')),
               (4 OF (8 OF 'EMPTY')),
               (2 OF (8 OF 'WHITE')));
  'FAILURES' IS RECORD OF
    ['COUNT' IS INTEGER;
     'FAIL-MESS' IS
       ARRAY (1 THRU 2) OF STRING (20) OF CHAR]
    INITIAL = (0, (C'OVER PRESSURE',
                  C'OVER VOLTAGE')) $
```

11. This example shows the initialization of an array composed of records.

```
DECLARE, VARIABLE,
  'DATES' IS ARRAY (1 THRU 100) OF RECORD OF
  ['DAY', 'MONTH', 'YEAR' IS INTEGER]
  INITIAL = (100 OF (0,0,0)) $
```

12. This example illustrates the concept of most significant bit (MSB) and least significant bit (LSB) for digital data patterns:

```
DECLARE, VARIABLE,
  'DATA PATTERNS' IS ARRAY (1 THRU 1000)
  OF STRING (32) OF BIT $

DECLARE, VARIABLE,
  'MSB', 'LSB' IS STRING (32) OF BIT
  INITIAL = (X'80000000', X'00000001') $
```

6.4 DEFINE statements definition

6.4.1 Function

The function of DEFINE statements is to establish and label a portion of the <complete atlas test program structure> that is not executed until the label is invoked by direct or indirect reference in a subsequent <main procedural statement>.

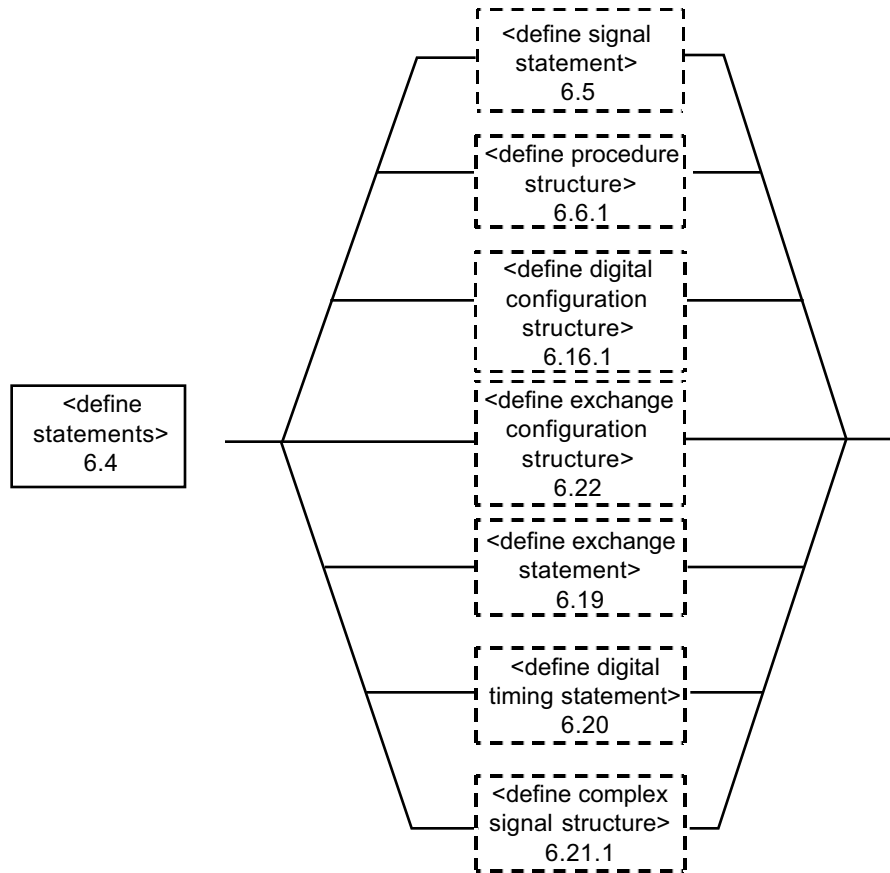
DEFINE statements establish and identify all or part of one statement or a series of statements.

