

IEEE Standard for Waveform and Vector Exchange (WAVES)

Sponsor
**Design Automation Standards Subcommittee
of the
IEEE Computer Society**

and

**Automatic Test Program Generating Subcommittee
of the
IEEE Standards Coordinating Committee on Abbreviated Test Language for All Systems
(ATLAS) (SCC 20)**

Approved December 5, 1991

IEEE Standards Board

Approved May 22, 1992

American National Standards Institute

Abstract: The Waveform and Vector Exchange Specification (WAVES) is an application of the VHSIC Hardware Description Language (VHDL). This standard covers the dataset structure, header file, WAVES files, external files, execution model, and waveform semantics, and defines three standard packages to support WAVES datasets.

Keywords: VHDL, WAVES, WAVES datasets, WAVES standard packages

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

Copyright © 1992 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Published 1992
Printed in the United States of America

ISBN 1-55937-195-1

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

IEEE Standards documents are developed within the 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 which 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 once 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.

Acknowledgments

[These acknowledgments are not a part of IEEE Std 1029.1-1991, IEEE Standard for Waveform and Vector Exchange Specification (WAVES).]

The WAVES Waveform and Vector standardization effort has been supported by volunteers with expertise in computer systems design and manufacturing, aerospace, communications, CAD tool development, IC model development, and automatic test systems. The final language design presented in this document is the result of all participants, with significant contributions from Lee Shombert and Al Gilman. The hard work of the committee and the dedication to meeting the WAVES requirements is to be commended.

Al Lowenstein

Chair, SCC20 ATPG

Chair, DASS TEST

Bob Hillman

Chair, WASG

Foreword

[This foreword is not a part of IEEE Std 1029.1-1991, IEEE Standard for Waveform and Vector Exchange Specification (WAVES).]

The WAVES language was developed to permit simulation histories and test vector descriptions to move freely back and forth among simulation and test environments in a common, nonproprietary format. The primary purpose of WAVES is to define a common data format that can be used for this exchange of stimulus/response and test information among simulators and test equipment.

The WAVES effort arose out of the merger of two separate efforts: The work of the IEEE Standards Coordinating Committee (SCC20) ATPG Subcommittee to standardize the description of Waveforms and Test Vectors, and that of the IEEE Computer Society Design Automation Standards Subcommittee (DASS) to standardize VHSIC Hardware Description Language (VHDL) usage for test applications. A merger of these objectives and a search for a common solution was acknowledged to be the most desirable solution.

The conflict between capability and interfacing cost eventually led to the splitting of the standard into two levels. WAVES Level 1 is the minimal level that provides a simple implementation of core capabilities. For example, in WAVES Level 1 the external files must be in a WAVES-defined format. Level 2 provides the flexibility to organize the information in various ways, including providing interfaces to user-defined file formats, Level 2 being the most expressive and flexible. Level 1 is a strict subset of Level 2 and provides simplification while retaining core capabilities.

The following typographic conventions are used in this standard:

- Boldface is used to indicate reserved words. Lowercase reserved words are VHDL reserved words. Mixed-case reserved words are additional reserved words defined in this standard (WAVES reserved words).
- Italics are used in the text to introduce new terms and to set off multiple-word terms (e.g., waves header file) from the surrounding text. Italics are used in Chapter 5. for the name of trace events, and are also used for syntactic categories in the body of the text.
- Much of WAVES is defined in terms of a grammar, which is explained in Appendixes C and D.

The WAVES working group effort has been supported by international group volunteers with expertise in digital simulation and test, and from a variety of different environments including CAD tool development, automated test

system development, device manufacturers, system OEMs, and other areas. The synergy of such a group has resulted in a standard that implements a global consensus. The following people contributed to the development of the WAVES standard by participation in the working group in various aspects:

Mostapha Aboulhamid	J. Scott Calhoun	Marvin Friedman
Guy Adam	Terry Campbell	Ronald Froman
Gordon Adshead	David M. Cantwell	Walter Fruth
G. Adsmead	Steve Carlson	Michael Frye
Suresh Agarwahal	Harold W. Carter	Hideo Fujiwara
David Agnew	William Cartmell	Alain Gautron
Daniel P. Ahrens	Morris Chang	Nelson Gildenmeister
Robert W. Allen, Jr.	Art Chapman	Alfred S. Gilman
Sheila Anderson	Charles R. Childress	John Goddard
Thomas Andrews	Howey Chin	George Gohagan, Jr.
Samuel Archer-Davies	Daniel Chipiro	Rich Goldman
James Armstrong	Berthet Christian	Michael J. Gooding
Kunihiro Asada	Moon Jung Chung	Chris Gorringer
James H. Aylor	David Coelho	Arnold Greenspan
Brian Bailey	Frank Cole	John S. Grout
Keith Baker	Mike Cole	Pascal Guiraudou
Wendell C. Baker	Alain Collet	Thomas A. Gurda
Peter Barck	Frank Conforti	Andrew Guyler
Jean-Michael Barre	Robert Cook	William Guzek
David L. Barton	Tedd Corman	Jeffrey J. Haeffele
Larry Bashaw	Albert Cosand	Jeff Hahs
Steve Becraft	James Cottrell	Don Hall
Jean-Luc Becu	Les Crawly	Andy Halliday
Richard Beegle	John Crownover	Thomas M. Hamilton
Craig Bell	Gary C. Cubbage	Arthur Hann
Behrooz Bendali	Stephanie D'Ambra	William A. Hanna
Jean-Michel Berge	Robert Dahlberg	James P. Hanna
Dieter W. Bergman	Brian A. Dalio	Randolph E. Harr
Victor Berman	Steve Danbra	Carl W. Harris
David B. Bernstein	Danny Davis	Shuuji Hashimoto
J. Bhasker	Warren Debany	Bruce L. Havlicsek
William D. Billowitch	Vincent DeBuono	Steven E. Hebert
Saptal Birak	Warren Decamp	Herwig Heckel
Mark Birnaum	Joanne DeGroat	John Heiser
Michael Blair	Antonie C.S. deJager	Nobuaki Hidaka
Fonkova A. Blaise	Gregory deMare	John Hillawi
Brad Blough	James Dion	Robert G. Hillman
Dominique Borrione	Michael J. Donahue	Frederick Hinchliffe
James Brandt	Michael Dukes	John W. Hines
Patrice Brard	Thomas M. Elliott	Willis J. Horth
David Brittle	James Emerick	Rob Houghton
Lisa Brockett	Dave Erstad	Andy Huang
Ronald Brooks	Mark Faust	Chris Humphreys
Peter J. Brooks	Graham Fellows	Monique Hyvernaud
Leon Brunfield	Len Finegold	Yukihiko Ido
Wolfgang Buhr	Christine Fisher	Gary Imken
Antonius Bunsen	Ed Fitzgerald	Mitsuaki Ishikawa
Phil Burlison	Jacques Flandrois	Syun Ishiyama
Bruce Burton	Rob Fletcher	John Ivie
Frank Cada	Lars H. Fransson	John S. Jacobson

David Jakopac
Phillippe Jeannin
John Jensen
Marth Kachmar
Gunther Kainz
Kim Kanzaki
Steve Karlovic
Carl Karrfalt
Mort Katz
Helmuth Kaunzinger
Steve Kelem
Peter Kiesel
Choon Kim
Lisa Kirsch
Eskil Kjelkerud
Rich Knoll
Frances Koo
Jan M. Koopmans
Jed Krohnfeldt
John Kuban
Tsuneo Kurobe
Sonja V. Kval
Henry Lahore
Martin Lampard
Brian LaPorte
John Larkin
Mario Larouche
Marc Laurent
Dave Lawson
Jean LeBrun
Carl Leeber
David E. Lehtonen
Carol Lemlein
J.P. Letellier
John E. Lewis
Richard Linderman
Russ Lohmeier
David B. Loveman
Alfred Lowenstein
David Luckham
Heike Ludwig
Joseph Luhukay
Bruce Lundeby
Elmer Lyons
Bob Mackey
Mark Malinoski
Tadayasu Mamoto
Jean-Paul Marinier
Michael A. Martin
Michael S. Martin
Toshio Maruyama
Bill McCaudless
Kirk McClure
William McKinney

