IEEE Std 1226.2-1993
(Issued April 1994
for trial use)

IEEE ABBET™

Trial-Use Standard for Ada-Based ATLAS-Level Test Procedure Interface for A Broad-Based Environment for Test (ABBET)

Sponsor

**Standards Coordinating Committee 20,
Abbreviated Test Language for All Systems (ATLAS)**

Approved December 2, 1993

IEEE Standards Board

Abstract: Ada packages that specify data types and services for the Ad-based ATLAS-level test procedure interface for a broad-based environment for test (ABBET) are defined. They are to be used to support the development of Ada-based, UUT-directed, signal-oriented test programs. **IEEE ABBET** uses the features of Ada to specify a comprehensive environment for integrating product design data, test requirements/strategies, and test results management in the implementation of automated test systems and test control programs.
Keywords: ABBET, Ada, C/ATLAS

ABBET is a trademark of the Institute of Electrical and Electronics Engineers, Inc.

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

Copyright © 1994 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1994. Printed in the United States of America.

ISBN 1-55937-386-5

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.

AVA Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1226.2-1993

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 are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Publication of this trial-use standard for comment and criticism has been approved by the Institute of Electrical and Electronics Engineers. Trial-use standards are effective for 24 months from the date of publication. Comments for revision will be accepted for 18 months after publication. Suggestions for revision should be directed to the Secretary, IEEE Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08817-1331, and should be received no later than August 30, 1995. It is expected that following the 24-month period, this trial-use standard, revised as necessary, shall be submitted to the IEEE Standards Board for approval as a full-use standard.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1226.2-1993

Introduction

[This introduction is not a part of IEEE Std 1226.2-1993, **IEEE ABBET** Trial-Use Standard for Ada-Based ATLAS-Level Test Procedure Interface for A Broad-Based Environment for Test (ABBET).]

In the 1960's, Aeronautical Radio Incorporated (ARINC) started the development of the Abbreviated Test Language for Avionics Systems (ATLAS). In 1976, management of the ATLAS standard was passed to the IEEE and the meaning of the ATLAS acronym was changed to Abbreviated Test Language for All Systems to reflect its broader field of application.

Within the IEEE, development of ATLAS-related standards (IEEE Std 416, IEEE Std 716, IEEE Std 771, etc.) was vested in the Standards Coordinating Committee 20 (SCC20), also known as the ATLAS Committee. In the mid-1980's, SCC20 broadened the scope of its activities to include other potential testing related standards including Automatic Test Program Generation (ATPG) and Test Equipment Description Language (TEDL).

The initial motivation for ATLAS development was to provide unambiguous human-to-human communication of test requirements. ATLAS evolved to become a test language that performs the functions of a specification language and a programming language. It fulfills an important role as a specification language by providing signal-oriented statements not found in other languages. However, as a general programming language, it suffers from a variety of limitations.

The cost of supporting the large variety of computer programming languages prompted the United States Department of Defense (DoD) to develop the Ada Programming Language (ISO/ANSI/MIL-STD-1815A-1983) and mandate its use for all DoD software applications.

Ada was developed to support advanced software engineering practices. Specifically, Ada addresses the controlled preparation and life cycle support of large complex programs. High-quality commercial Ada compilers and other development tools are now widely available. In addition, Ada supports the definition of standardized interfaces at various levels of abstraction. This capability can be used to develop a layered approach to the implementation of test software and provide programming capabilities beyond those supported by ATLAS.

By the late 1980's there was a growing feeling that Ada might provide the framework needed to integrate the various test-related functions. Eventually, it was generally agreed that an ideal standardized test domain would utilize the advantages of Ada programming and implementation features while retaining the high-level, UUT-oriented test specification aspects of ATLAS. The standardized Ada test environment would also facilitate integration of other test-related standards.

At the 89-1 SCC20 meeting in Edinburgh, Scotland, an IEEE Project Authorization Request (PAR) was prepared to develop an Ada Based Environment for Test (ABET) and the first draft of a requirements document was generated. The ABET PAR that was approved by the IEEE Standards Board on November 9, 1989 defined the scope of the proposed ABET standard as:

> **"To define an alternative environment for the implementation of ATLAS test specifications using the programming capabilities of the Ada Language."**

At the 90-1 SCC20 meeting in San Diego, California, the ABET requirements document and an associated rationale document were revised and approved by the ABET Subcommittee.

At the August, 1990 and October, 1990 meetings of the ABET subcommittee (in Washington, DC, and Fort Lauderdale, Florida, respectively) a draft ABET architecture document was developed. This base document for the IEEE Std 1226 set of ABET standards defines the ABET architecture and specifies the requirements for the component standards that define each of the ABET functions and their interfaces.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

In 1993, because of an acronym conflict with the IEEE Accreditation Board for Engineering Technology (ABET) and to reflect an increase in the scope of the 1226 family of standards, the name of the 1226 subcommittee and resulting standard was changed to ABBET (A Broad-Based Environment for Test).

This standard is the **IEEE ABBET** component standard that specifies the ABBET Ada-Based ATLAS-Level Test Procedure Interface (ALTPI). The first proposed draft of this standard (version 1.00) was submitted to the ABBET subcommittee by ITT Avionics at the May, 1990 meeting in San Diego, California. Draft version 1.01 was accepted as a strawman working draft document by the ABBET/ATLAS Working Group at the August, 1990 ABBET meeting. These proposed drafts, and subsequent updates, were developed by ITT Avionics with funding provided by the DoD DARPA/ISTO/STARS (Software Technology for Adaptable, Reliable Systems) program under ESD/AVS master contract No. F19628-88-D-0028 with Boeing Aerospace and Electronics and subcontract No. HA4278 with ITT Avionics.

All the drafts up to, and including, the version 5.1c draft developed in October, 1991, were based on the ATLAS verb-oriented approach. At the February, 1992 meeting, the ABBET subcommittee decided to change to a virtual resource object-oriented approach based on an analysis of several alternate proposals. The object-oriented version 6.0 draft that was developed in March, 1992 and later updates of this standard were funded by the USAF ABBET Technical Advisory Group (ATAG) through ASD/AEGB(SMGB) prime contracts with Softech and TRW.

Version 6.6 of this standard was balloted in November, 1992. It was updated to version 6.7 in response to the ballot comments and sent out for negative ballot resolution in October, 1993. The final version, 6.8, was approved by the IEEE Standards Board for publication as a trial-use standard on December 2, 1993.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

Participants

The ABBET/ATLAS Working Group that developed this standard consisted of the following membership:

Jehuda Ziegler, John M. Goddard, William A. Whitaker *Co-chairs*
Kerry Kugler, Gary L. Hardenburg, Paul B. Knese *Co-secretaries*

Donald Abernathie
Thomas Andrews
Christine Augeard
Jean-Yves Barbier
Eric Bergaud
Ivo Bordignon
Patrice Brard
Ronald Brooks
David Bui
Mark J. Cain
E. Elaine Cashar
Patrice Chassard
Jim Clemson
Frank Conforti
Andrew G. Costen
Jack Cross
Steven Damiani
Greg Darnell
Bernard Dathy
Christian Delval
Joseph C. DiGrazia
Pascal Dormal
Bernard Dumont
Pete Faulkner
Christine Fisher
Janet Floyd
Kenneth Fox
Philippe Frugier
Roger S. Goldberg
Jerry M. Grasso
T. Winston Griffin
Mitchell Hager
Randy J. Hansel
Bob Hardy
Gary Hinkle
Gary Jenkins
Helmuth Kaunzinger
Mike Kihara
Wayne Kim
Glenn Lafflin
Terry Lambert
Thomas Lane
Mats Landerholm
Larry Losz
Rolf Lundin
Michael S. Martin
Gayle Matysek
Bob McGarvey
Charles Morris
Joyce Mueller
Les Nelson
Tom Murphy
Peter Nguyen
John Odak
Kate O'Sullivan
Charles Oliverio
Catherine Ozenfant
Jean Pouilly
Keith R. Poulin
Robert E. Ramsey
James R. Reeder
Bill Reeves
Victor V. Reznack
Michael Rhodes
Robert Rolfe
Albert Sanson
Kenneth Schoonover
Michael Seavey
Steven C. Shutt
David W. Smith
Philip Smith
Robert Smith
Daniel Snyder
Joseph Stanco
Douglas Stinson
Carlos D. Vera
Roger L. Williams
Debora Wilkins
James B. Wyburn
Danny Yam
R. Mike Yarbrough
Bob Zimbardi

The following individuals provided significant proposals, reviews, and comments toward the development of this standard:

Donald H. Abernathie
Linda G. Burgermeister
Mark J. Cain
E. Elaine Cashar
Andrew Costen
Jack Cross
Patricia Cundiff
Greg Darnell
Bernard Dathy
Joseph C. DiGrazia
Thao Duong
Mark Finn
John M. Goddard
Jerry M. Grasso
T. Winston Griffin
Gary L. Hardenburg
Gary M. Jenkins
Kerry Kugler
Michael S. Martin
Gayle Matysek
Charles G. Morris
Joyce Mueller
Barbara Neyman
Peter Nguyen
John Odak
Carles Oliverio
James R. Reeder
John Scoggins
Michael L. Seavey
Steven C. Shutt
Douglas E. Sorensen
David W. Smith
Dan Snyder
Louis M. Springer
Joseph J. Stanco
Douglas Stinson
Mike Walters
William A. Whitaker
Roger L. Williams
Dennis Yaeger
Jehuda Ziegler

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

The following individuals were on the balloting committee:

Guy Adam
Jean-Yves Barbier
Eric Bergaud
Michael Blair
Ivo Bordignon
Patrice Brard
Alan Bridges
Jerrold Brody
Ron Brooks
Terry Campbell
E. Elaine Cashar
Andrew Costen
James Cottrell
Bernard Daniel
Greg Darnell
Bernard Dathy
Joseph DiGrazia
Bernd Dinklage
Pascal Dormal
Bernard Dugas
Christine Fisher
Jacques Flandrois
Jerry M. Grasso
Arnold Greenspan
T. Winston Griffin
Gary Hardenburg
Robert Hayes
Leonard Haynes
John E. Heiser
Gary L. Hinkle
Paul B. Knese
Kerry Kugler
Glenn Lafflin
Alfred Lowenstein
Rolf Lundin
Michael S. Martin
Gayle Matysek
Robert McGarvey
Harry McGuckin
Martin Meth
James Mills
Duncan Morrill
Charles Morris
William Neblett
Peter Nguyen
Goran Nordstrom
Hans Obermeier
Chuck Oliverio
Les Orlidge
Richard Patrick
Jean Pouilly
Narayanan Ramachandran
William Reeves
Robert Rolfe
Albert Sanson
Eric Saenz
Charles Scheetz
Helmut Scheibenzuber
Michael Seavey
John W. Sheppard
William Simpson
David Smith
Philip T. Smith
Joseph Stanco
Doug Stinson
William Taylor
Vince Terpstra
J. Richard Weger
William A. Whitaker
Kenneth Wilkinson
Roger L. Williams
Will Young
Jehuda Ziegler

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

When the IEEE Standards Board approved this standard on December 2, 1993, it had the following membership:

Wallace S. Read, *Chair* **Donald C. Loughry,** *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril
José A. Berrios de la Paz
Clyde R. Camp
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ramiro Garcia
Donald N. Heirman
Jim Isaak
Ben C. Johnson
Walter J. Karplus
Lorraine C. Kevra
E. G. "Al" Kiener
Ivor N. Knight
Joseph L. Koepfinger*
D. N. "Jim" Logothetis
Don T. Michael*
Marco W. Migliaro
L. John Rankine
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp
Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
David E. Soffrin
Stanley I. Warshaw

Rachel A. Meisel
IEEE Standards Project Editor

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

Contents

CLAUSE PAGE

1. Overview . . . 1
1.1 Scope . . . 1
1.2 Purpose . . . 1
1.3 Application . . . 1

2. References . . . 2

3. Definitions . . . 4
3.1 Key terms . . . 4
3.2 Abbreviations and acronyms . . . 4

4. Description . . . 5
4.1 Objectives . . . 5
4.2 Overview . . . 5
4.2.1 Test Resources and Classes . . . 6
4.2.2 Ada implementation of the resource classes and objects . . . 6
4.2.3 Links between object classes . . . 7
4.2.4 Structural overview of analog resources . . . 7
4.2.5 Structural overview of digital resources . . . 8
4.2.6 Data bus testing structure . . . 8
4.2.7 Resource allocation . . . 9
4.3 Relationship to ABBET architecture . . . 10
4.3.1 ABBET conforming implementations . . . 11
4.4 Relationship to reference documents . . . 12

5. Specification . . . 14
5.1 Test program structure . . . 14
5.1.1 Program components . . . 14
5.1.2 Test labeling . . . 14
5.1.3 Test encapsulation . . . 15
5.1.4 Multitasking support . . . 17
5.1.5 Required resource instantiation . . . 17
5.1.6 Test program elaboration . . . 19
5.1.7 Status flag operations . . . 21
5.1.8 Test completion . . . 23
5.2 Data processing . . . 23
5.2.1 Authorized characters . . . 23
5.2.2 Number representations . . . 23
5.2.3 Scope of identifiers . . . 24
5.2.4 Program control . . . 24
5.2.5 Computational operations . . . 24
5.2.6 Mathematical operations . . . 24
5.2.7 Digital data operations . . . 24
5.2.8 ABBET standard physical units . . . 25
5.2.9 Physical data conversions . . . 25
5.3 Input/output . . . 26
5.3.1 File input/output . . . 26
5.3.2 Operator input/output . . . 26
5.4 Connection, path requirements . . . 26
5.4.1 Pin descriptors . . . 26
5.4.2 Connection identifiers . . . 29

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

5.4.3 Path description 32
5.4.4 Connection error exception 35
5.5 Resource capability requirements 35
5.5.1 ANALOG_SENSOR_CAPABILITY 35
5.5.2 ANALOG_SOURCE_CAPABILITY 36
5.5.3 SIMPLE_LOAD_CAPABILITY 37
5.5.4 TIME_INTERVAL_SENSOR_CAPABILITY 37
5.5.5 TIMER_SOURCE_CAPABILITY 37
5.5.6 DIGITAL_SENSOR_CAPABILITY 38
5.5.7 DIGITAL_SOURCE_CAPABILITY 38
5.6 Event monitor capability 39
5.6.1 ANALOG_SIGNAL_MONITOR_CAPABILITY 39
5.6.2 ANALOG_TIME_MONITOR_CAPABILITY 39
5.6.3 MANUAL_INTERVENTION_CAPABILITY 39
5.6.4 DIGITAL_EVENT_MONITOR_CAPABILITY 40
5.7 Event monitor resources 40
5.7.1 ANALOG_SIGNAL_MONITOR 40
5.7.2 ANALOG_TIME_MONITOR 41
5.7.3 MANUAL_INTERVENTION 41
5.7.4 DIGITAL_EVENT_MONITOR 42
5.8 UUT-directed virtual resources 42
5.8.1 ANALOG_SENSOR 44
5.8.2 ANALOG_SOURCE 46
5.8.3 SIMPLE_LOAD 48
5.8.4 TIME_INTERVAL_SENSOR 49
5.8.5 TIMER_SOURCE 51
5.8.6 DIGITAL_SENSOR 52
5.8.7 DIGITAL_SOURCE 53
5.9 Signal oriented procedures 54
5.9.1 State transition diagrams 56
5.9.2 Single-action verbs 62
5.9.3 Multiple action verbs 67
5.9.4 Digital verbs 70
5.9.5 ANTICIPATE_VIA_CONNECTION Procedure 74
5.10 Test timing and synchronization procedures 75
5.10.1 Timer Source WAIT and RESET procedures 75
5.10.2 RESET_ALL_TIMERS, RESET_TIMERS 76
5.10.3 DO_SIMULTANEOUS 76
5.10.4 DO_TIMED_DIGITAL 76
5.10.5 STIM_RATE_ONLY 78
5.10.6 STIM_EVENT_ONLY 78
5.10.7 STIM_RATE_AND_SENSE_DELAY 79
5.10.8 STIM_RATE_AND_SENSE_EVENT 79
5.10.9 STIM_EVENT_AND_SENSE_DELAY 79
5.10.10 STIM_EVENT_AND_SENSE_EVENT 80
5.10.11 SENSE_RATE_ONLY 80
5.10.12 SENSE_EVENT_ONLY 81
5.11 Digital bus testing procedures 81
5.12 Field/subfield data definitions 84
5.12.1 ATLAS_TYPES 85
5.12.2 NOUN_MODIFIER_TYPES 97
5.12.3 RESOURCE_DESCRIPTION 104
5.12.4 SIGNAL_DESCRIPTION 107