This capability can be employed to avoid rewriting often-used C/ATLAS segments by simply writing the <label> in the appropriate place in a subsequent DEFINE or procedural statement.

6.4.2 Formal syntax

Reference define-statements

6.4.3 Syntax diagram



6.4.4 Rules applicable to all DEFINE statements

1. DEFINE statements are not allowed in a <local preamble structure>.
2. A DEFINE statement may only refer to previous DEFINE statements by the appropriate <label>.

6.5 DEFINE <signal> statement

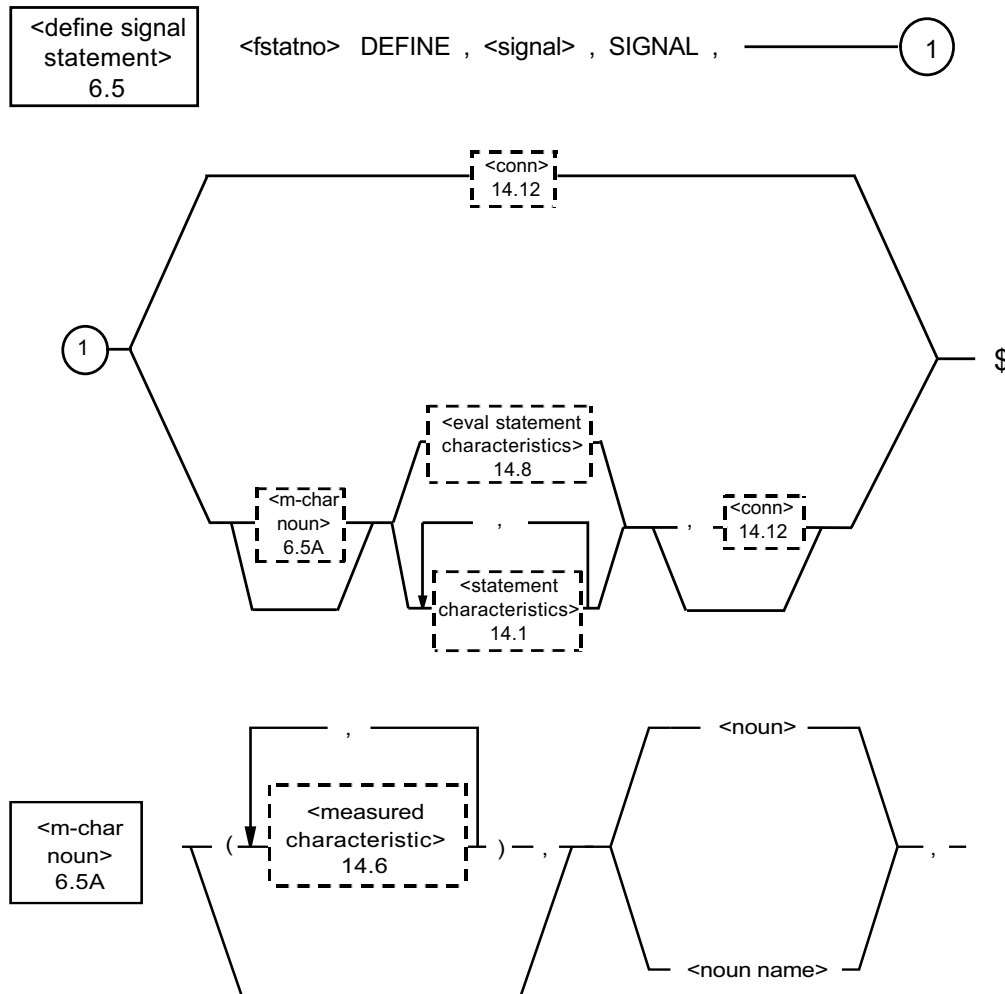
6.5.1 Function

The <define signal statement> assigns a <label> to one or more adjacent fields of a C/ATLAS signal-oriented statement. When the <signal> label is written in a subsequent statement, a strict character substitution takes place to form a complete statement. Quantities in the <define signal statement> may be represented by a parentheses pair () to indicate that the quantity is variable and a specific value will be provided when the label is referred to in the subsequent <define statement> or <main procedural statement>. In a <define signal statement> the number of spaces between the parentheses is not important.