Manfried A. Meschke
Frans A.C. Meys
Larry M. Mikkelsen
Tadjan Milan
Jose Miranda
Mukund Modi
Clayton L. Mohr
Larry Moran
Craig Morency
Gabe Moretti
David S. Morris
John Moschner
Wolfgang Mueller
Steve Munroe
Michio Murakami
Thomas W. Murphy
James Nagy
Hiroshi Narimatsu
Wolfgang Nebel
Brad Nelson
Joe Newman
Dana Nichel
Junji Nishiura
Lothar Nowak
Ghulam Nurie
Martin O'Farrell
Jim O'Reilly
Hans Obermeier
Kenzo Okumura
Leslie Orlidge
Robert Parker
Richard Patrick
Rodger Philips
Alain Plassart
Andrew Poon
Gary S. Porter
Signe L. Post
Jean Pouilly
Virginia Price
Michael Proctor
J. Pukite
Eric John Purslow
Carol Pyron
Brian Raffaelli
James Reeder
Eugene L. Reinhard
Earl Reinkensmeyer
Matteo Sonza Reorda
Dave Riccardi
Tim Rice
John Richards
Dwane Rigsby
Gordon Robinson
Parker Robinson

Robert M. Rolfe
Deborah L. Rooney
Jacques Rouillard
Christian Roux
Ashim Roy
William E. Russell, Jr.
Ashraf M. Salem
Michael Sandlin
Albert Sanson
Terutoshi Sasami
Gabriele Saucier
Frederick W. Sauer
John Saunders
James E. Scharf
Helmut Scheibenzuber
Patrick U. Schibli
Steven M. Schlosser
Goel Schoen
Thomas A. Scholz
Jerry Sewell
Moe Shahdad
Ravi Shankar
Brijendra Sharma
Mark Shaver
Bruce Shay
Scott Sheldon
Lee A. Shombert
Takeshi Sigematzu
Steven P. Smith
Richard C. Smith
J. W. Smith
Harvey A. Soloway
Joseph J. Stanco
Alec G. Stanculescu
Fred Steinberg
Mark Stockburger
Micki Stone
Mariella Styers
Tetsuo Tada
Ryuiti Takagi
Akio Tamama
Toshio Tamamura
Daniel Tang
Jay Taylor
David Taylor
Stephen Taylor
Jim Teisher
Mitsuo Teramoto
Jacques Tete
Peter Thelwall
Jon Thomas
Paul A. Traceski
Erwin Trischler
Erwin Trischler

H. Edward Tsou
John Turino
Louis Ungar
Cary Ussery
Radha Vaidyanathan
Dean Vendl
Venkat Venkataraman
Bas Verhelst
Hai Vo-Ba
Raymond P. Voith
Rene J. Vuarnoz

Leroy Wagner
Michael G. Wahl
Charles J. Walton
Ivan Dennis Ward
Shigeru Watari
Thomas Watts
Ronald Waxman
J. Richard Weger
Jim Welch
Ron Werner
Ted Whitehouse

John Willner
Scott Wilson
Mark A. Woloshyn
James Wyburn
Dhiren Xavier
Takao Yano
Chua Thian Yee
Will Young
Alex Zamfirescu
David A. Zeek
Gary Zeigler

The following persons were members of the balloting group that approved this document for submission to the IEEE Standards Board:

Mostapha Aboulhamid
G. Adsmead
Thomas Andrews
James H. Aylor
Behrooz Bandali
Craig Bell
Jean-Michel Berge
David B. Bernstein
William D. Billowitch
Michael Blair
Fonkova A. Blaise
James Brandt
Peter Brooks
Scott Calhoun
Terry E. Campbell
Steven Carlson
Moon Jung Chung
Frank Conforti
Brian A. Dalio
Joanne DeGroat
Thomas M. Elliott
Jacques P. Flandrois
Rob Fletcher
Michael Frye
Nelson W. Gildenmeister
Alfred S. Gilman