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

5.12.5 SYNCHRONIZATION_TYPES . . . 115
5.12.6 EVENT_DESCRIPTION . . . 121
5.12.7 DIGITAL_DESCRIPTION . . . 123
5.12.8 DATA_BUS_DESCRIPTION . . . 135
5.13 Nouns and their modifiers . . . 144
5.13.1 Standard ATLAS nouns . . . 144
5.13.2 AC_SIGNAL (alternating current signal) . . . 146
5.13.3 ADF (Automatic direction finder) . . . 147
5.13.4 AM_SIGNAL (Amplitude modulation signal) . . . 148
5.13.5 AMBIENT_CONDITIONS . . . 148
5.13.6 ATC (Air traffic control) . . . 150
5.13.7 COMMON . . . 152
5.13.8 DC_SIGNAL (Direct current signal) . . . 154
5.13.9 DISPLACEMENT . . . 154
5.13.10 DME (Distance measuring equipment) . . . 154
5.13.11 DOPPLER . . . 155
5.13.12 EARTH . . . 156
5.13.13 EM_FIELD (Electro magnetic field) . . . 157
5.13.14 EVENTS . . . 157
5.13.15 FLUID_SIGNAL . . . 158
5.13.16 FM_SIGNAL (Frequency modulation signal) . . . 160
5.13.17 HEAT . . . 160
5.13.18 IFF (Identification, friend or foe) . . . 160
5.13.19 ILS (Instrument landing system) . . . 162
5.13.20 IMPEDANCE . . . 164
5.13.21 LIGHT . . . 164
5.13.22 LOGIC_CONTROL . . . 164
5.13.23 LOGIC_DATA . . . 165
5.13.24 LOGIC_LOAD . . . 173
5.13.25 LOGIC_REFERENCE . . . 174
5.13.26 MANOMETRIC . . . 175
5.13.27 PAM (Pulse amplitude modulation) . . . 175
5.13.28 PM_SIGNAL (Phase modulated signal) . . . 176
5.13.29 PULSED_AC (Pulsed alternation current signal) . . . 176
5.13.30 PULSED_AC_TRAIN . . . 177
5.13.31 PULSED_DC . . . 179
5.13.32 PULSED_DC_TRAIN . . . 179
5.13.33 PULSED_DOPPLER . . . 181
5.13.34 RADAR_SIGNAL . . . 181
5.13.35 RAMP_SIGNAL . . . 182
5.13.36 RANDOM_NOISE . . . 182
5.13.37 RESOLVER . . . 182
5.13.38 ROTATION . . . 183
5.13.39 SHORT . . . 187
5.13.40 SQUARE_WAVE . . . 187
5.13.41 STEP_SIGNAL . . . 188
5.13.42 SUP_CAR_SIGNAL (Suppressed carrier signal) . . . 188
5.13.43 SYNCHRO . . . 189
5.13.44 TACAN (Tactical air navigation) . . . 189
5.13.45 TIME_INTERVAL . . . 193
5.13.46 TRIANGULAR_WAVE_SIGNAL . . . 193
5.13.47 TURBINE_ENGINE_DATA . . . 193
5.13.48 VIBRATION . . . 194

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

5.13.49 VOR (VHF omnidirectional radio range) . . . 195
5.13.50 WAVEFORM . . . 196
5.14 Noun modifiers . . . 199
5.14.1 Identifiers for pulse-type signals . . . 201
5.14.2 Standard ATLAS noun modifiers . . . 201
5.14.3 Standard ATLAS noun modifier prefixes . . . 206
5.14.4 Standard ATLAS noun modifier suffixes . . . 206
5.14.5 Standard noun modifier enumeration values . . . 213
5.14.6 Modifier value descriptors . . . 214
5.14.7 Noun modifier definitions . . . 215

6. Conformance . . . 239
6.1 Ada language capabilities . . . 239
6.2 ABBET test procedure language capabilities . . . 239
6.3 Subsets and extensions . . . 239
6.3.1 Supported nouns, modifiers, and connections . . . 239
6.3.2 Reusable higher-level test procedure components . . . 245

7. Ada Packages . . . 246
7.1 ATLAS_TYPES package specification . . . 248
7.2 NOUN_MODIFIER_TYPES package specification . . . 257
7.3 SIGNAL_DESCRIPTION package specification . . . 264
7.4 RESOURCE_DESCRIPTION package specification . . . 273
7.5 EVENT_DESCRIPTION package specification . . . 276
7.6 DIGITAL_DESCRIPTION package specification . . . 278
7.7 PATH_DESCRIPTION package specification . . . 284
7.8 SYNCHRONIZATION_TYPES package specification . . . 287
7.9 ATLAS_NOUNS_MODIFIERS_AND_CONNECTIONS package specification . . . 292
7.10 ANALOG_SENSOR_CAPABILITY package specification . . . 305
7.11 ANALOG_SOURCE_CAPABILITY package specification . . . 306
7.12 SIMPLE_LOAD_CAPABILITY package specification . . . 307
7.13 TIME_INTERVAL_SENSOR_CAPABILITY package specification . . . 308
7.14 TIMER_SOURCE_CAPABILITY package specification . . . 309
7.15 ANALOG_SENSOR package specification . . . 310
7.16 ANALOG_SOURCE package specification . . . 313
7.17 SIMPLE_LOAD package specification . . . 315
7.18 TIME_INTERVAL_SENSOR package specification . . . 317
7.19 ANALOG_SIGNAL_MONITOR_CAPABILITY package specification . . . 319
7.20 ANALOG_TIME_MONITOR_CAPABILITY package specification . . . 320
7.21 MANUAL_INTERVENTION_CAPABILITY package specification . . . 321
7.22 ANALOG_SIGNAL_MONITOR package specification . . . 322
7.23 ANALOG_TIME_MONITOR package specification . . . 323
7.24 MANUAL_INTERVENTION package specification . . . 324
7.25 DIGITAL_EVENT_MONITOR_CAPABILITY package specification . . . 325
7.26 DIGITAL_SENSOR_CAPABILITY package specification . . . 326
7.27 DIGITAL_SOURCE_CAPABILITY package specification . . . 327
7.28 DIGITAL_EVENT_MONITOR package specification . . . 328
7.29 DIGITAL_SENSOR package specification . . . 329
7.30 DIGITAL_SOURCE package specification . . . 331
7.31 TIMER_SOURCE package specification . . . 332
7.32 SIMULTANEOUS_DIGITAL package specification . . . 333
7.33 DO_TIMED_DIGITAL package specifications . . . 334
7.34 DATA_BUS_DESCRIPTION package specification . . . 339

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

7.35 EXCHANGE_DEFINITION package specification 349
7.36 PRIMARY_DATA_BUS package specification 350
7.37 ALTERNATE_DATA_BUS package specification 351
7.38 DO_DATA_BUS_EXCHANGE package specification 352
7.39 ABBET_EVALUATIONS package specification 353
7.40 VIRTUAL_RESOURCE_GROUP package specification 355
7.41 ABBET_TEST_CAPSULE package specification 356
7.42 ATLAS_CONTROL package specification 357

Annex A. Test program examples 358
A.1 Supported NOUNS_MODIFIERS_AND_CONNECTIONS package specification 359
A.2 ALTPI analog TPS example 364
A.2.1 ANALOG_TEST_OTHER_UNITS package 365
A.2.2 TEST_CONNECTIONS package 366
A.2.3 ANALOG_TEST_CAPABILITIES package 368
A.2.4 ANALOG_TEST_RESOURCES package 370
A.2.5 ANALOG_TEST_VIRTUAL_RESOURCE_GROUPS package 372
A.2.6 ANALOG_TEST Procedure 373
A.2.7 ANALOG_TEST_CAPSULE package 375
A.2.8 CAPSULE_EXECUTION Procedure 377
A.3 ALTPI Digital TPS Example 378
A.3.1 XMITRCV package 378
A.4 ALTPI noun, modifier extension TPS example 382
A.4.1 VACSENCP package 382
A.4.2 VACUUT package 382
A.4.3 VACTEST package 383
A.5 ATE device-to-device connection extension example 384

Annex B. Standard package file organization 386

Index to ABBET Ada identifiers 388

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

Figures

Figure 1—Hardware and software transition from UUT to instrument 5
Figure 2—Structural overview of analog resources 8
Figure 3—Structural overview of digital resources 9
Figure 4—Data bus testing structure 9
Figure 5—ABBET architecture as defined in IEEE Std 1226-1993 10
Figure 6—ABBET test and diagnostic services 11
Figure 7—Layered implementation of ABBET ALTPI and lower-level interfaces 12
Figure 8—ALTPI package elaboration order diagram 19
Figure 9—State diagram for a virtual analog source or load resource without event monitor 56
Figure 10—State diagram for a virtual analog source or load resource with an event monitor 57
Figure 11—State diagram for a virtual analog sensor without event monitor 58
Figure 12—State diagram for a virtual analog sensor with an event monitor 59
Figure 13—Single action verb illustration 63
Figure 14—Example single 1553B bus interfacing several avionics systems 82
Figure 15—Illustration of RZ, NRZ, and BIP pulse types for 2#1011_0010# 125
Figure 16—ATC interrogation signal 151
Figure 17—ATC reply signal 151
Figure 18—Defined voltage ranges for these examples 173
Figure 19—Pulsed ac train for example 1 178
Figure 20—Pulsed ac train for example 2 178
Figure 21—Pulsed dc train for example 1 180
Figure 22—Pulsed dc train for example 2 180
Figure 23—Inertially (Earth) stabilized coordinate frame 183
Figure 24—UUT referenced stabilized coordinate frame 184
Figure 25—Rotation quantity, rate, and acceleration illustration 185
Figure 26—Upside-down missile UUT heading north 186
Figure 27—Definition of waveform period, sample_time, and sample_width 197
Figure 28—Example waveform with voltage samples 198
Figure 29—Voltage stepped and ramped waveforms 199
Figure 30—Modifiers used with pulse-type signals 202
Figure 31—Peak and peak-to-peak characteristics for voltage waveforms 210
Figure 32—Instantaneous voltage characteristics 211
Figure 33—Defined voltage ranges for the example 237

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

THIS PAGE WAS
BLANK IN THE ORIGINAL

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1226.2-1993

IEEE ABBET™

Trial-Use Standard for Ada-Based ATLAS-Level Test Procedure Interface for A Broad-Based Environment for Test (ABBET)

1. Overview

The set of standards and guides that specify A Broad-Based Environment for Test (ABBET) was developed by the ABBET Subcommittee of the IEEE Standards Coordinating Committee 20 (SCC20). The initial set of **IEEE ABBET** component standards specify ISO/ANSI/MIL-STD-1815A-1983[1] Ada programming language constructs for performing automatic testing and integrated diagnostics. ABBET uses the features of Ada to specify a comprehensive environment for integrating product design data, test requirements/strategies, and test results management in the implementation of automated test systems and test control programs.

1.1 Scope

This standard specifies Ada packages that define general purpose test support services and data types needed to complete the definition and implementation/application of the other **IEEE ABBET** component standards. Packages are included in this standard if they are applicable to more than one ABBET interface.

This standard specifies the Ada-based ATLAS-Level Test Procedure Interface (ALTPI) for A Broad-Based Environment for Test (ABBET) as described in the IEEE Std 1226-1993.

1.2 Purpose

The purpose of this standard is to specify Ada constructs for writing automated test programs in Ada that are independent of the automated test equipment (ATE) on which they are to be executed. This is accomplished by defining a set of Ada constructs that provide the same UUT-directed, signal-oriented test capabilities of IEEE Std 716-1989.

This trial-use standard incorporates the capabilities of IEEE Std 716-1989. The intent is to update this standard to incorporate the enhancements to the C/ATLAS language being defined in the revision of IEEE Std 716-1989 (anticipated to be completed in 1994) during the two year trial-use period.

1.3 Application

This standard can be used to develop:

a) Test systems that provide the services specified herein
b) Test programs that use the services specified herein
c) Software tools that help create and execute ATLAS-level Ada test programs (e.g., ATPG and test strategy tools)

The Ada subprograms and data structures specified in this standard define the interface between the test system and the test programs that it hosts. The system's test executive software provides the specified test services by implementing the bodies of these subprograms. The test programs use these services by calling the subprograms.

[1] Information on references can be found in clause 2.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

2. References

This standard shall be used in conjunction with the exact versions of the publications referenced herein:

ARDC Standard Atmosphere Survey 115, Cambridge Research Center.

ARINC Spec 570, Automatic Direction Finder (ADF).[2]

ARINC Spec 572 and 711, Transponder and VOR Receiver.

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

ARINC Spec 575, Subsonic Air Data System, Bi-Polar RZ Pulse Class.

ARINC Spec 578, Instrument Landing System (ILS).

ARINC Spec 579, VHF Omnidirectional Radio Range (VOR).

ARINC Spec 626-1-1989, Standard ATLAS for Modular Testing.

CCITT Document, Volume 3, Rec G792, Entry No. 8003, AMI Pulse Class.[3]

CCITT Document, Volume 3, Rec G792, Entry No. 8005, HDB Pulse Class.

ICAN Standard Atmosphere International Committee on Air Navigation, 1924.

ICAO Standard Atmosphere 1964 Document 7488.2.

ICAO ANEX 10 Sections 3.1 and 3.5, DME and ILS Frequency Pairing.

IEEE Std 100-1992, The New IEEE Standard Dictionary of Electrical and Electronics Terms (ANSI).[4]

IEEE Std 268-1982, American National Standard Metric Practices (ANSI).

IEEE Std 416-1984 (Reaff 1991), IEEE Standard ATLAS Test Language (ANSI).

IEEE Std 716-1989, IEEE Standard C/ATLAS Test Language.

IEEE P716-1994, C/ATLAS Test Language.[5]

IEEE Std 993-1990, Trial-Use Standard for Test Equipment Description Language (TEDL)

IEEE Std 1029.1-1992, Waveform and Vector Exchange Specification (WAVES)

---

[2] ARINC publications are available from the Aeronautical Radio, Inc., 2551 Riva Road, Sales Dept., Annapolis, MD 21401, USA.

[3] CCITT publications are available from the CCITT General Secretariat, International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse, and in the United States from the National Technical Information Service, Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161, USA.

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

[5] Draft versions of authorized standards projects that have not yet been approved by the IEEE Standards Board are indicated by an IEEE Pnnn.n project authorization number. A manuscript copy of the latest draft version is available from the IEEE Standards Department, (908) 562-3800.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

IEEE Std 1226-1993, Trial-Use Standard for A Broad-Based Environment for Test (ABBET), Overview and Architecture.

IEEE Std 1226.1-1993, Trial-Use Standard for Common Ada Packages for A Broad-Based Environment for Test (ABBET).

IEEE P1226.3, ABBET Test Equipment Configuration Interface (TECI).

IEEE P1226.4, ABBET Test Resource Interface (TRI).