6.5.2 Formal syntax

Reference define–signal–statement

6.5.3 Syntax diagram

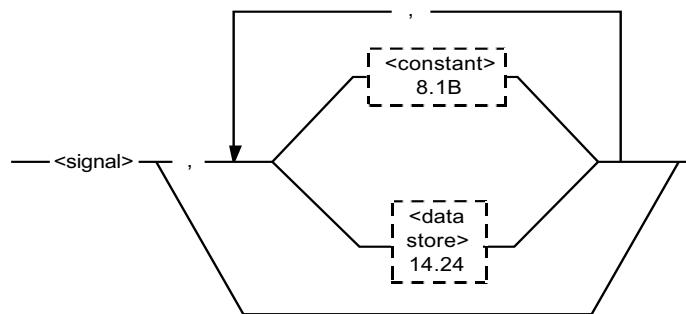


NOTE—A <decimal number>, <digital number>, and <connection> may be designated as variable by replacing the item by a pair of parentheses. <label>s defined in a <define signal statement> may be used either in subsequent DEFINE statements or in procedural statements. The value for any parentheses pairs in the definition shall be stated explicitly whenever the <label> is used. The values are listed in sequence immediately following the <label> and will replace the parentheses pairs in the definition in strict sequence, and then the complete character sequence from the definition will replace the <label> and all variables, if any, following it.

A valid C/ATLAS statement shall result when the substituted information is combined with the other information in the statement. Following the parentheses pairs designating a variable quantity, a range field containing non-variable quantities may be given (is optional).

6.5.4 Rules

1. This type of statement is employed to establish and identify one or more fields that follow the verb of a signal-oriented statement for direct substitution in place of the <label> when the <label> is written in a subsequent statement.
2. Only complete comma-to-comma or comma-to-dollar fields that are adjacent in the completed procedural statement may be included in the same DEFINE signal statement.
3. Connections and the numerical values associated with any <real quantity> in a DEFINE statement may be designated variable by inserting a parentheses pair in place of the number or connection. The dimensions of the variables shall immediately follow the parentheses pairs. Only blank spaces, if desired, may be written between the left and right parentheses. When a characteristic is designated variable in a <define signal statement>, a range field may be included to state the lowest and highest values that will be assigned to that characteristic in subsequent statements.
4. In the statement that references the definition, the <label> is inserted in place of the defined fields and is followed by the value(s) of any parenthetical variables from the DEFINE statement. These values may be either constant or DECLARED variables. Each value is written in the same sequence as defined, and preceded by a comma regardless of the location of commas in the DEFINE statement.
5. If a DEFINE <signal> references the <label> of a previous DEFINE <signal> statement, then any pairs of parentheses in the referenced DEFINE shall be filled explicitly. They may not be filled by "daisy-chaining" through another pair of parentheses in the calling DEFINE statement.
6. A dimension should not be specified as a variable.
7. The DEFINE <signal> statement provides a standard format that may be used to predefine portions of signal-oriented statements. It may also be used to predefine subfields of the REMOVE statement that conform to the standard format and to predefine the connection field in digital statements. The mechanism for using predefined portions of a signal-oriented statement is not included in the syntax diagrams of clause 11 or in the corresponding Formal Syntax Definition.
8. Signals are referenced within individual statements according to the following diagram:



6.5.5 Notes and examples

Each example in this clause is written with a corresponding procedural statement.