Rich Goldman
Arnold Greenspan
Jeffrey J. Haeffele
Thomas M. Hamilton
Arthur Hann
William A. Hanna
James P. Hanna
Bruce Havlicsek
John E. Heiser
John I. Hillawi
Robert Hillman
John Hines
Chirs Humphreys
David Jakopac
Carl Karrfalt
Choon B. Kim
Eskil Kjelkerud
Frances Koo
Alfred Lowenstein
Joseph F. P. Luhukay
Robert J. Mackey
Michael Martin
Larry L. Moran
Craig D. Morency
David S. Morris
Wolfgang W. Nebel
Jim O'Reilly

Gary Porter
Michael Proctor
J. Pukite
Brian Raffaelli
Earl Reinkensmeyer
Gordon Robinson
Robert Rolfe
Deborah L. Rooney
Jacques Rouillard
Ashim Roy
William Russell
Ashraf M. Salem
Steven M. Schlosser
Joel Schoen
Moe Shahdad
Ravi Shankar
Mark Shaver
Lee A. Shombert
Jon H. Thomas
Erwin Trischler
Louis Y. Ungar
Cary Ussery
Dean C. Vendl
Bas C. Verhelst
Ron Werner
Will W. Young

When the IEEE Standards Board approved this standard on December 5, 1991, it had the following membership:

Marco W. Migliaro, *Chair*
Donald C. Loughry, *Vice Chair*
Andrew G. Salem, *Secretary*

Dennis Bodson
Paul L. Borrill
Clyde Camp
James M. Daly
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ingrid Fromm
Thomas L. Hannan

Donald N. Heirman
Kenneth D. Hendrix
John W. Horch
Ben C. Johnson
Ivor N. Knight
Joseph Koepfinger*
Irving Kolodny
Michael A. Lawler

John E. May, Jr.
Lawrence V. McCall
T. Don Michael*
Lloyd A. “Peter” Morley
Stig L. Nilsson
John L. Rankine
Ronald H. Reimer
Gary S. Robinson
Terrance R. Whittemore

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Fernando Aldana
Satish K. Aggarwal

James Beall

Richard B. Engelman
Stanley Warshaw

Mary Lynne Nielsen
IEEE Standards Project Editor

CLAUSE	PAGE
1. Dataset Structure	1
1.1 References	1
2. Header File	2
2.1 Header File Format	2
2.2 Dataset Identification	3
2.3 Waves File Identification	3
2.4 External File Identification	4
2.5 Waveform Identification	5
3. WAVES Files.....	5
3.1 Syntax.....	5
3.2 Semantics	6
3.3 Standard Packages.....	8
3.4 Structure	8
4. External Files	10
4.1 WAVES Level 1	10
5. Execution Model	13
5.1 Traces	13
5.2 Events	14
5.3 Subprogram Semantics	16
6. Waveform Semantics	20
6.1 Event Values	20
6.2 Tag Values	23
7. Standard Packages.....	25
7.1 Waves_Standard.....	25
7.2 Waves_Interface.....	28
7.3 Waves_Objects.....	36
Annex A Reserved Words (Informative)	45
Annex B Glossary (Informative).....	51
Annex C WAVES Level 2 Syntax (Informative)	52
Annex D WAVES Level 1 Syntax (Informative)	63

IEEE Standard for Waveform and Vector Exchange (WAVES)

1. Dataset Structure

The Waveform and Vector Exchange Specification (WAVES) is an application of the VHSIC Hardware Description Language (VHDL), described in IEEE Std 1076-1987 [1]¹. This document uses the term “VHDL” to refer to the VHSIC Hardware Description Language and the term “VHDL LRM” to refer to IEEE Std 1076-1987 [1]. In no case shall the WAVES specification be in conflict with IEEE Std 1076-1987 [1].

The WAVES specification defines two “levels” of datasets. These levels are called WAVES Level 1 and WAVES Level 2. All language structures in WAVES Level 1 are included in WAVES Level 2, all of which in turn are found in VHDL. In this sense, WAVES Level 1 is a proper subset of WAVES Level 2, which is in turn a proper subset of VHDL.

A WAVES dataset is the logical collection of data required to specify one or more waveforms completely. A WAVES dataset consists of a *header file*, one or more *waves files*, and zero or more *external files* of ancillary data. The header file specifies how the dataset is assembled from the waves files and the external files. The waves files conform to the syntax and semantics of WAVES Level 2 or WAVES Level 1. The external files are in a format that may be used by the waves files to produce a WAVES waveform.