IEEE P1226.5, ABBET Bus Interface (BI).

ISO 1000 : 1981, SI units and recommendations for the use of their multiples and of certain other units.[6]

ISO/ANSI/MIL-STD-1815A-1983, Ada Programming Language.

ISO 2955 : 1983, Information processing—Representation of SI and other units in systems with limited character sets.

MIL-A-28826, Identification, Friend or Foe (IFF) (Classified).[7]

MIL-STD-291B, Distance Measuring Equipment (DME) Specification.

Telemetry Standards Document 106-71, issued by the Telemetry Working Group (TWG) of the Inter-Range Instrumentation Group of the Range Commanders Council (RCC), White Sands Missile Range, NM (Revised January 1971).

WADC Standard Atmosphere Technical Report 54-215, Wright Patterson AFB, OH.

[6] ISO 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.

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

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

3. Definitions

This clause defines key terms, abbreviations, and acronyms that are used in this standard. General ABBET terms are defined in clause 3 of IEEE Std 1226-1993. For electrical and electronics terms not defined in this clause, refer to IEEE Std 100-1992. For definitions of all the Ada language terms, refer to ISO/ANSI/MIL-STD-1815A-1983.

3.1 Key terms

3.1.1 SI: Abbreviation for "International System of Units" that are specified in ISO 1000 : 1981 for use in the measurement of physical quantities.

3.1.2 test object: An encapsulated stand-alone, executable test procedure.

3.1.3 virtual resource: An abstract representation of a UUT-directed, signal-oriented test capability. It is represented in ALTPI by instantiation of an Ada generic package with links to the test capability and path description objects.

3.1.4 `with`: Ada language construct to make the contents of an external Ada library unit visible within another Ada compilation unit.

3.2 Abbreviations and acronyms

ABBET	A Broad-Based Environment for Test
ABET	Ada-Based Environment for Test
ALTPI	Ada-Based ATLAS-Level Test Procedure Interface for ABBET
ATE	Automated Test Equipment
ATLAS	Abbreviated Test Language for All Systems (as defined in IEEE Std 416-1984, IEEE Std 716-1985, IEEE Std 716-1989, and ARINC specification 626-1-1989)
ATPG	Automated Test Program Generation (tools)
C/ATLAS	Common/Abbreviated Test Language for All Systems (as defined in IEEE Std 716-1985, IEEE Std 716-1989)
CAP	Common Ada Packages for ABBET (IEEE Std 1226.1-1993)
UUT	Unit Under Test

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

4. Description

4.1 Objectives

This standard specifies the Ada-Based ATLAS-Level Test Procedure Interface for ABBET as defined in IEEE Std 1226-1993. This interface provides UUT-directed, signal-oriented test commands that are equivalent to those defined in IEEE Std 716-1989 C/ATLAS for generating automated test programs that are independent of the ATE on which they will run. General programming support is provided by the standard Ada language constructs defined in ISO/ANSI/MIL-STD-1815A.

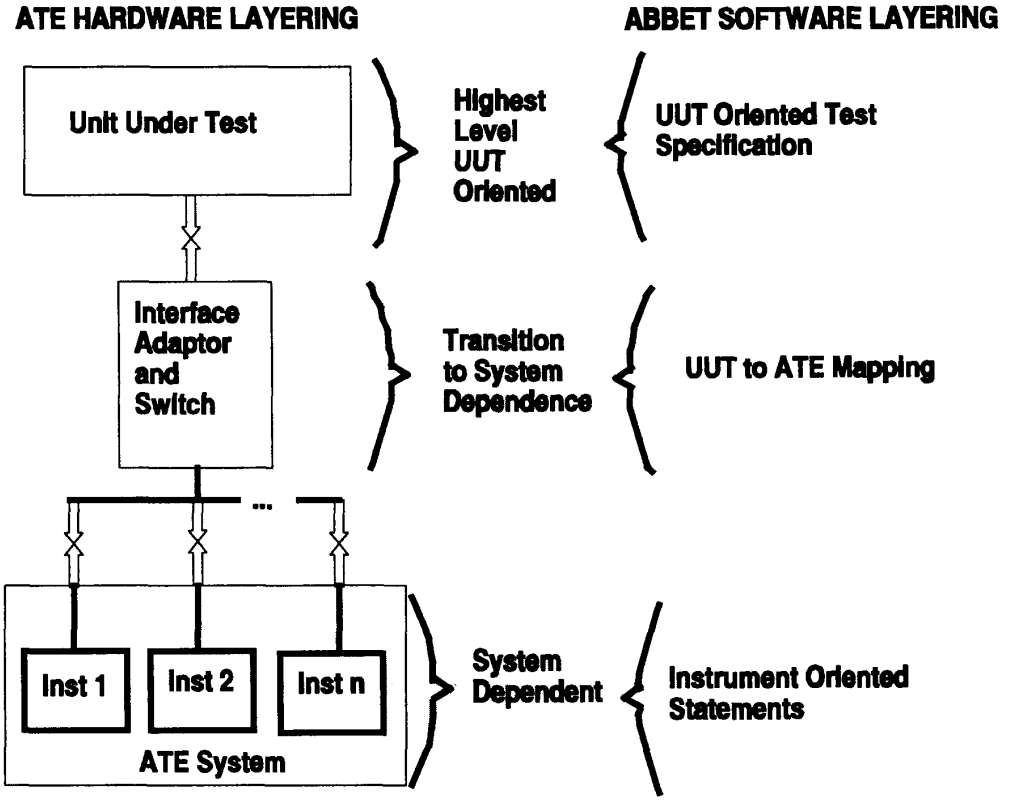


Figure 1—Hardware and software transition from UUT to Instrument

Figure 1 illustrates the relationship between the UUT, ITA, and ATE hardware elements and the corresponding software domains that transitions UUT-directed test specifications to ATE instrument-directed commands. UUT-directed test programs are transitioned to the specific instruments in the host ATE system by UUT to ATE mappings defined by the ABBET ATE system interface.

4.2 Overview

This subclause provides an overview of the elements of this standard and its relationship to the ABBET architecture and reference documents.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

4.2.1 Test resources and classes

ALTPI provides an object-oriented approach to performing UUT-directed, signal-oriented testing. There are five categories of generic signal-oriented resource classes and one category of connection characteristics that support test operations. These are the following:

Class Category	Included Classes
Test Resource Capability	Analog Source, Analog Sensor, Simple Load, Digital Source, Digital Sensor capabilities
Event Monitor Capability	Analog, Time, Digital and Manual Intervention Capability
Event Monitor	Analog, Time, Digital and Manual Intervention Actuations
Connection Characteristics	Path Description
Virtual Resource	Analog Source, Analog Sensor, Simple Load, Digital Source, Digital Sensor signal controls
Test Resource Synchronization	Timer Synchronization, Do-Simultaneous, Do-Timed-Digital

Objects produced from the classes in the test resource capability category are data objects that describe the signal capabilities of the test resource. These data objects are later bound to executable objects produced from the virtual resource classes.

Objects produced from the classes in the event monitor capability category are data objects that describe the capabilities of the event monitor resource. These data objects are later bound to executable objects produced from the event monitor classes.

Objects produced from the classes in the event monitor category provide both specific event description information and executable procedures to enable/disable the event. An event monitor object is created by binding an event monitor capability object to it. This binding specifies the capabilities to be used by the event monitor.

Objects produced from the path description class in the connection characteristics category are data objects that define the UUT signal connections (pins, paths) that are then bound to the test resources when the classes in the virtual resource category are instantiated. The number of pins and connection types have to match the terminals on the virtual resource with which it is bound.

Objects produced from the classes in the virtual resource category provide executable UUT-directed, signal-oriented procedures. A virtual resource object is created by binding a test resource capability object to it. This binding specifies the capabilities to be used by the virtual resource. Optionally, the virtual resource object can bind to itself one or two event monitor objects that will perform signal synchronization or gating. Additionally, the virtual resource object is bound with objects of the path description class.

The classes in the test resource synchronization category include timed digital STIM, SENSE, and DELAY events.

4.2.2 Ada implementation of the resource classes and objects

The specification of the resource classes is accomplished in this standard by defining corresponding Ada generic packages. To create an object of a given class, the generic package is instantiated with the data values that characterize the particular object. Default parameter values for the class are defined for each of the optional generic parameters.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

This classification bounds the number of objects in the ABBET ATLAS-level specification to a reasonable number. As they are generics, extensions (i.e., nouns, noun modifiers, and connections) within the classes are easily made.

The instantiation of a test resource capability generic package defines a class of testing capabilities. The instantiation of the virtual resource package provides the values of the parameters that describe the specific resource capabilities and UUT connections. This constrains the general capabilities of the virtual resource class into a specified subset. The physical world analogy is a multimeter that provides a large set of testing capabilities. Setting the multimeter to a particular function (voltmeter, ammeter, ohmmeter, etc) is analogous to instantiating the corresponding virtual resource object. This concept allows the Ada compiler to check subsequent uses of the test functions by a test program against the capabilities of the resource object.

Each of the virtual resource generic packages defines a set of signal-oriented control procedures applicable to the resource class. Virtual resource objects are created by instantiating the virtual resource generic packages. These objects are accessed from the ABBET Ada-based ATLAS-level test programs by calling (with dotted notation) the signal-oriented procedures defined in the corresponding generic virtual resource package.

4.2.3 Links between object classes

A key to this approach is the binding links between associated objects. The package specification for a class in the *_CAPABILITY category defines an identifier record that contains a resource number and the data structure characteristics that are required for creating and binding to lower level objects. The event monitor objects have an event identifier that is assigned to the event monitor object at its instantiation. The created virtual resource object uses these links to bind to signal timing and test resource capabilities. Additionally, the virtual resource is bound to UUT by the TPS/UUT defined enumeration types for the UUT pins and path sets.

This structure provides the functional capabilities of ATLAS in an object-oriented system. The interaction between classes of objects is precise and unambiguous. This setup allows for building complex testing objects from known capabilities in a predictable fashion. With this structure, it is possible to know the complete linkage between the test resource, signal timing, and paths to the UUT of a virtual resource object at the time of the object's instantiation. The implication of this is that all resource checking can be done at elaboration time. This means the ABBET/ATLAS-level interface can be implemented as a run-time process without the need for special preprocessors or postprocessors.

The data structures created at the elaboration of the virtual resource object are intentionally constrained by the test resource object to which it is bound. These constraints reduce programming and program errors. The test resource capability definitions allow a run-time system to easily check correctness from within a virtual resource object to which it is bound. This system structure should completely eliminate the need for any run-time semantic checking of the data input to, or output from, calls to the signal-oriented procedures. The test resource capability object is a template for a particular signal-oriented function that can be assumed to be semantically correct because it elaborated correctly. The only run-time requirement on the signal-oriented procedures is that the input data match the test resource capability object to which it is bound.

This process of instantiating virtual resource objects essentially provides the functions of the ATLAS preamble constructs (REQUIRE, IDENTIFY, etc.) during the elaboration of the Ada test program. The signal timing requirements (events) are also moved to the elaboration of the virtual resource object.

4.2.4 Structural overview of analog resources

A structure chart of the relationship between the resource and event monitor capability categories, the virtual resource control and event monitor actuation categories, and the UUT pins and paths is presented in Figure 2.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

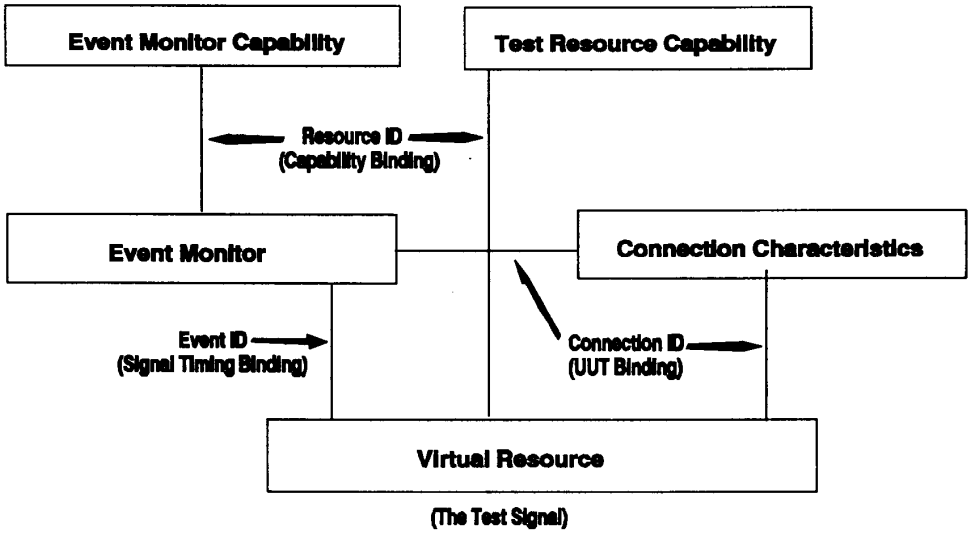


Figure 2—Structural overview of analog resources

The test resource capability and event monitor capability objects define the signal characteristics of the resources. The connection characteristics define the required paths to connect the resources to the UUT and to special ATE resources (common, ground, etc.). Event monitor resources are created from the event monitor capability objects and associated connection characteristics. The virtual resource objects are then created from the test resource capability and associated connection characteristics objects with an optional event monitor resource object.

These objects are bound to each other with the resource, event, and connection ID identifiers.

4.2.5 Structural overview of digital resources

A structure chart describing the relationship between the digital resource and event monitor capability categories, digital virtual resource control and event monitor actuation categories, the UUT pins and paths, and the digital test resource synchronization objects is presented in Figure 3.

The digital resource capability and digital event monitor capability objects define the signal characteristics of the resources. The connection characteristics define the required paths to connect the resources to the UUT and to special ATE resources (common, ground, etc.). Digital event monitor resources are created from the digital event monitor capability objects and associated connection characteristics. The digital virtual resource objects are then created from the digital resource capability and associated connection characteristics objects with an optional digital event monitor resource object. The digital virtual resources can also be related in a test resource synchronization object.

These objects are bound to each other with the resource, event, and connection ID identifiers and timed digital construct.

4.2.6 Data bus testing structure

The relationship between the data bus testing packages is presented in Figure 4.