```
000011 DEFINE, 'RZ STIM', SIGNAL, AC SIGNAL,
          VOLTAGE ( ) V, FREQ ( ) HZ,
          CNX HI J2-1 LO J2-2 $
111133 APPLY, 'RZ STIM', 110, 400 $
```

Note that the label is inserted in its appropriate place in the procedural statement. Definition 000011 included the noun; so, in step 111133, 'RZ STIM' is written following the verb.

```
000022 DEFINE, 'GRID', SIGNAL,
          VOLTAGE ( ) RANGE 1V TO 10V ERRLMT + - 10 PC,
          CNX HI J2-3 LO J2-4 $
111144 APPLY, DC SIGNAL, AC-COMP MAX 5MV, 'GRID', 'VDC' $
111155 APPLY, AC SIGNAL, DC-OFFSET 20V, 'GRID', 'VDC2' $
```

Step 000022 did not include the noun; so, in steps 111144 and 111155, it is possible to use any noun and provide statement characteristics in addition to VOLTAGE. The only restriction in this regard is that the completed statement, after substitutions, conforms to C/ATLAS format requirements.

```
000033 DEFINE, 'PWR PINS', SIGNAL,
          CNX HI ( ) LO ( ) $
000099 APPLY, DC SIGNAL, VOLTAGE 500V, AC-COMP 5V,
          'PWR PINS', J1-1, J1-4 $
```

This example illustrates how a connection field can be defined by itself. Note that the pins are separated by commas in the APPLY statement because they are values for separate parameters, even though a comma does not appear in the normal pin format.

```
112244 DEFINE, 'MULTIMETER', SIGNAL, AC SIGNAL,
          FREQ MAX 410 HZ, VOLTAGE MAX 15V,
          CURRENT MAX 1 MA,
          CNX HI J1-2 LO J1-3 $
88 MEASURE, (FREQ INTO 'SAVE1'), 'MULTIMETER' $
98 MEASURE, (VOLTAGE INTO 'SAVE2'), 'MULTIMETER' $
99 MEASURE, (CURRENT INTO 'SAVE3'), 'MULTIMETER' $
```

This example illustrates how a sensor signal may be defined. In the procedural section, various <measured characteristic> fields can be supplied to make the signal serve for several measurement operations.

6.6 PROCEDURE definition

6.6.1 <define procedure structure>

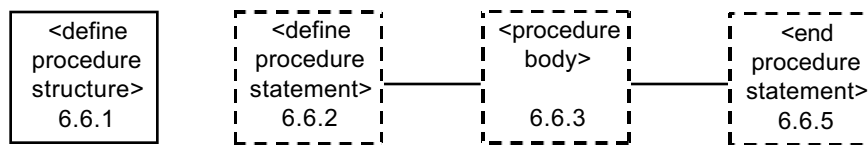
6.6.1.1 Function

The <define procedure structure> identifies one or more complete C/ATLAS statements that may be subsequently invoked by use of the PERFORM statement. The definition consists of three parts: the <define procedure statement>, the <procedure body>, and finally, the <end procedure statement>.

6.6.1.2 Formal syntax

Reference define-procedure-structure

6.6.1.3 Syntax diagram



6.6.1.4 Rules

1. The <define procedure structure> is employed to establish and label one or more complete C/ATLAS statements that are executed in the sequence defined whenever the procedure label is called in a subsequent PERFORM statement.
2. The first statement of the series is a <define procedure statement>. The series of C/ATLAS statements is written next. The last statement of the series is an <end procedure statement>.
3. There is no limit to the number of statements that may be included in a defined procedure.

6.6.1.5 Notes and examples

```

000421  DEFINE, 'NBD', PROCEDURE ... $      define-procedure-statement
        DECLARE, ... $
        .
        .
        APPLY, ... $                        procedure-body
        .
        .