A WAVES waveform is produced by a *waveform generator procedure*. The waveform generator procedure is defined by the waves files. The waveform generator procedure may use external files as a source of data.

NOTE — Because all language structures included in WAVES Level 1 are included in WAVES Level 2, a Level 1 dataset is always a legal Level 2 dataset. Therefore, the WAVES levels do not represent two distinct specifications but rather a single specification.

1.1 References

[1] IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual (ANSI).²

[2] Hoare, C. A., *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985.

¹The numbers in brackets correspond to those of the references in 1.1.

²IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes line, Piscataway, NJ 08855.

2. Header File

The WAVES dataset header file specifies information that is not available within the waves files or the external files, yet that is necessary or useful to describe the WAVES dataset completely.

The header file specifies the file names of the waves files and references standard waves units in a legal order for WAVES analysis. The order of analysis for waves files and standard waves units is the same as that of the equivalent VHDL design units. In the case of multiple waves units within a single waves file, the waves units shall be in the proper order for analysis. Referenced waves units are standard packages whose bodies may be different between different organizations and installations, and that may not be shipped with the dataset. Inclusion of their names in the header file specifies that they are to be included in the analysis order as though the waves units themselves appeared there.

The WAVES dataset header file allows the specification of a mapping of logical file names to physical file names for external files.

In addition to the above information, the header file specifies how the dataset is to be identified, the names of the waveform generator procedures, and any additional comments as necessary.

2.1 Header File Format

The format of the header file is a sequence of data records. Each data record starts with a keyword and is terminated by an end-of-record.

A data record has one or more parameters depending upon the keyword.

Blank lines are permitted, as are lines containing only VHDL-style comments. A comment line begins with a double hyphen. Leading white space is ignored in data records and comments. The syntax of the productions below (called the metasyntax) is specified at the beginning of Appendix C.

```
header_file →
    TITLE <string> <cr>
    DEVICE_ID <string> <cr>
    <status_id> { <status_id> }
    <waves_unit_id> { <waves_unit_id> }
    { <external_file_id> }
    <waveform_id> { <waveform_id> } ◇

status_id →
    DATE <string> <cr>
    ORIGIN <string> <cr>
    AUTHOR <string> <cr>
    { AUTHOR <string> <cr> }
    { OTHER <string> <cr> } ◇

waves_unit_id →
    { <context_item> <cr> } <waves_unit_reference>
    | <waves_file_id> ◇

waves_unit_reference →
    WAVES_UNIT <package_name> <library> <cr>
    | WAVES_UNIT_SPEC <package_name> <library> <cr>
    | WAVES_UNIT_BODY <package_name> <library> <cr> ◇
```

```

waves_file_id →
    WAVES_FILENAME <file_name> <library> <cr> ◇

external_file_id →
    EXTERNAL_FILENAME <file_name> <file_logical_name> <cr> ◇

waveform_id →
    WAVEFORM_GENERATOR_PROCEDURE <procedure_name> <cr> ◇

```

In the above syntax, all-uppercase text represents header file keywords. Header file keywords are case insensitive. The *cr* represents the end of record. Parameters labeled *string* begin with the first nonspace character after the keyword and include all characters up to the end of record. Other parameters are delimited by space characters or equivalents (such as tabs).

The nonterminal *context_item* is included in the WAVES syntax, along with many other VHDL language structures, by the definition in 3.1. A *context_item* shall fit entirely on a single record when used in a header file.

NOTE — Section 3.1 defines the syntax of the WAVES standard by including all VHDL language structures except for an explicit list of exceptions; thus, a definition of *context_item* does not explicitly appear. A syntax summary for WAVES that includes the definition of *context_item* appears in Appendixes C and D.

The definition of *context_item* implies that **library** and **use** are also header file keywords.

2.2 Dataset Identification

The *dataset_id* provides useful information about the name of the dataset, the author, the release date, the origin, and a device identifier. The *dataset_id* is required in the WAVES dataset header file. Every field in the *dataset_id* takes a *string* as a parameter. The *string* begins with the first nonwhite character following the keyword and continues to the end of the record. There is no predefined format for any of the parameters.