Do data bus exchange objects are created from exchange definition and event monitor resources. These are in turn created from primary/alternate data bus objects that correspond to externally (non-ABBET) defined data bus specifications.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

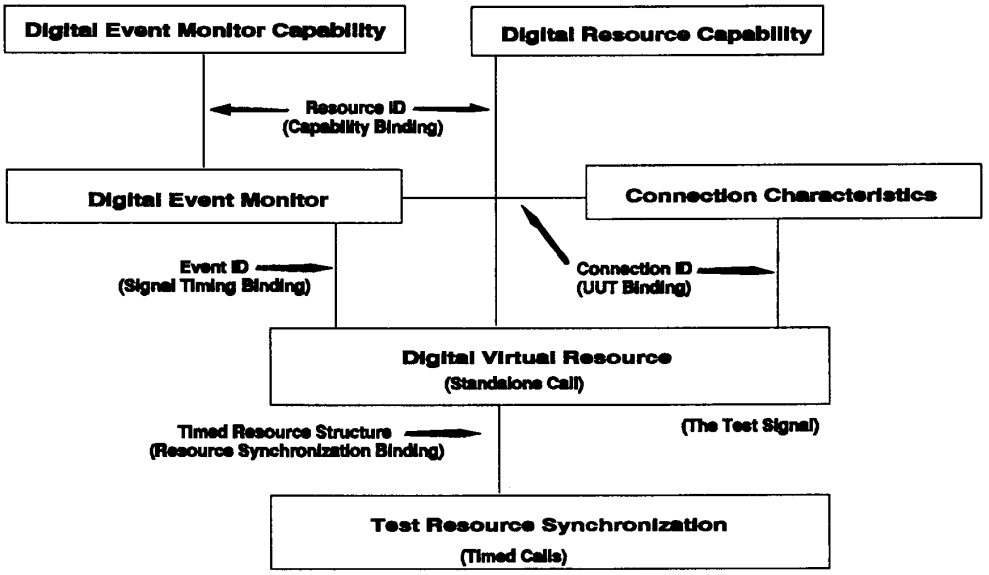


Figure 3—Structural overview of digital resources

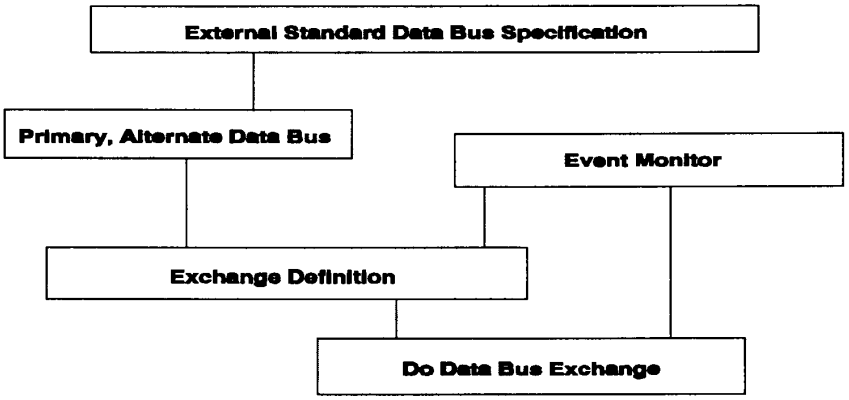


Figure 4—Data bus testing structure

4.2.7 Resource allocation

This standard supports two methods for specifying virtual resource requirements and allocating them to real physical resources.

The preferred approach is for the TPS writer to specify the required resource capabilities by instantiating the required resource capability objects. This approach is modeled after C/ATLAS where the required virtual resources are specified by the TPS in a manner that is theoretically independent of the ATE system. Most ATLAS systems perform resource allocation at compile time with ATLAS compilers tailored to the ATE system. Some ATLAS systems may perform dynamic allocation, but in most cases the ATLAS functionality provided by a particular ATE system is bounded. In ABBET this allocation could be accomplished by the ATE test executive at the TPS elaboration time for the created object.

It is also possible to present the capabilities provided by a particular ATE system to the TPS writer. This is accomplished by instantiating the resource capability objects that are supported by the ATE system and

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

allowing the TPS developer to `with` them into the TPS code. These capability objects are independent package instantiations that reside in the Ada library as independent units. The TPS `with`'s in only the required resource objects. The Ada compiler will verify that only resources available in the library are referenced.

The allocation of the resources required by the TPS to the resources available in the ATE can be accomplished manually by the TPS developer, automatically by a TPS development tool, or dynamically by the real-time test executive.

4.3 Relationship to ABBET architecture

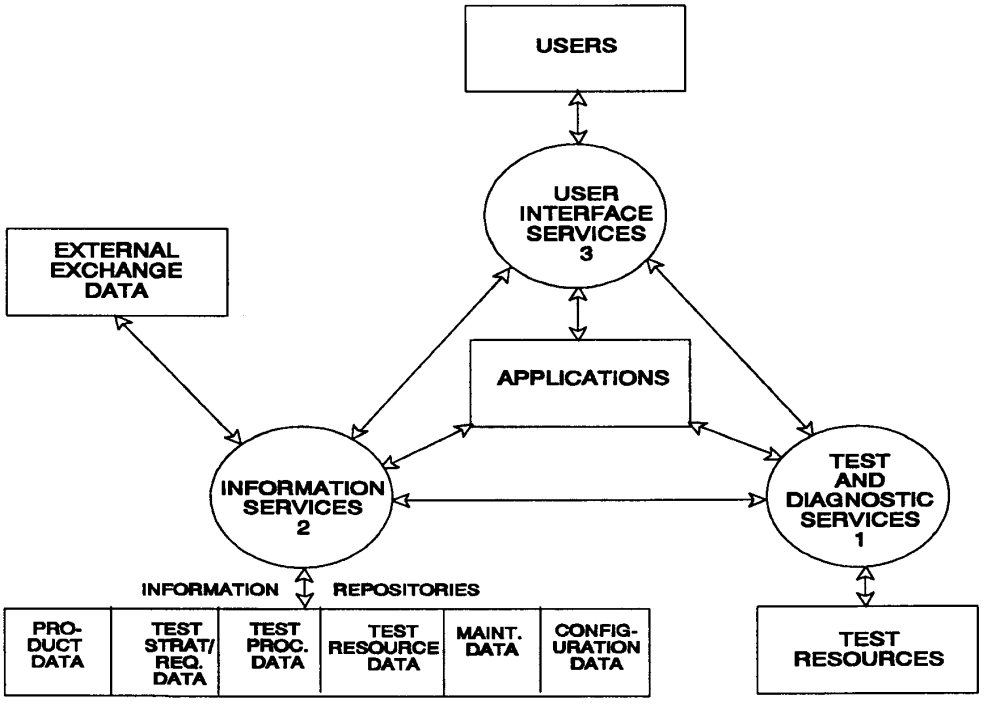


Figure 5—ABBET architecture as defined in IEEE Std 1226-1993

The ABBET architecture defined in IEEE Std 1226-1993 (see Figure 5) consists of a framework of standardized test-related service interfaces and information representations.

This standard specifies the services to be provided at the test procedure layer interface of the ABBET test and diagnostics services grouping.

The ABBET test and diagnostics services (see Figure 6) include a hierarchical set of interface layers that relate the UUT-directed test requirements to the specific ATE resources on which they are to be tested. The layers include the test strategy/requirements, test procedure, test resource management, and instrument control interfaces used to perform software-controlled, automated testing.

The test strategy/requirements layer provides services that support development of the UUT test requirements, test development strategies, automatic test program generation, and diagnostic models, including the use of artificial intelligence and expert system interfaces. The Test Strategy implementation also executes the developed test programs with sequencing-based application-dependent test strategies.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

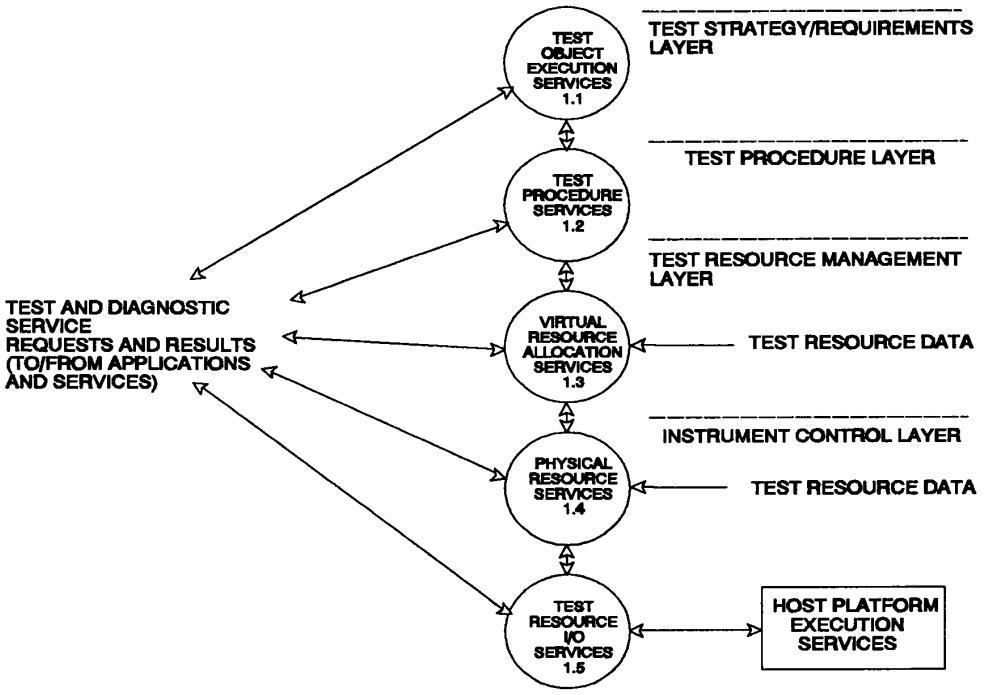


Figure 6—ABBET test and diagnostic services

The ATLAS-level test procedure layer provides services for generating and executing UUT-directed, signal-oriented test programs that are independent of any particular ATE. It also specifies standard constructs for developing encapsulated test objects that can be executed by an ABBET test strategy implementation.

The test resource management layer provides the services required to generate instrument control signals necessary for implementing a test procedure on a specific ATE. These services include mapping virtual resources to the real resources, control of real resources, performing signal routing between test resources and the UUT, access to non-UUT signal-oriented test functions, and maintaining/reporting ATE status.

The instrument control layer standardizes the software interface for various command and communications protocols used by programmable instruments. The services provide standard support for instrumentation buses and instrument communication protocols.

4.3.1 ABBET conforming implementations

ABBET conforming implementations may support one or more of these standardized interfaces.

Figure 7 illustrates an ATE test executive implementation that conforms to ABBET ALTPI and lower-level interfaces. The test program is written with UUT-directed, signal-oriented commands by calling the test control procedures defined in the ABBET CAP (IEEE Std 1226.1-1993 Common Ada Packages) and ALTPI (Ada-based ATLAS-level Test Procedure Interface) component standards.

The test executive (lighter shaded boxes) implements the bodies of all the ABBET-standardized Ada package specifications (darker shaded boxes). The bodies of the ALTPI packages control the test instruments/switches by calls to the instrument drivers implemented in conformance with the ABBET P1226.4 Test Resource Interface (TRI).

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

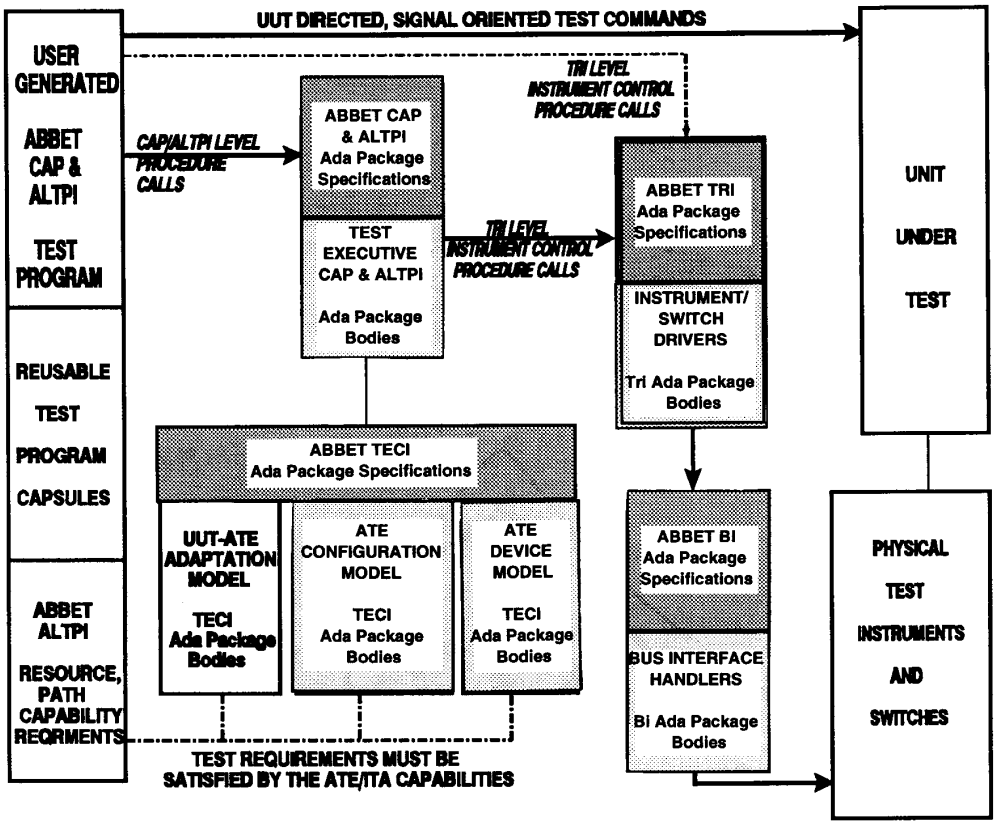


Figure 7—Layered Implementation of ABBET ALTPI and lower-level interfaces

The internal processing relates the virtual instruments and UUT connections to the physical instruments/paths based on the system configuration files that conform to the ABBET P1226.3 Test Equipment Configuration Interface (TECI). The ATE configuration and device models describe the ATE system and the UUT-ATE adaptation model describes the interface adapter.

The instrument drivers communicate with the test instruments via the bus interface handlers that conform to the ABBET P1226.5 Bus Interface (BI) component standard.

When necessary, the test program can directly call the TRI conforming instrument drivers for any special purpose test functions not available at the ATLAS level. To determine the correct physical resources the test program must access the test system configuration information via procedures that conform to the ABBET TECI standard.

Although not specifically indicated, the Test Operator and File I/O functions (defined in IEEE Std 1226.1-1993) are included in the implementation by providing their package specifications and corresponding implementation bodies in the ATLAS-level test procedure interface implementation.

4.4 Relationship to reference documents

The following reference documents (listed in clause 2) provide the key requirements for the Ada constructs that have been specified in this standard:

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

a) ISO/ANSI/MIL-STD-1815A-1983, Ada Programming Language
b) IEEE Std 716-1989, C/ATLAS Test Language
c) ARINC Spec 626-1-1989 ATLAS Language

The Ada programming language provides the basic language components used in defining this standard. All the specified ALTPI constructs can be compiled by any validated Ada compiler. All standard Ada language constructs and services must be available in any implementation that conforms to this standard.

This standard specifies Ada constructs that provide all the services of the C/ATLAS language defined in IEEE Std 716-1989. Those C/ATLAS programming features that are directly available in Ada are to be implemented with the standard Ada vocabulary. All other ATLAS test functions are provided by Ada data and subprogram specifications whose names are similar, and whose functions are semantically identical, to the corresponding C/ATLAS statements. If there are any conflicts in terminology or semantics between Ada and C/ATLAS then Ada has precedence.

Capabilities specified in ARINC Spec 626-1-1989 that are not available in C/ATLAS were also considered for incorporation in this standard. This was especially true where an approach is optional in IEEE Std 716-1989 and mandatory in ARINC Spec 626-1-1989. Incorporating the ARINC Spec 626-1-1989 approach would provide the maximum capability without conflicting with IEEE Std 716-1989. If there are any conflicts in terminology or testing approach between IEEE Std 716-1989 C/ATLAS and ARINC Spec 626-1-1989 then C/ATLAS has precedence.

The intent is to incorporate the enhancements to the Ada and C/ATLAS languages now being defined in revisions to ISO/ANSI/MIL-STD-1815A-1983 and IEEE Std 716-1989 standards when this standard is upgraded to a full-use standard after the two year trial-use period.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

5. Specification

This clause specifies the test execution and control services that shall be provided by implementations that conform to this standard. These services consist of the following:

a) General programming support that shall be provided by the standard Ada language constructs defined in ISO/ANSI/MIL-STD-1815A-1983.

b) Common physical and mathematical data types and operator/file input/output functions needed to support testing applications that shall be provided by implementations of the standardized set of common ABBET Ada packages defined in IEEE Std 1226.1-1993.

c) UUT-directed, signal-oriented C/ATLAS test constructs that are equivalent to those defined in IEEE Std 716-1989 that shall be provided by implementations of the Ada package specifications specified in clause 7.

These provide the ability to develop test programs in Ada that are independent of the ATE on which they are to be executed.