        END, 'NBD' $                        end-procedure-statement
  
```

6.6.2 DEFINE PROCEDURE statement

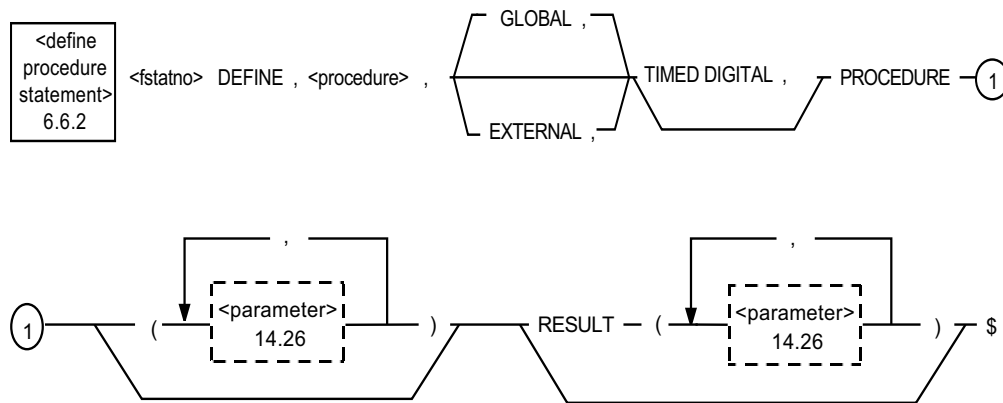
6.6.2.1 Function

The <define procedure statement> identifies the following <procedure body> and assigns labels (known as formal parameters) to data items, if any, which are referenced within the following <procedure body>. The formal parameters are equated to actual parameters at the time of invocation by the PERFORM statement. This vehicle for transferring data to and from the procedure provides the facility to change, if desired, specified data items at each procedure invocation.

6.6.2.2 Formal syntax

Reference define-procedure-statement

6.6.2.3 Syntax diagram



6.6.2.4 Rules

1. Formal parameter names for a specific <define procedure statement> shall be unique. The names may be the same as labels or formal parameters of other procedures, however, it is recommended that such naming conflicts be avoided.
2. The parameter labels assigned in the formal parameter list will be written in place of the corresponding numbers or connections in the C/ATLAS statements within the <procedure body>.
3. When the procedure is referenced by a PERFORM statement, a value is passed to the <parameter>s, thus making the value available for use in subsequent statements within the <procedure body>.

Note—A <label> specified in a DECLARE statement in the <main preamble structure> may be referenced from within procedure bodies without the need to be passed as a formal parameter.

4. Any formal parameters appearing in the first parenthetical pair should only be referenced in the <procedure body> as an independent variable since they are called by value. Any assignment

to this class of parameters will not cause assignment of the equivalent actual parameter during procedure invocation.

5. Formal parameters appearing in the second parenthetical pair may be referenced in the <procedure body> as either dependent or independent variables. RESULT <parameter>s are called by reference, with the restriction that the actual parameters shall be variables.

6. Variables shall be assigned labels in the formal <parameter> list of the DEFINE <procedure> statement. This list follows the word PROCEDURE in the statement and is formed by writing the <parameter> labels of all values to be input to the procedure at execution time. The list is enclosed in parentheses.

7. A <procedure> can be made global by using the GLOBAL attribute. In this case, the <procedure> has to be unique within the entire <complete atlas test program structure>. A global <procedure> can be referenced in modules if it is established in the <atlas program structure> or in another module of the <complete atlas test program structure>. It can be referenced in an <atlas program structure> if it is established in a module of the <complete atlas test program structure>.

8. To reference a global <procedure>, the <define procedure statement> shall be used with the attribute EXTERNAL. The sequence of parameters has to be the same as in the corresponding global definition (i.e., the first parameter of the external definition corresponds to the first parameter of the global definition).

9. RESULT <parameter>s may be included in the formal parameter list. They are enclosed in a pair of parentheses that follows the word RESULT.