The interpretation of the *dataset_id* fields is as follows:

Title	A descriptive name for the dataset
Device_Id	The Unit Under Test (UUT) to which the dataset is to be applied
Date	A timestamp
Origin	The originating Computer Aided Design (CAD) or Automatic Test Equipment (ATE) environment of the dataset
Author	The person(s) or organization(s) producing the dataset
Other	Comments relating to the dataset

NOTE — The definition of *string* means that spaces may be used freely in the *dataset_id* fields.

2.3 Waves File Identification

The *waves_unit_id* statements provide the names of the WAVES source files and standard packages in the correct order for analysis. The format provides a mechanism for specifying the library into which each file is analyzed. The *library_logical_name* is also the name that appears in the *use_clauses* of other waves units when referring to the waves unit.

A waves file may be provided as part of the physical dataset. The name of the file shall be provided by a *waves_unit_id* statement. The *file_name* is the name of the waves file in the local operating system syntax. The *file_name* is any legal file specification, except that space characters are not permitted in the file specification. The *library* parameter indicates the WAVES library into which the waves file is to be analyzed. Every waves unit in the waves file will be analyzed into the specified library.

A waves file may also consist of a reference to an already existing waves file in the design library of the organization receiving the dataset. In particular, the standard WAVES packages **Waves_Interface** and **Waves_Objects** are included in a WAVES dataset through *waves_unit_reference* statements. A *waves_unit_reference* statement is required that names the waves units being referenced. The *package_name* parameter is the name of the pre-existing waves unit that is to be analyzed as part of the waves dataset. The *library* parameter indicates the WAVES library into which the waves unit is to be analyzed.

Pre-existing waves units must be present in the design library of the organization receiving the dataset. The keyword with which the reference is identified also identifies which parts of the package shall be analyzed when the rest of the dataset is analyzed, as follows:

Waves Unit	Causes both the package declaration and the package body to be analyzed
Waves Unit Spec	Causes only the package declaration to be analyzed
Waves Unit Body	Causes only the package body to be analyzed

A *waves_unit_reference* may be preceded by one or more *context_item*. A *context_item* provides required context information to pre-existing waves units. The parameter to a *context_item* is textually inserted at the beginning of the pre-existing waves unit prior to analysis of that unit.

A *context_item* is not required for the standard WAVES packages, provided that one and only one declaration of the required type has been analyzed prior to the *waves_unit_reference* naming the standard WAVES package.

The standard VHDL packages *Standard* and *TextIO* may not be referenced in a *waves_unit_id* statement. The WAVES package **Waves_Standard** may not be referenced in a *waves_unit_id* statement. The libraries *Std* and **Waves_Std** may not be specified in a *waves_unit_id* statement.

NOTE — Because *file_name* parameters are delimited by space characters, the file name may not contain any spaces. On some operating systems, the file name is case sensitive.

The *library* and *package_name* parameters are syntactically an *identifier*, as defined in 3.1.

The context clauses required to analyze the standard WAVES packages are defined in 3.4.

2.4 External File Identification

The *external_file_id* provides a mechanism for specifying a logical name for an external file accessed from a WAVES dataset. The *file_logical_name* is the name used in the file declaration statement of the waves file that defines the external file. This name shall exactly match the value of the string expression used in the file declaration. The *file_name* is the physical file name of the external data file in the syntax of the local operating system. Zero or more *external_file_id* statements may be in a WAVES dataset header file.

NOTE — Any double quotes used in the waves file to delimit a string literal are not part of the *file_logical_name*.

Because *file_name* parameters are delimited by space characters, the file name may not contain any spaces. On some operating systems, the file name is case sensitive.

The file allows the writer of a dataset to specify a logical name rather than a physical file name in the waves file *file_declaration* statement.

2.5 Waveform Identification

The *waveform_id* identifies a waveform generator procedure. The identification includes the library and package in which the procedure is declared. The *procedure_name* is composed of the library name, the package name, and the procedure name separated by dots (“.”) No white space is permitted in the *procedure_name*. One or more *waveform_id* statements may be in a WAVES dataset header file.