There is also a tailorable package that describes the nouns, modifiers, and connections that are supported by a particular ATE system. This package shall be generated in accordance with the rules established in clause 6.

The Ada packages in clause 7 are presented in dependency order, i.e., packages specified in earlier subclauses do not depend on any packages specified in later subclauses. The Ada packages defined in IEEE Std 1226.1-1993 and in this standard may be compiled in the order specified in the beginning of clause 7 of this standard.

Clause 5 is organized by the functions of the specified constructs. It specifies the structure of ALTPI test programs, general data processing and input/output constructs, test resource object declarations, signal-oriented procedures, the lower-level field/subfield data definitions, and definitions of the ATLAS nouns and their modifiers. Note that the field and noun/modifier definitions are thus presented in reverse dependency order. The index can be used to help cross-reference the Ada specifications in clause 7 and in clause 5.

5.1 Test program structure

5.1.1 Program components

A test program implemented in accordance with this standard consists of an Ada program that uses the standardized Ada packages defined in IEEE Std 1226.1-1993 and in clause 7 of this standard to perform all test operations.

The test program includes a set of instantiations of the ALTPI standardized Ada-generic packages to define the resources and paths required to perform the test, and an Ada "main procedure" that controls test operation. Stand alone test program components can also be encapsulated as ALTPI test capsules and executed from a central execution processor.

Annex A provides examples of complete test programs implemented in accordance with this standard.

5.1.2 Test labeling

Procedures for labeling ALTPI test programs and sections of code with meaningful character strings are defined in the ATLAS_CONTROL package (see 7.42, pg. 357).

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

The TPS is required to provide a name for the test program to the ALTPI (test executive) implementation by calling the PROGRAM_NAME procedure.

```
procedure PROGRAM_NAME (NAME : in STRING);
```

The TPS may optionally name tests within the test program using the BEGIN_TEST and END_TEST procedures:

```
procedure BEGIN_TEST (NAME : in STRING);

procedure END_TEST;
```

and label test sections using the TEST_LABEL procedure:

```
procedure TEST_LABEL (ID : in STRING);
```

The implementation may use these labels to provide the operator with on-line status information to monitor test operation, or record them for off-line test result analysis.

5.1.3 Test encapsulation

Independent tests, where the test conditions are completely stated and are not dependent on any previous tests, may be grouped into separate encapsulated test objects to allow flexibility in running or rerunning these tests in any order.

Test encapsulation is defined completely by the TPS. Each stand alone test capsule shall assume that no signals have been attached to the UUT and no signal-oriented statements have been executed before the start of its execution. The ABBET_TEST_CAPSULE generic package (see below and 7.41, pg. 356) provides the mechanism for encapsulating ALTPI test objects.

```
generic

  TEST_NAME           : STRING;
    -- Name identification of encapsulated test

  MAX_EXECUTION_COUNT : NATURAL_NUMBER := 1;
    -- Maximum loop count for EXECUTE, 0 = can loop indefinitely

  MAX_EXECUTION_TIME  : SECOND := 0.0;
    -- Maximum time for execution of this test

  CONCURRENT          : BOOLEAN := FALSE;
    -- Indicates whether this test can run concurrently with others


  with function TEST_EXECUTABLE return BOOLEAN;
    -- Generic formal function which checks for executability of test

  with procedure SETUP_TEST;
    -- Generic formal procedure which performs test set up

  with procedure EXECUTE_TEST;
    -- Generic formal procedure which executes test

  with procedure TEAR_DOWN_TEST;
    -- Generic formal procedure which tears down test set up

package ABBET_TEST_CAPSULE is

  function EXECUTABLE return BOOLEAN;
    -- Function that checks encapsulated test executability

  procedure SETUP;
    -- Procedure that executes encapsulated test setup

  procedure EXECUTE;
    -- Procedure that executes encapsulated test
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
  procedure TEAR_DOWN;
    -- Procedure that executes encapsulated test tear down

end ABBET_TEST_CAPSULE;
```

The TEST_NAME generic parameter identifies the encapsulated test. The MAX_EXECUTION_COUNT generic parameter gives the number of times the EXECUTE portion of the test object can be called in a loop. The MAX_EXECUTION_TIME generic parameter gives the maximum time this object will take to execute one loop; a (default) value of 0.0 indicates that no maximum time was specified. The CONCURRENT generic parameter indicates whether this test can be run concurrently with other test objects (e.g., there are no critical timing requirements to prevent it from running concurrently with another test).

The TEST_CAPSULE package also provides for the specification of generic formal subprograms for operating on the encapsulated test object. The TEST_EXECUTABLE function checks the current executability of the test, the SETUP_TEST procedure sets up the test, the EXECUTE_TEST procedure actually runs the test, and the TEAR_DOWN_TEST procedure tears down the test set up.

The test capsule and its formal subprograms are defined by the test program. The ATE system implementation (e.g., a test strategy processor) can execute the test via calls to the SETUP, EXECUTE, and TEAR_DOWN procedures. It can first check that the encapsulated test is executable by calling the EXECUTABLE function. The specific actions performed by these procedures/functions are application-dependent and may include UUT and ATE resource operations (e.g., the EXECUTABLE function may check UUT status and CAPTURE the required resources).

Example:

To create a test capsule (TEST_1_CAPSULE) the user instantiates:

```
package TEST_1_CAPSULE is new ABBET_TEST_CAPSULE
          (TEST_NAME           => "TEST_1 EXAMPLE CAPSULE",
           MAX_EXECUTION_COUNT => 1,
           MAX_EXECUTION_TIME  => 60.0,
           CONCURRENT          => FALSE,
           TEST_EXECUTABLE     => TEST_1_EXECUTABLE,
           SETUP_TEST          => TEST_1_SETUP,
           EXECUTE_TEST        => TEST_1_EXECUTE,
           TEAR_DOWN_TEST      => TEST_1_TEAR_DOWN);
```

where TEST_1_EXECUTABLE, TEST_1_SETUP, TEST_1_EXECUTE, and TEST_1_TEAR_DOWN are the user-defined functions/procedures below.

```
function TEST_1_EXECUTABLE return BOOLEAN is
  RESOURCES_CAPTURED : BOOLEAN;
begin
  -- See if we can get the resources
  TEST_GROUP_1.CAPTURE (CAPTURED => RESOURCES_CAPTURED,
                        TIMEOUT  => DONT_WAIT);
  -- Check the returned boolean
  if RESOURCES_CAPTURED then
    return TRUE;
  else
    -- Can't capture resources at this time
    return FALSE;
  end if;
end TEST_1_EXECUTABLE;

procedure TEST_1_SETUP is
begin
  -- Apply power to UUT
  PS2.APPLY
        (CONTROL  => (1=>VOLTAGE(28.0*V)),
         LIMIT_TO => (1=>CURRENT(MAX,0.1*A)),
         CNX      => UUT_POWER);
end TEST_1_SETUP;
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
procedure TEST_1_EXECUTE is
begin
  PSPUL.APPLY
          (CONTROL => (PRF(1.0*MEGA*HZ),
                       VOLTAGE_P(5.0*V),
                       DC_OFFSET_VOLTAGE(0.5*V),
                       DUTY_CYCLE(PC(50.0))),
           CNX     => TEST_1_PDC);
end TEST_1_EXECUTE;

procedure TEST_1_TEAR_DOWN is
begin
  REMOVE_ALL;
end TEST_1_TEAR_DOWN;
```

The program can be executed by:

```
if TEST_1_CAPSULE.EXECUTABLE then
   TEST_1_CAPSULE.SETUP;
   TEST_1_CAPSULE.EXECUTE;
   TEST_1_CAPSULE.TEAR_DOWN;
end if;
```

5.1.4 Multitasking support

ABBET supports development of multitasking test programs with the use of Ada tasks. To provide proper test controls, each independent task is identified with a THREAD_ID parameter that is managed by the test system implementation. The system status flag functions (GO, NOGO, HI, LO, MAX_TIME) also have a thread identifier to enable managing a separate set of flags for each multitasking thread.

5.1.5 Required resource instantiation

The test resources required to execute the test program are specified by instantiating the ALTPI generic resource packages for the required resource objects. This includes the test resource capability, path connection resource, event monitor, and virtual resource objects. The set of resources required to run a particular test are specified by instantiating the VIRTUAL_RESOURCE_GROUP generic with its four generic parameters.

The GROUP_IDS generic formal parameter is an array of VIRTUAL_RESOURCE_IDENTIFIER that contains the RESOURCE_NUMBER of the resource capability, any EVENT_IDENTIFIER_NUMBER for strobing or synchronizing events, and the PATH_SETS_NUMBER for the connections made to the UUT.

The REQUIRED_PATH_SETS generic formal parameter allows the specification of a subset of all the possible path sets associated with each of the virtual resources in the virtual resource group. This generic formal array is created with the PATHS, ALL_PATHS, or RESOURCE function in the particular virtual resource object. If no REQUIRED_PATH_SETS are supplied in the VIRTUAL_RESOURCE_GROUP instantiation then the default that all paths sets for the GROUP_IDS resources are required is implied.

The CONCURRENT_RESOURCES generic formal parameter allows the specification of multiple sets of concurrently required resources/path sets. This array is created by using the CONCURRENT function in the RESOURCE_DESCRIPTION package along with the PATHS, ALL_PATHS, or RESOURCE function in the particular virtual resource object. This allows the capture processing to verify the availability of required resources (source, sensor, load), event monitors, and paths to the UUT. If no CONCURRENT_RESOURCES are supplied in the VIRTUAL_RESOURCE_GROUP instantiation then the default that none of GROUP_IDS resources are required to be available simultaneously is implied.

The provided constructs allow the test programmer to capture the use of resources for a particular multiprocessing thread, and then release them when they are no longer required.

```
generic
  GROUP_NAME : STRING;
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
    -- Name assigned to this group of virtual resource objects

  GROUP_IDS  : VRG_IDENTIFIERS;
    -- Virtual Resource objects that are members of this group

  REQUIRED_PATH_SETS   : VRG_REQUIRED_RESOURCE_ARRAY := ALL_PATH_SETS_REQUIRED;
    -- Virtual Resource required path sets

  CONCURRENT_RESOURCES : VRG_CONCURRENT_RESOURCES_ARRAY
                           := NO_CONCURRENT_RESOURCES;
    -- Virtual Resource/path sets required to be concurrent

package VIRTUAL_RESOURCE_GROUP is

   procedure CAPTURE
               (CAPTURED :    out BOOLEAN;
                TIMEOUT  : in     SECOND    := DONT_WAIT;
                THREAD   : in     THREAD_ID := MAIN_THREAD);
     -- Attempts to capture all members of the group within the timeout.
     -- Flag CAPTURED indicates whether the capture occurred.
     -- Capture includes allocation of virtual resource paths and check for
     -- concurrency capability of the specified resources/path sets

   procedure RELEASE
               (IDS : in VRG_IDENTIFIERS := GROUP_IDS);
     -- Releases the members in IDS for use


end VIRTUAL_RESOURCE_GROUP;
```

Instantiation of the VIRTUAL_RESOURCE_GROUP generic results in an object with two state manipulation procedures—CAPTURE and RELEASE. The CAPTURE procedure has a THREAD identifier parameter to associate the virtual resource group with a particular processing thread.

The CAPTURE procedure shall attempt to capture all the specified resources and paths and return a Boolean (CAPTURED) indication of whether it was able to capture all the resources. If it only managed to capture some of the resources during the TIMEOUT period, it shall release them before returning. A TIMEOUT period of zero (DONT_WAIT) indicates that a single attempt shall be made to capture all the resources. Checking for the availability of resources includes determining their ability to operate concurrently with its set of required CONCURRENT_RESOURCES.

The RELEASE procedure will release any or all (default) of the previously captured resources for use by other processing threads. The Ids input to this procedure must be a member of the GROUP_IDS generic parameter. If a resource ID that is not part of the instantiated group is input to the RELEASE procedure, an error shall be detected and handled by the implementation in an implementation-defined manner. If an attempt is made to release an already released resource, no error shall be produced but the implementation may make note of it.

NOTE—In general, a CAPTURE call with a zero DONT_WAIT timeout is useful for a multi-tasking test controller that wants to determine if a test can be run at a particular time. If the test cannot be run, it would execute other tests first and then try to execute the test again later. A CAPTURE call with a non-zero (or WAIT_FOREVER) timeout is useful for specifying a test program that will keep trying to capture the resources until the TIMEOUT period expires (or until it is aborted).

Example:

The user can instantiate a resource TEST_GROUP_1 with:

```
  package TEST_GROUP_1 is new VIRTUAL_RESOURCE_GROUP
            (GROUP_NAME           => "Test 1 Example",
             GROUP_IDS            => (RES_2WIRE.ID,RES_4WIRE.ID,VAC.ID,
                                      PS2.ID,VDC.ID,PSPUL.ID),
             REQUIRED_PATH_SETS   => ALL_PATH_SETS_REQUIRED,
             CONCURRENT_RESOURCES => (1 => CONCURRENT
                                             ((PS2.PATHS(UUT_POWER),
                                               PSPUL.PATHS(REAL_TEST_2_PDC)))));
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

The resources can then be captured with:

```
TEST_GROUP_1.CAPTURE
               (CAPTURED => RESOURCES_CAPTURED,
                TIMEOUT  => DONT_WAIT);
```

The resources can be released with:

```
TEST_GROUP_1.RELEASE;
```

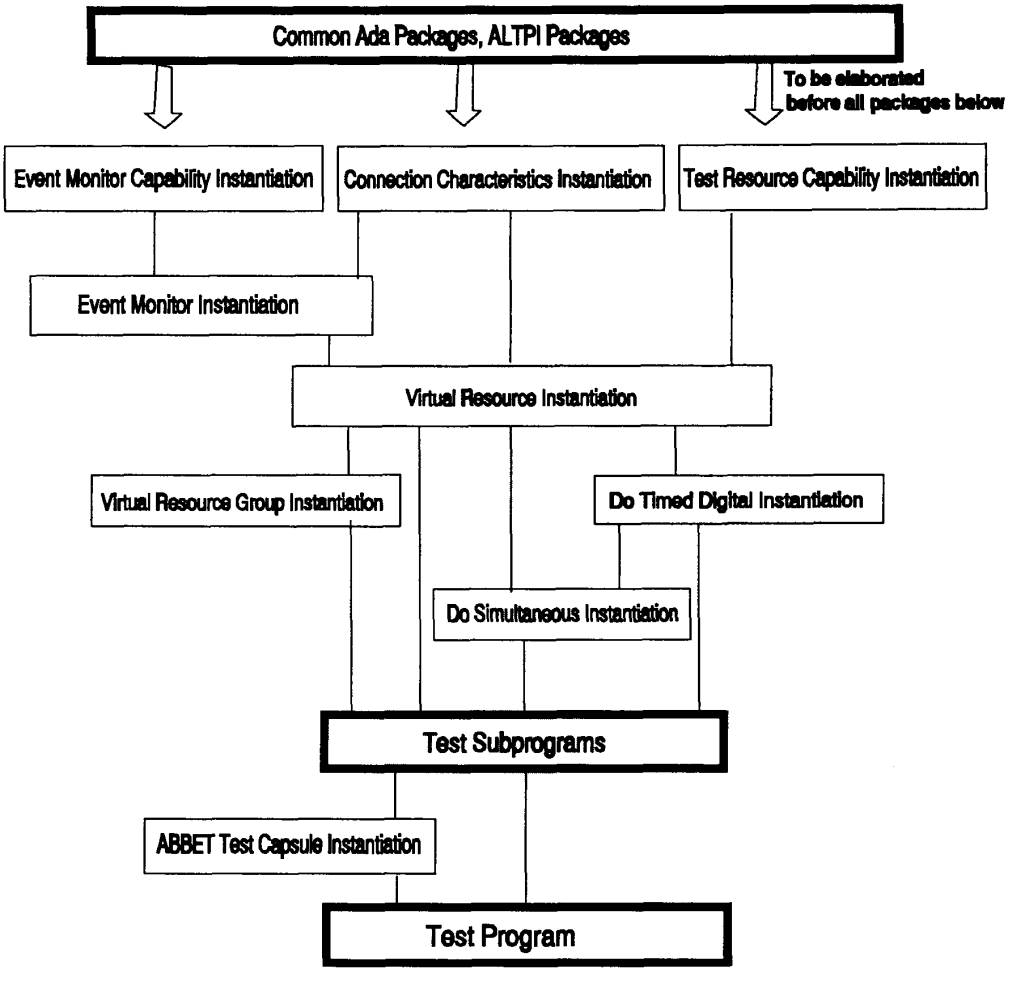


Figure 8—ALTPI package elaboration order diagram

5.1.6 Test program elaboration

As with any Ada program, the elaboration order is important in the sense that certain items must be elaborated before later items. Generally, the Ada compiler automatically performs the elaboration in the correct order based on its internal processing algorithms.

Figure 8 provides a top-down description of the necessary elaboration order for instantiation of the various classes of Ada packages in an ALTPI test program. The lines connecting the boxes indicate an elaboration

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

dependency. The top level has the basic ABBET Common Ada packages and the ALTPI data type descriptions that are generally needed by all the packages below them.

None of the standard ALTPI package specifications require the elaboration of any other package body for their own elaboration. The instantiation of ALTPI generics by the test program does depend on earlier system allocation of identifiers for resources and events. The ID functions, that return the identifier allocated by the system, automatically prevent the possibility of accessing the data before the initialization body is elaborated.

If the Ada compiler generates a "program error" because its elaboration algorithm does not elaborate all the packages in the correct order (e.g., this may happen with some compilers if a generic is instantiated in a package specification), then the user can guide the compiler with the `pragma elaborate` directive.

The list of Ada packages in each box in Figure 8 are:

Common Ada Packages
- ABBET_NUMERIC_PRECISION
- ABBET_TYPES
- ABBET_SI_UNITS

ALTPI Packages
- ATLAS_TYPES
- NOUN_MODIFIER_TYPES
- SIGNAL_DESCRIPTION
- RESOURCE_DESCRIPTION
- EVENT_DESCRIPTION
- DIGITAL_DESCRIPTION
- PATH_DESCRIPTION
- SYNCHRONIZATION_TYPES
- ATLAS_NOUNS_MODIFIERS_AND_CONNECTIONS
- DATA_BUS_DESCRIPTION

Event Monitor Capability Instantiation
- ANALOG_SIGNAL_MONITOR_CAPABILITY
- ANALOG_TIME_MONITOR_CAPABILITY
- MANUAL_INTERVENTION_CAPABILITY
- DIGITAL_EVENT_MONITOR_CAPABILITY

Test Resource Capability Instantiation
- ANALOG_SENSOR_CAPABILITY
- ANALOG_SOURCE_CAPABILITY
- SIMPLE_LOAD_CAPABILITY
- TIME_INTERVAL_SENSOR_CAPABILITY
- TIMER_SOURCE_CAPABILITY
- DIGITAL_SENSOR_CAPABILITY
- DIGITAL_SOURCE_CAPABILITY
- PRIMARY_DATA_BUS
- ALTERNATE_DATA_BUS
- EXCHANGE_DEFINITION

Connection Characteristics Instantiation
- PATH_CONNECTIONS
- PATH_SETS_DEFINITION

Event Monitor Instantiation
- ANALOG_SIGNAL_MONITOR

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

ANALOG_TIME_MONITOR
MANUAL_INTERVENTION
DIGITAL_EVENT_MONITOR

Virtual Resource Instantiation
ANALOG_SENSOR
ANALOG_SOURCE
SIMPLE_LOAD
TIME_INTERVAL_SENSOR
TIMER_SOURCE
DIGITAL_SENSOR
DIGITAL_SOURCE
DO_DATA_BUS_EXCHANGE

Virtual Resource Group Instantiation
VIRTUAL_RESOURCE_GROUP

Do-Timed Digital Instantiation
STIM_RATE_ONLY
STIM_EVENT_ONLY
STIM_RATE_AND_SENSE_DELAY
STIM_RATE_AND_SENSE_EVENT
STIM_EVENT_AND_SENSE_DELAY
STIM_EVENT_AND_SENSE_EVENT
SENSE_RATE_ONLY
SENSE_EVENT_ONLY

Do-Simultaneous Instantiation
DO_SIMULTANEOUS

ABBET Test Capsule Instantiation
ABBET_TEST_CAPSULE

5.1.7 Status flag operations

The ABBET_EVALUATIONS package (see 7.39, pg.353) defines a set of overloaded COMPARE procedures to establish the state of HI, LO, GO, and NOGO flags and a set of functions to obtain their current status.

5.1.7.1 Compare procedures

The COMPARE procedures establish the state of HI, LO, GO, and NOGO flags based on the relationship between a variable identifier (VALUE) and a specified set of limit(s) (EVALUATION_FIELD) for various ABBET data types. The COMPARE STRING_1 to STRING_2 and COMPARE_FILES procedures can only check for equality and set the GO/NOGO flags.

The THREAD parameter associates the COMPARE execution with a particular thread. The FORM parameter allows a free-form string to be input for implementation-defined purposes (test labeling, data logging, comparison data units, etc.).

```
procedure COMPARE
            (VALUE  : in REAL;
             EVAL   : in EVALUATION_FIELD;
             THREAD : in THREAD_ID        := MAIN_THREAD;
             FORM   : in STRING           := "");

procedure COMPARE
            (VALUE  : in REAL_OUTPUT_VALUE;
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
                EVAL   : in EVALUATION_FIELD;
                THREAD : in THREAD_ID          := MAIN_THREAD;
                FORM   : in STRING             := "");

  procedure COMPARE
              (VALUE  : in NUMBER;
               EVAL   : in EVALUATION_FIELD;
               THREAD : in THREAD_ID         := MAIN_THREAD;
               FORM   : in STRING            := "");

  procedure COMPARE
              (VALUE  : in NUMBER_OUTPUT_VALUE;
               EVAL   : in EVALUATION_FIELD;
               THREAD : in THREAD_ID            := MAIN_THREAD;
               FORM   : in STRING               := "");

  procedure COMPARE
              (VALUE  : in DIGITAL;
               EVAL   : in EVALUATION_FIELD;
               THREAD : in THREAD_ID         := MAIN_THREAD;
               FORM   : in STRING            := "");

  procedure COMPARE
              (VALUE  : in DIGITAL_OUTPUT_VALUE;
               EVAL   : in EVALUATION_FIELD;
               THREAD : in THREAD_ID            := MAIN_THREAD;
               FORM   : in STRING               := "");

  procedure COMPARE
              (STRING_1 : in STRING;
               STRING_2 : in STRING;
               THREAD   : in THREAD_ID := MAIN_THREAD;
               FORM     : in STRING    := "");

  procedure COMPARE_FILES
              (FILE_1 : in STRING;
               FILE_2 : in STRING;
               THREAD : in THREAD_ID := MAIN_THREAD;
               FORM   : in STRING    := "");
```

The COMPARE procedure shall be confined to assist in setting the status (HI, LO, GO, and NOGO) flags needed to display or log failures. These procedures shall not be used to directly control program execution.

Example:

An example COMPARE procedure call is:

```
COMPARE (OUT_23, LL_UL(UL=>23.75*V,LL=>22.25*V));
```

5.1.7.2 Operator Input of GO, NOGO condition

A procedure is defined to input a GO, NOGO condition from the test operator and accordingly set the system status flags for the appropriate processing thread. It is up to the implementation to decide how the operator communicates with the test system (e.g., via control/display or hardware switch setting).

```
  procedure INPUT_OPERATOR_GO_NOGO
              (THREAD : in THREAD_ID := MAIN_THREAD);
```

5.1.7.3 STATUS (GO, NOGO, HI, LO, MAX_TIME) Functions

ABBET provides for global GO, NOGO, HI, LO, and MAX_TIME condition status identifiers similar to those provided by C/ATLAS. The BOOLEAN value of any of these condition identifiers may be referenced by using the corresponding status function defined in the ABBET ATLAS package with the same name as the desired status condition. The THREAD parameter associates the call of the function to a particular multi-processing thread.

```
  function GO       (THREAD : in THREAD_ID := MAIN_THREAD) return BOOLEAN;
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
function NOGO     (THREAD : in THREAD_ID := MAIN_THREAD) return BOOLEAN;
function HI       (THREAD : in THREAD_ID := MAIN_THREAD) return BOOLEAN;
function LO       (THREAD : in THREAD_ID := MAIN_THREAD) return BOOLEAN;
function MAX_TIME (THREAD : in THREAD_ID := MAIN_THREAD) return BOOLEAN;
```

The initial values of these boolean functions are:

```
GO        = TRUE
NOGO      = FALSE
HI        = FALSE
LO        = FALSE
MAX_TIME  = FALSE
```

5.1.8 Test completion

ALTPI defines some global control procedures that can be used to complete processing of a single thread or the complete test program.

The REMOVE_ALL calls act on all resources used in a processing thread or all resources used in a test program (depending on whether a THREAD parameter is included in the call).

```
procedure REMOVE_ALL (THREAD : in THREAD_ID);

procedure REMOVE_ALL;
```

The FINISH call terminates all testing, returns the test equipment to a quiescent state, and returns program control to the operator/operating system.

```
procedure FINISH;
```

The REMOVE call for a specific virtual resource is described in 5.9.

5.2 Data processing

ABBET data processing capabilities are provided by the Ada language constructs defined in the ISO/ANSI/MIL-STD 1815A-1983 and by some specialized higher-level Ada procedures/functions defined in the ABBET component standards. These capabilities include the ability to perform calculations, compare test values with specified limits, and control test operations based on the status expressed in the GO, NOGO, HI, LO, and MAX_TIME flags. These are described in more detail in the following subclauses.

5.2.1 Authorized characters

The character set authorized by the Ada language (see ISO/ANSI/MIL-STD 1815A-1983 paragraph 2.1) are authorized for use in ABBET programming statements.

5.2.2 Number representations

The Ada language conventions (see ISO/ANSI/MIL-STD 1815A-1983 paragraph 2.4) shall be used to express literal values in ABBET for integer, fixed point, floating point, binary, octal, and hexadecimal numbers.

The ABBET_TYPES package defined in IEEE Std 1226.1-1983 specifies the consistent set of Ada data types that shall be used by all the ABBET conforming implementations. It includes definitions of the REAL and NUMBER data subtypes of BASE_FLOAT and BASE_INTEGER as well as binary data representations and digital bit arrays.

The ABBET_NUMERIC_PRECISION package defined in IEEE Std 1226.1-1993 specifies the numeric precision for BASE_FLOAT as range 15 digits and for BASE_INTEGER as range -2**31..2**31-1. The precision for the REAL and NUMBER data types are derived from the BASE_FLOAT and

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

BASE_INTEGER types. An ABBET implementation may define lower data precision values only if it is restricted by hardware limitations and it must clearly identify the limitation.

5.2.3 Scope of identifiers

The scope of identifiers is as specified by the Ada language. Only identifiers defined in the package specifications may be externally accessed. A user may access these externally defined identifiers by `with`ing the package in which they are defined.

5.2.4 Program control

Test program procedural controls are provided by the Ada language constructs listed below with the ISO/ANSI/MIL-STD 1815A-1983 sections or paragraphs in which they are defined.

a) **if, elsif, case, loop, for, while, exit, return, and goto** (Section 5).

b) **Subprogram Calls** (Paragraph 6.4)

c) **Task Activation/Termination, Delay Statement, Duration, Time, Priorities, and Abort** (Paragraphs 9.3–9.10)

d) **Exceptions** (Section 11)

5.2.5 Computational operations

The ABBET computational operations are provided by the Ada language constructs listed below with the ISO/ANSI/MIL-STD 1815A-1983 sections or paragraphs in which they are defined.

a) **Expressions and Operators** (Paragraphs 4.4–4.10)

b) **Assignment Statement** (Paragraph 5.2)

5.2.6 Mathematical operations

The ABBET_ELEMENTARY_FUNCTIONS package defined in IEEE Std 1226.1-1993 specifies the following set of elementary mathematical functions:

```
SQRT, LOG, EXP, **, SIN, COS, TAN, COT, ARCSIN, ARCCOS, ARCTAN,
ARCCOT, SINH, COSH, TANH, ARCSINH, ARCCOSH, ARCTANH, ARCCOTH
```

that operate on the ABBET standardized REAL data type.

5.2.7 Digital data operations

The ABBET_TYPES package defined in IEEE Std 1226.1-1993 specifies the following set of functions:

```
"&", "and", "or", "xor", "not", SHIFT_LEFT, SHIFT_RIGHT,
ROTATE_LEFT, ROTATE_RIGHT, A_SHIFT_RIGHT
```

that perform digital data operations on the ABBET standardized digital data types.

This package also specifies the following set of allocation and conversion functions:

```
DIG, DBA, NUM, ALLOCATE, DEALLOCATE, SET_ACCESS, VALUE, DIG_BIN,
DIG_OCT, DIG_HEX, DBA_BIN, DBA_OCT, DBA_HEX
```

that operate on the ABBET standard data types.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

5.2.8 ABBET standard physical units

The ABBET_SI_UNITS defined in IEEE Std 1226.1-1993 shall be used to quantify all physical quantities, as indicated by the measured/applied noun modifiers, in all ABBET implementations. The STANDARD_NOUNS_MODIFIERS_AND_CONNECTIONS package (see 7.9, pg. 292) specifies (by the comments included with each noun modifier definition) which ABBET SI unit shall be used to quantify each noun modifier.

The ABBET_SI_UNITS include the seven basic SI units:

METER, KILOGRAM, SECOND, AMPERE, KELVIN, MOLE, and CANDELA;

the two supplementary SI units:

RADIAN, STERADIAN;

and the set of derived SI units.

Decimal multiples of these units shall be expressed as powers of 10 in the measured values. Prefix multipliers (milli, centi, kilo, etc.) shall not be used in internal calculations or in passing data across ABBET interfaces to avoid errors in calculations and mixing of units. Conversion constants for the standard set of SI prefixes are provided in this package for conversions to the units measured/displayed by external input/output devices.

A related ABBET_OTHER_UNITS package defines conversion constants to convert between SI units and other (English) units that are allowed to be used for external input/output to instruments and operator displays in ABBET-conforming implementations. These other units have traditionally been used in test applications and are authorized by such standards as IEEE Std 716-1989.

The standardized ABBET_SI_UNITS are defined as subtypes of the REAL and NUMBER data types defined in ABBET_TYPES. This allows efficient ABBET-conforming implementations while retaining standardized names and definitions of all the units. A dimensional analysis of the test program source code can also be performed with a preprocessor tool that checks defined dimensions for these units and the operations on them.

Defining the SI units as `new` derived Ada data types would enable any Ada compiler to perform automatic strong data type checking but it would require definitions of the operators for all allowed operations, including all intermediate calculation results, between the units. This results in large compilation, link, memory, and run-time overheads that are impractical with today's Ada compilers.

An optional version of the ABBET_SI_UNITS package that defines all these units as subtypes of discriminated Ada records that support dimensional analysis checking is provided in annex C of IEEE Std 1226.1-1993. The records are discriminated by an array of integers that define the units in terms of powers of the seven basic SI units. A dimensional analysis of the results of all computations can be performed at run-time or by a preprocessor by comparing the powers of the input quantities and computed results. Definitions of the mathematical operator functions that automatically compute the powers of the results are also provided. A user may compile the test software with the dimensional analysis form during software development for the run-time checking and then recompile (without any source code changes) with the standard ABBET_SI_UNITS package for efficient program run-time operation.

5.2.9 Physical data conversions

IEEE Std 1226.1-1993 defines an ABBET_UNITS_CONVERSIONS package that specifies a set of useful generic conversion functions to convert between the standard ABBET_SI_UNITS and the other physical units that are allowed in ABBET (specified in the ABBET_OTHER_UNITS package) for interfacing with

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

external devices. The user instantiates these conversion functions with the appropriate conversion constants defined in the ABBET_OTHER_UNITS package.

5.3 Input/output

Standardized procedures for input and output of test data from/to computer files and operator controls/displays are specified in other **IEEE ABBET** component standards as described below.

5.3.1 File input/output

The ABBET_TEXT_IO, ABBET_DIRECT_IO, and ABBET_SEQUENTIAL_IO packages specified in IEEE Std 1226.1-1993 define standardized procedures/functions for input and output of the ABBET standardized data types from/to computer files. They provide input/output capabilities that are similar to those provided by the corresponding packages defined in ISO/ANSI/MIL-STD-1815A-1983.

IEEE Std 1226-1993 specifies that standardized test data logging formats are to be defined in a Maintenance Data Collection and Retrieval Interface for ABBET component standard.

5.3.2 Operator input/output

The ABBET_TEXT_IO and OPERATOR_IO packages specified in IEEE Std 1226.1-1993 define standard procedures/functions for input and output of test data from/to the test operator.

IEEE Std 1226-1993 specifies that standardized data exchange formats for operator input/output are to be defined in an **IEEE ABBET** User Interface component standard.

5.4 Connection, path requirements

In developing a TPS to test a UUT, the user needs to define the UUT's external connections (terminals, pins, and paths) and the type of signals to be applied/measured at these connections.

5.4.1 Pin descriptors

Pin Descriptors describe the terminal types for connecting virtual resources to perform required testing. The PIN_DESCRIPTOR_TYPE indicates whether it is a normal or reference pin. The PIN_DESCRIPTOR_NUMBER identifies the terminal type. For the standard ATLAS_PIN_DESCRIPTORS (see 5.4.1.1) the PIN_DESCRIPTOR_NUMBER is effectively equated to the position of the enumerated standard pin descriptor.