10. All formal <parameter>s are local and are followed by their <type>.

11. The TIMED DIGITAL path is used to define a TIMED DIGITAL procedure, which can only be invoked using a PERFORM, TIMED DIGITAL statement from within a <do timed digital body> (reference 12.5.7).

6.6.3 PROCEDURE body

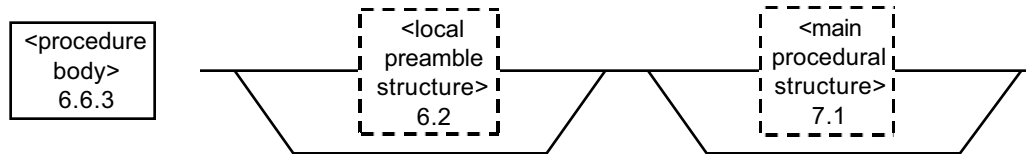
6.6.3.1 Function

The <procedure body> consists of an optional <local preamble> that may include only DECLARE statements. Those DECLARE statements may be utilized to provide local data declarations. The following <main procedural structure> consists of one or more C/ATLAS procedural statements that describe the desired logic.

6.6.3.2 Formal syntax

Reference procedure-body

6.6.3.3 Syntax diagram



6.6.3.4 Rules

1. Statement numbers appearing within a <procedure body> may only be referenced from within the bounds of that procedure.
2. Within a single <atlas program structure> or <atlas module structure>, a previously defined procedure may be invoked in a subsequent procedure by including a PERFORM statement within the <procedure body>. Recursion by the use of procedures in different <atlas module structure>s or the <atlas program structure> is not permitted (i.e., a C/ATLAS procedure can never be nested within itself).
3. Procedure bodies may not contain entry points (E-Flags).
4. When the <define procedure statement> is used with the attribute EXTERNAL, the <procedure body> is empty.
5. When the <define procedure statement> is used with the attribute TIMED DIGITAL, the <main procedural structure> will be restricted to the statements defined in the <do timed digital body> (reference 12.5.7).

6.6.4 LEAVE <procedure> statement

6.6.4.1 Function

Causes execution of a <procedure> to terminate prior to normal sequential execution of the END <procedure> statement.

6.6.4.2 Formal syntax

Reference leave-procedure-statement

6.6.4.3 Syntax diagram



6.6.4.4 Rules

1. When a LEAVE <procedure> statement is executed, control is transferred to the END <procedure> statement.
2. LEAVE <procedure> statements shall be located within the applicable <define procedure structure>.
3. Any structures in the procedure containing the LEAVE <procedure> statement are also exited upon execution of the LEAVE <procedure> statement.

6.6.5 END PROCEDURE statement

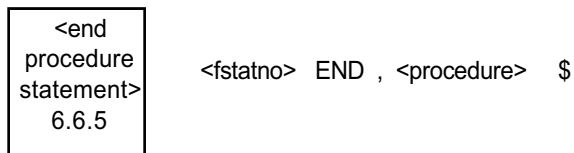
6.6.5.1 Function

To delimit a procedure definition.

6.6.5.2 Formal syntax

Reference end-procedure-statement

6.6.5.3 Syntax diagram



6.6.5.4 Rules

1. The procedure <label> following the END verb shall be identical to that appearing within the <define procedure statement>.
2. Following the execution of the <end procedure statement>, control returns to the first statement following the PERFORM that invoked the procedure.

6.7 REQUIRE statement

6.7.1 Function

The REQUIRE statement allows a test procedure writer to establish and label a Unit-Under-Test oriented summary of the characteristics needed by a test resource to implement the <complete atlas test program structure>. The test resources thus established are termed "virtual resources" since they refer to Unit-Under-Test requirements and not the "real resources" in the hardware used for implementation. The implementation of the <complete atlas test program structure> requires a checking and equating of the virtual resources to the real resources available for implementation. The <requirement> label serves as a linking mechanism that may be utilized in this process.