```
type PIN_DESCRIPTOR_TYPE is (NORMAL,REFERENCE);

type PIN_DESCRIPTOR_NUMBER is range -1..INTEGER'LAST;

NULL_PIN_DESCRIPTOR_NUMBER : constant PIN_DESCRIPTOR_NUMBER := -1;
```

PIN_DESCRIPTOR is a private type that identifies the pin descriptor in all the ALTPI data structures. There can be normal and reference pin descriptors.

```
type PIN_DESCRIPTOR
       (PIN_TYPE : PIN_DESCRIPTOR_TYPE := NORMAL) is private;

subtype NORMAL_PIN_DESCRIPTOR    is PIN_DESCRIPTOR(NORMAL);
subtype REFERENCE_PIN_DESCRIPTOR is PIN_DESCRIPTOR(REFERENCE);

NO_PDESC : constant PIN_DESCRIPTOR;
```

Functions for creating the private PIN_DESCRIPTOR data types from pin descriptor identifying numbers are:

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
generic
  PIN_DESC_NUM : PIN_DESCRIPTOR_NUMBER;
function PIN_DESCRIPTOR_NORMAL
          return NORMAL_PIN_DESCRIPTOR;

generic
  PIN_DESC_NUM : PIN_DESCRIPTOR_NUMBER;
function PIN_DESCRIPTOR_REFERENCE
          return REFERENCE_PIN_DESCRIPTOR;
```

The private part definition for PIN_DESCRIPTOR is:

```
private

  type PIN_DESCRIPTOR
        (PIN_TYPE : PIN_DESCRIPTOR_TYPE := NORMAL) is
    record
      PDESC : PIN_DESCRIPTOR_NUMBER;
    end record;

  NO_PDESC : constant PIN_DESCRIPTOR := (NORMAL,NULL_PIN_DESCRIPTOR_NUMBER);
```

A function to provide access to the contents of the private PIN_DESCRIPTOR is:

```
  function GET_PIN_DESCRIPTOR_NUMBER
            (PIN_DESC : in PIN_DESCRIPTOR)
            return PIN_DESCRIPTOR_NUMBER;
```

5.4.1.1 Standard pin descriptors

This subclause specifies the standard enumerated set of ATLAS pin descriptors for connecting virtual resources.

```
  type ATLAS_PIN_DESCRIPTOR
        is (REFERENCE_PHASE,
            HIGH,LOW,VIA,
            SCREEN,GUARD,
            A_PHASE,B_PHASE,C_PHASE,NEUTRAL,
            X,Y,Z,
            S1,S2,S3,S4,
            R1,R2,R3,R4,
            TRUE_SIGNAL,COMPL_SIGNAL,
            TO,FROM);

  START_PIN_DESCRIPTOR_EXTENSIONS : constant PIN_DESCRIPTOR_NUMBER := 256;
```

The definitions of these pin descriptors are:

a) **REFERENCE_PHASE:** The reference connection capability is used for signals that shall be defined relative to some reference signal, such as a phase-sensitive or a time-related function. The stated UUT points define the location of the reference signal. When the reference signal is one phase of a multi-phase source that is wholly internal to the test equipment, and, therefore, cannot be identified by UUT connections, it may be identified in the connection list by a reference phase identifier.

b) **HIGH:** This denotes the hot side of a two-wire circuit. The HIGH side has the same polarity as the magnitude field of the statement when measured with respect to the LOW side. HIGH may also designate the side of the circuit that is most isolated (highest impedance) from the UUT ground reference.

c) **LOW:** This denotes the cold or return side of a two-wire circuit. The LOW side is the reference for the function described in the signal statement. LOW may also designate the side of the circuit that is least isolated (lowest impedance) from the UUT ground reference. (Note that the HIGH and LOW connections conform to a different magnitude convention than do the evaluation field and

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

system status flag operations where algebraic magnitude is determined, i.e., to put dc of –8V with respect to –6V on the UUT, –8V goes with HIGH and –6V goes with LOW.)

d) **VIA:** This denotes the following:

a) A terminal for a series type connection (e.g., current flow)

b) A port for the transmission of electromagnetic signals (e.g., waveguide)

c) A port for fluid type signals (e.g., manometric, hydraulic)

In all cases, flow to the UUT is arbitrarily designated positive.

In item a) the source or sensor should be considered as being connected between the referenced UUT terminal(s) and all other already existing connections to the referenced UUT terminal(s), without causing any temporary interruption to the flow. It should be noted that a previous statement should have provided a path for the return flow.

In item b) negative power-ratio quantities (e.g., –8 dBm) may be written as a statement characteristic. However, the negative sign should not be construed as being an indication of the direction of power transfer.

e) **SCREEN:** This denotes a terminal on the UUT to which an electrostatic shield around the signal conductors is to be connected.

f) **GUARD:** This denotes a terminal on the UUT to which the guard terminal, if any, of the sensor device is to be connected in order to reduce the effect of common mode voltages between the LOW side of the UUT signal and the chassis ground of the sensor.

g) **A_PHASE, B_PHASE, C_PHASE, NEUTRAL:** Connection pins for up to three phases and neutral. These pins can be used to describe various phase configurations as shown in the following examples:

1) Single Phase - A_PHASE, B_PHASE, or C_PHASE and NEUTRAL
2) Two Phase - A_PHASE, B_PHASE, and Neutral
3) Three Phase Delta - A_PHASE, B_PHASE, and C_PHASE
4) Three Phase Wye - A_PHASE, B_PHASE, C_PHASE and NEUTRAL

h) **X, Y, Z:** The pin set for three-wire synchro-stator connections, or to designate the axis of an orthogonal coordinate frame for mechanical rotation.

i) **S1, S2, S3, S4:** Four-wire resolver stator connections pin set.

j) **R1, R2, R3, R4:** Synchro-Resolver rotor connections pin set.

k) **TO:** A pin name for non-electrical signals. It is used when only a single connection is required (irrespective of the flow to or from the UUT). It is also used when two connections are required to define the connection that carries flow to the UUT (*see:* **FROM**).

l) **FROM:** A pin name for non-electrical signals. It is used when two connections are required and defines the connection that carries flow from the UUT. When the parameter representing flow can assume positive and negative values, a positive value represents flow to the UUT through the TO connection; a negative value represents flow to the UUT through the FROM connection.

m) **TRUE_SIGNAL, COMPL_SIGNAL:** The names for the connections that carry the TRUE and COMPLEMENTARY signals for a differential digital signal. Used in place of HIGH. The

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

connections for a three wire differential digital signal are written using TRUE_SIGNAL, COMPL_SIGNAL and LOW. For a two wire signal, HIGH and LOW may be used.

Normal or reference pin descriptors can be created by declaring functions that return NORMAL_PIN_DESCRIPTOR and REFERENCE_PIN_DESCRIPTOR constants for the desired pin descriptor. A recommended practice for naming the reference function is to combine REF_ with the pin descriptor name, e.g., REF_HIGH and REF_LOW.

```
  P8 : constant PIN_DESCRIPTOR_NUMBER
                  := ATLAS_PIN_DESCRIPTOR'POS(NMX.TRUE_SIGNAL);
function TRUE_SIGNAL  is new PIN_DESCRIPTOR_NORMAL(P8);

  P9 : constant PIN_DESCRIPTOR_NUMBER
                  := ATLAS_PIN_DESCRIPTOR'POS(NMX.COMPL_SIGNAL);
function COMPL_SIGNAL is new PIN_DESCRIPTOR_NORMAL(P9);

  P10 : constant PIN_DESCRIPTOR_NUMBER := ATLAS_PIN_DESCRIPTOR'POS(NMX.HIGH);
function REF_HIGH is new PIN_DESCRIPTOR_REFERENCE(P10);

  P11 : constant PIN_DESCRIPTOR_NUMBER := ATLAS_PIN_DESCRIPTOR'POS(NMX.LOW);
function REF_LOW  is new PIN_DESCRIPTOR_REFERENCE(P11);
```

Extensions to the set of standard pin descriptors can be created by declaring constants equated to the START_PIN_DESCRIPTOR_EXTENSIONS+n values. For example:

```
function HOSE    is new PIN_DESCRIPTOR_NORMAL
                        (START_PIN_DESCRIPTOR_EXTENSIONS);
function PIPE    is new PIN_DESCRIPTOR_NORMAL
                        (START_PIN_DESCRIPTOR_EXTENSIONS+1);
function NOZZLE  is new PIN_DESCRIPTOR_NORMAL
                        (START_PIN_DESCRIPTOR_EXTENSIONS+2);
```

5.4.2 Connection identifiers

The CONNECTION_IDENTIFIER type identifies the UUT or ATE connection.

```
  type CONNECTION_NUMBER is range 0..INTEGER'LAST;

  type CONNECTION_TYPE is (UUT, ATE, REF);

  type CONNECTION_IDENTIFIER
         (C_TYPE : CONNECTION_TYPE := CONNECTION_TYPE'FIRST) is private;

  subtype UUT_CONNECTION_IDENTIFIER is CONNECTION_IDENTIFIER(UUT);
  subtype ATE_CONNECTION_IDENTIFIER is CONNECTION_IDENTIFIER(ATE);
  subtype REF_CONNECTION_IDENTIFIER is CONNECTION_IDENTIFIER(REF);
```

The set of standard connections can be extended with instantiations of the CONN_NUM generic functions that are specified below.

```
  START_CONNECTION_EXTENSIONS : constant CONNECTION_NUMBER := 1024;

  subtype CONNECTION_NUMBER_EXTENSION
            is CONNECTION_NUMBER
                 range START_CONNECTION_EXTENSIONS..CONNECTION_NUMBER'LAST;

  generic
    CONN_NUM : CONNECTION_NUMBER_EXTENSION;
  function ATE_CONNECTION_EXTENSION return ATE_CONNECTION_IDENTIFIER;

  generic
    CONN_NUM : CONNECTION_NUMBER_EXTENSION;
  function REF_CONNECTION_EXTENSION return REF_CONNECTION_IDENTIFIER;
```

The private part definition for CONNECTION_IDENTIFIER is:

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
private

  type CONNECTION_IDENTIFIER
       (C_TYPE : CONNECTION_TYPE := CONNECTION_TYPE'FIRST) is
    record
      C_NUMBER : CONNECTION_NUMBER;
    end record;

  NULL_CONNECTION : constant ATE_CONNECTION_IDENTIFIER := (ATE,0);
  ATMOSPHERE_CNX  : constant ATE_CONNECTION_IDENTIFIER := (ATE,1);
  COMMON_CNX      : constant ATE_CONNECTION_IDENTIFIER := (ATE,2);
  EARTH_CNX       : constant ATE_CONNECTION_IDENTIFIER := (ATE,3);

  PHASE_A_REF  : constant REF_CONNECTION_IDENTIFIER := (REF,1);
  PHASE_B_REF  : constant REF_CONNECTION_IDENTIFIER := (REF,2);
  PHASE_C_REF  : constant REF_CONNECTION_IDENTIFIER := (REF,3);
  PHASE_BC_REF : constant REF_CONNECTION_IDENTIFIER := (REF,4);
  PHASE_CB_REF : constant REF_CONNECTION_IDENTIFIER := (REF,5);
  PHASE_CA_REF : constant REF_CONNECTION_IDENTIFIER := (REF,6);
  PHASE_BA_REF : constant REF_CONNECTION_IDENTIFIER := (REF,7);
  PHASE_AB_REF : constant REF_CONNECTION_IDENTIFIER := (REF,8);
  PHASE_AC_REF : constant REF_CONNECTION_IDENTIFIER := (REF,9);
```

5.4.2.1 Standard ATE connection identifiers

The visible (non-private) standard ATE connection identifier specifications are:

```
  NULL_CONNECTION : constant ATE_CONNECTION_IDENTIFIER;  -- Connection num = 0
  ATMOSPHERE_CNX  : constant ATE_CONNECTION_IDENTIFIER;  -- Connection num = 1
  COMMON_CNX      : constant ATE_CONNECTION_IDENTIFIER;  -- Connection num = 2
  EARTH_CNX       : constant ATE_CONNECTION_IDENTIFIER;  -- Connection num = 3
```

where:

a) ATMOSPHERE_CNX indicates that the line is connected to the surrounding atmosphere.

b) COMMON_CNX indicates that the lead of the measurement, stimulus, or load device is to be connected to COMMON.

c) EARTH_CNX indicates that the lead of the stimulus, load, or measurement device is to be connected to EARTH.

5.4.2.2 Standard reference phase connection identifiers

The visible (non-private) standard reference phase connection identifier specifications, which can be used with the REFERENCE_PHASE pin descriptor, are:

```
  PHASE_A_REF  : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 1
  PHASE_B_REF  : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 2
  PHASE_C_REF  : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 3
  PHASE_BC_REF : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 4
  PHASE_CB_REF : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 5
  PHASE_CA_REF : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 6
  PHASE_BA_REF : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 7
  PHASE_AB_REF : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 8
  PHASE_AC_REF : constant REF_CONNECTION_IDENTIFIER;  -- Connection num = 9
```

where:

a) PHASE_A_REF, PHASE_B_REF, and PHASE_C_REF indicates that the reference voltage is between the stated phase and Neutral.

b) PHASE_BC_REF, PHASE_CB_REF, PHASE_CA_REF, PHASE_BA_REF, PHASE_AB_REF, and PHASE_AC_REF indicate the high side of the referenced voltage (first letter) and the low side (second letter).

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

5.4.2.3 Digital UUT connections

All logic signals, being ordinary analog pulses or dc levels, follow the standard analog conventions for the UUT connections except for the following special definitions for differential signals and for DATA bit and HIGH/LOW pin sequences.

5.4.2.3.1 Connections for SERIAL DATA

Two-wire SERIAL DATA connections are written using the standard HIGH/LOW pin names. Three-wire electrical differential signals use the pin descriptors TRUE, COMPL, and LOW. The bits of a SERIAL DATA signal appear at the digital interface in the sequence specified in the TYPE field for a LOGIC DATA signal, that is, either the least-significant or the most-significant bit first.

5.4.2.3.2 Connections for PARALLEL DATA

For two-wire PARALLEL DATA signals, the number of HIGH connections shall be the same as the number of bits in the parallel digital word. Two possible alternatives exist for the LOW connections:

a) One circuit return (LOW connection) may be provided for all HIGH connections.

b) A separate circuit return (LOW connection) may be provided for each HIGH connection.

Entries in the connection field are written as follows:

a) All HIGH connections are written first, in a sequence that corresponds to the left-to-right bit sequence defined in the digital value (i.e., the most-significant-bit (MSB) of the DIGITAL data is transmitted over the first pin).

b) The LOW connection(s) is (are) written next. If individual UUT signal returns are provided, they are written in the same sequence as the HIGH connections (i.e., corresponding HIGH and LOW connections represent the two sides of one electrical circuit).

Thus, for the two options above,

a)
```
Required Connections:  (HIGH, HIGH, HIGH, HIGH, LOW)
UUT Pins:              (UUT(J1_1), UUT(J1_2), UUT(J1_3), UUT(J1_4),
                       UUT(J1_5))
Bit Pattern:           BIN("1010")
Pin State:             J1_1=>1, J1_2=>0, J1_3=>1, J1_4=>0
```

b)
```
Required Connections:  (HIGH, HIGH, HIGH, HIGH, LOW, LOW, LOW, LOW)
UUT Pins:              (UUT(J1_1), UUT(J1_2), UUT(J1_3), UUT(J1_4),
                       UUT(J1_11), UUT(J1_12), UUT(J1_13), UUT(J1_14))
Bit Pattern:           BIN("0101")
Pin State:             J1_1=>0, J1_2=>1, J1_3=>0, J1_4=>1
```

5.4.2.3.3 Electrical differential signals

The input or output for UUT digital DATA signals may require transmission in an electrical differential form. As illustrated below, this differential form places a digital signal (called TRUE) on one line and its complement (called COMPL) on another line.

These connections are written as follows for a PARALLEL DATA signal:

a) Pin names are TRUE_SIGNAL, COMPL_SIGNAL, and LOW.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

b) The number of COMPL_SIGNAL connections equals the number of TRUE connections. Two options for the number of LOW connections exist:

1) One circuit return (LOW connection) may be provided for all high side connections.

2) A separate circuit return (LOW connection) may be provided for each high side connection.

c) Connections are written in the following sequence: all TRUE_SIGNAL connections, all COMPL_SIGNAL connections, all LOW connections.

```
Required Connections:   (TRUE_SIGNAL,TRUE_SIGNAL,TRUE_SIGNAL,
                         COMPL_SIGNAL,COMPL_SIGNAL,COMPL_SIGNAL, LOW)
UUT Pins:               (UUT(J1_1), UUT(J1_2), UUT(J1_3), UUT(J1_4),
                         UUT(J1_5), UUT(J1_6), UUT(J1_10))
```

Floating differential signals can be defined by omitting the LOW connection. In some cases, differential connections may be treated as two-wire HIGH/LOW circuits. However, the validity of this method shall be determined for each individual application or logic family. This complement function can be implemented within the test system either by hardware (by providing an inversion of the TRUE line) or by software (by forming the complement bit pattern and placing it on a second TRUE line).

5.4.2.4 Non-electrical connections

The input or output of non-electrical signals to or from the UUT may require the connection of pipes, mechanical linkages, etc. These may be single lines or linkages, or may be pairs of lines, one carrying signal flow to the UUT and one carrying signal flow from the UUT. The pin names for a pair of non-electrical connections are TO (for the connection that normally carries signal flow to the UUT) and FROM (for the connection that normally carries signal flow from the UUT). If multiple connections are written after the pin descriptor, all specified UUT points shall be connected together. For pipes a manifold is necessary; for mechanical or other linkages, all UUT points shall be tied together.

5.4.3 Path description

The PATH_DESCRIPTION constructs are defined below to provide the data types and PATH_CONNECTIONS and PATH_SETS_DEFINITION generic package to enable the user to specify these requirements in a standardized way. The user instantiates these generics with the characteristics of the particular path set.

Objects produced from the connection description class are data objects that define the path set connections (pins, paths) that are then bound to the test resources via the PATH_SETS function call when the virtual resource classes are instantiated. The number of pins and connection types have to match the terminals on the virtual resource with which it is bound.

5.4.3.1 Path sets identifier

This identifies the path sets that can be used by a virtual resource.

```
type PATH_SETS_IDENTIFIER is private;

NULL_PATH_SETS_NUMBER : constant PATH_SETS_NUMBER := 0;
  -- Unallocated path sets number

NULL_PATH_SETS : constant PATH_SETS_IDENTIFIER;
  -- Unallocated path sets identifier
```

The private part definition for CONNECTION_IDENTIFIER is:

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
private

  type PATH_SETS_IDENTIFIER is
    record
      ID    : PATH_SETS_NUMBER;              -- Unique ID number generated by
      PATHS : ACCESS_PATH_RECORD_SET_ARRAY;  -- PATH_SETS_DEFINITION generic
    end record;

  NULL_PATH_SETS : constant PATH_SETS_IDENTIFIER
                              := (NULL_PATH_SETS_NUMBER,null);
```

Defined functions that allow other packages to extract the contents of this private type are:

```
  function GET_PATH_SETS_NUMBER
             (ID : in PATH_SETS_IDENTIFIER)
             return PATH_SETS_NUMBER;

  function GET_PATH_SETS
             (ID : in PATH_SETS_IDENTIFIER)
             return ACCESS_PATH_RECORD_SET_ARRAY;
```

5.4.3.2 Path set

A path set is used to specify sets of virtual resource connection points to UUT pin relationships. Each PATH_RECORD_SET specifies one complete set of virtual resource to UUT pin association, e.g., HIGH => J1, LO => J2. The PATH_RECORD_SET_ARRAY defines all sets of virtual resource to UUT pin association for this particular virtual resource and the UUT.

Generally, connector designations assigned by the UUT manufacturer are used to describe the attachment points (Examples: J1_2, P2_AA for electrical connector; and AIRSPEED, STATIC, PORT_2, FITTING_A for pneumatic connectors). The names used shall be legal Ada enumeration names and be defined in one Ada enumeration type. Most functions, such as dc signals and synchro signals, have multiple leads which shall be attached to the correct UUT points. The pin descriptors (such as HIGH and LO; X, Y, and Z; etc.) are provided in the language to identify the signal port to which the pins to be connected.

```
  type PATH_RECORD is
    record
      P_DESC   : PIN_DESCRIPTOR;      -- Pin descriptor type
      C_TYPE   : CONNECTION_TYPE;     -- Connection type for connection
      C_NUMBER : CONNECTION_NUMBER;   -- Connection number of connection
    end record;

  type PATH_RECORD_SET is array (NUMBER range <>) of PATH_RECORD;

  type ACCESS_PATH_RECORD_SET is access PATH_RECORD_SET;

  type PATH_RECORD_SET_ARRAY
        is array (NUMBER range <>) of ACCESS_PATH_RECORD_SET;

  type ACCESS_PATH_RECORD_SET_ARRAY is access PATH_RECORD_SET_ARRAY;
```

5.4.3.3 Path connections and path set definition generics

These generic packages are instantiated to create the path sets definition functions used in the instantiation of the UUT connection generic.

The PATH_CONNECTIONS generic is instantiated for the UUT connection point enumerated type. The inner generic package PATH_SET_DEFINITION is then instantiated for each virtual resource set of path sets.

```
  generic

    type UUT_CONNECTION is (<>);

  package PATH_CONNECTIONS is
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
    -- Function to create UUT pin identifier from UUT_CONNECTION enumeration

    function UUT
              (CONN : in UUT_CONNECTION)
              return UUT_CONNECTION_IDENTIFIER;

    -- Function to pass ATE pin identifier

    function ATE
              (CONN : in ATE_CONNECTION_IDENTIFIER)
              return ATE_CONNECTION_IDENTIFIER;

    -- Function to pass REF pin identifier

    function REF
              (CONN : in REF_CONNECTION_IDENTIFIER)
              return REF_CONNECTION_IDENTIFIER;

    -- Generic package for creating path sets

    generic

      CNX : REQUIRE_CNX_STATEMENT;

      type CNX_SET is (<>);

      CAPABILITY : REQUIRE_CAPABILITY_STATEMENT := NO_REQ_CAP;

    package PATH_SETS_DEFINITION is

      -- Type mark for path set

      subtype PATH_SET is CNX_SET;

      -- Types to specify paths

      type PATH_SET_ARRAY  is array (CNX'RANGE) of CONNECTION_IDENTIFIER;
      type PATH_SETS_ARRAY is array (PATH_SET)  of PATH_SET_ARRAY;

      -- Function to build set of path sets and generate id value

      function PATH_SETS
                (SETS : in PATH_SETS_ARRAY)
                return PATH_SETS_IDENTIFIER;

    end PATH_SETS_DEFINITION;

  end PATH_CONNECTIONS;
```

The UUT function creates the UUT_CONNECTION_IDENTIFIER private data type from the input UUT_CONNECTION enumeration literal. The corresponding ATE and REF functions are included (even though they just return their input parameter) to provide consistency in their usage with the UUT function.

Example:

An example of a UUT description, including the specification of the UUT's TEST_PINS and instantiations of the UUT_PATHS generic package to specify the connection paths between a dc power source and the UUT's power connections, and between a 2-wire resistance measurement capability and the UUT's ADAPTER and SHORT_CHECK connections, is:

```
  -- Unit under test pin names

  type TEST_PINS is (P1_R1,P1_R2,P1_5,P1_6,P1_7,P1_8,P1_1,P1_2,P1_3,P1_4);

  package TEST_1_PATH_CONNECTIONS is new PATH_CONNECTIONS(TEST_PINS);

  use TEST_1_PATH_CONNECTIONS;

  -- Power supply path sets

  type DC_POWER_PATHS is (UUT_POWER);
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
package DC_POWER_P is new PATH_SETS_DEFINITION
          (CNX     => (HIGH,LOW),
           CNX_SET => DC_POWER_PATHS);

DC_POWER_CNX : PATH_SETS_IDENTIFIER
                 := DC_POWER_P.PATH_SETS
                      (SETS => (UUT_POWER => (UUT(P1_R1),UUT(P1_R2))));

-- 2-wire resistance measurement path sets

type RESISTANCE_2WIRE_PATHS is (ADAPTER,SHORT_CHECK);

package RESISTANCE_2WIRE_P is new PATH_SETS_DEFINITION
          (CNX     => (HIGH,LOW),
           CNX_SET => RESISTANCE_2WIRE_PATHS);

RESISTANCE_2WIRE_CNX : PATH_SETS_IDENTIFIER
  := RESISTANCE_2WIRE_P.PATH_SETS
       (SETS => (ADAPTER     => (UUT(P1_R1),UUT(P1_R2)),
                 SHORT_CHECK => (UUT(P1_7), UUT(P1_8))));
```

5.4.4 Connection error exception

A named exception is defined for CONNECT error. The exception may be raised when the attempt is made to connect a sensor to more than one path set.

```
CONNECT_ERROR : exception;
```

5.5 Resource capability requirements

Resource capability generic packages are defined to enable the user to specify the signal-oriented characteristics of the UUT's test requirements. They are instantiated in the TPS with the signal values for the generic parameters that specify the resource capability requirements for testing the UUT.

The test resource capability object classes defined by these generic packages are the following:

a) ANALOG_SENSOR_CAPABILITY,
b) ANALOG_SOURCE_CAPABILITY,
c) SIMPLE_LOAD_CAPABILITY.
d) TIME_INTERVAL_SENSOR_CAPABILITY,
e) TIMER_SOURCE_CAPABILITY,
f) DIGITAL_SENSOR_CAPABILITY, and
g) DIGITAL_SOURCE_CAPABILITY

Objects produced from the classes in the Test Resource Capability category are data objects that describe the test resource capability requirements. These data objects are later bound to executable objects produced from the classes in the Virtual Resource category.

Optionally, the resource capability objects can be instantiated by the ATE to indicate the capabilities supported by the ATE's available physical resources. These resource capabilities can then be used to support the UUT-oriented generic virtual resources required by the ATLAS-level test programs.

All of the resource capability generics specify the NAME (string) of the resource, the supported signal NOUN, and the CNX pin descriptors. They also provide additional parameters that are unique to each resource class. These are described in the following subclauses.

5.5.1 ANALOG_SENSOR_CAPABILITY

The analog sensor capability generic package (see 7.10, pg. 305) defines the characteristics for specifying the signal capabilities of analog sensor resources.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993

```
generic

  NAME        : RESOURCE_NAME;
  SIGNAL      : ANALOG_NOUN;
  MEASUREMENT : REQUIRED_MEASUREMENTS;
  CONTROL     : REQUIRE_CONTROL_STATEMENT     := NO_REQ_CON;
  CAPABILITY  : REQUIRE_CAPABILITY_STATEMENT  := NO_REQ_CAP;
  LIMIT       : REQUIRE_LIMIT_STATEMENT       := NO_REQ_LIM;
  CNX         : REQUIRE_CNX_STATEMENT;

  IDENTIFIER  : RESOURCE_NUMBER := UNALLOCATED_RESOURCE;

package ANALOG_SENSOR_CAPABILITY is

  -- Resource identifier

  function ID return SENSOR_ANALOG_RESOURCE;

end ANALOG_SENSOR_CAPABILITY;
```

Example:

An example for the instantiation of an ANALOG_SENSOR_CAPABILITY resource by the TPS is:

```
-- 2-wire resistance measurement resource

package RESISTANCE_2WIRE is new ANALOG_SENSOR_CAPABILITY
          (NAME        => "RESISTANCE_2WIRE",
           SIGNAL      => IMPEDANCE,
           MEASUREMENT => (1=>RESISTANCE),
           CAPABILITY  => (1=>REQD(RESISTANCE,FROM=>0.0*OHMS,TO=>10000.0*OHMS,
                                                    BY=>1.0*OHMS)),
           CNX         => (HIGH,LOW),
           IDENTIFIER  => 3);
```

The resource characteristics reflect the resistance measurement capabilities needed to test the UUT. They could also be used to specify the resistance measurement capabilities of the physical resources within the ATE.

5.5.2 ANALOG_SOURCE_CAPABILITY

The analog source capability generic package (see 7.11, pg. 306) defines the characteristics for specifying the signal capabilities of analog source resources.

```
generic

  NAME        : RESOURCE_NAME;
  SIGNAL      : ANALOG_NOUN;
  CONTROL     : REQUIRE_CONTROL_STATEMENT     := NO_REQ_CON;
  CAPABILITY  : REQUIRE_CAPABILITY_STATEMENT  := NO_REQ_CAP;
  LIMIT       : REQUIRE_LIMIT_STATEMENT       := NO_REQ_LIM;
  CNX         : REQUIRE_CNX_STATEMENT;

  IDENTIFIER  : RESOURCE_NUMBER := UNALLOCATED_RESOURCE;

package ANALOG_SOURCE_CAPABILITY is

  -- Resource identifier

  function ID return SOURCE_ANALOG_RESOURCE;

end ANALOG_SOURCE_CAPABILITY;
```

Example:

An example for the instantiation of an ANALOG_SOURCE_CAPABILITY resource by the TPS is:

```
-- DC power source
```


Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 24,2026 at 12:25:56 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1226.2-1993