3. WAVES Files

A waves file is a legal VHDL design file.

3.1 Syntax

The permitted syntax of a waves file is a subset of the VHDL syntax.

3.1.1 WAVES Level 2

WAVES Level 2 syntax includes all of the VHDL syntax with the following deletions:

<alias_declaration>	<architecture_body>
<attribute_declaration>	<component_declaration>
<configuration_declaration>	<disconnection_specification>
<entity_declaration>	<file_type_definition>
<operator_symbol>	<physical_type_definition>
<signal_assignment_statement>	<signal_declaration>
<wait_statement>	

Furthermore, the following syntactic elements are modified by deleting part of their definition:

<actual_designator>	loses	open
<association_element>	loses	<formal_part> =>
<element_association>	loses	<choices> =>
<entity_class>	loses	entity
<entity_class>	loses	architecture
<entity_class>	loses	configuration
<entity_class>	loses	signal
<entity_class>	loses	component
<formal_designator>	loses	<generic_name>
<formal_designator>	loses	<port_name>
<interface_signal_declaration>	loses	bus
<interface_signal_declaration>	loses	:= <static_expression>
<interface_constant_declaration>	loses	:= <static_expression>
<interface_variable_declaration>	loses	:= <static_expression>
<mode>	loses	buffer
<mode>	loses	linkage
<package_body_declarative_item>	loses	<file_declaration>
<package_body_declarative_item>	loses	<use_clause>
<package_declarative_item>	loses	<file_declaration>
<package_declarative_item>	loses	<use_clause>
<subprogram_declarative_item> ,	loses	<use_clause>
<subtype_indication>	loses	<resolution_junction_name>

NOTE — It is a consequence of the requirement that a waves file is a legal VHDL design file that VHDL reserved words are WAVES reserved words, whether or not they are used by the WAVES syntax.

The above restrictions mean that WAVES Level 2 contains only syntax from the sequential portion of VHDL. No language constructs concerning parallel execution and simulation time are permitted, including signal assignment statements and wait statements. The effect is to reduce the WAVES syntax to that of a simple algorithmic programming language, such as Pascal.

The resulting syntax is summarized in Appendix C. The notation used for describing syntax is also described in Appendix C.

3.1.2 WAVES Level 1

WAVES Level 1 syntax includes all of the WAVES Level 2 syntax, with the following deletions:

<code><access_type_definition></code>	<code><allocator></code>
<code><composite_type_definition></code>	<code><floating_type_definition></code>
<code><incomplete_type_declaration></code>	<code><type_conversion></code>
<code><variable_assignment_statement></code>	<code><attribute_specification></code>
<code><subtype_declaration></code>	

Furthermore, the following syntactic elements are modified by deleting part of their definition:

<code><enumeration_literal></code>	loses	<code><character_literal></code>
<code><scalar_type_definition></code>	loses	<code><integer_type_definition></code>

NOTE — These restrictions prohibit variable assignment statements and most type declarations. A WAVES Level 1 tool becomes an expression evaluator, in contrast to a Level 2 tool, which is closer to an interpreter or compiler.

The resulting syntax is summarized in Appendix D. The notation used for describing syntax is also described in Appendix D.

3.2 Semantics

A WAVES file obeys the semantic rules established by IEEE Std 1076-1987 [1].

3.2.1 Private Types

Certain types are used by the various WAVES standard packages as system-dependent information. When used by the WAVES standard packages, a subtype is declared that is equivalent to the system dependent type. These system dependent types, and the subtypes that are used within the WAVES standard packages, are called private types. Intuitively, objects whose base type is a private type may only be used as a unit and not decomposed into parts. More formally:

- 1) The name of a private type object is forbidden in the following places:
 - a) As the prefix to a selected name
 - b) As the prefix to an indexed name
 - c) As the prefix to a slice name
 - d) As the prefix to an attribute name
- 2) Expressions of private types are restricted to functions declared in the WAVES standard packages.

The names of types that are designated as private will be identified when the types themselves are discussed.

NOTE — The WAVES standard packages as published in this document presume that system dependent types are declared in the package *Waves_System* residing in the library **Waves_Std**. However, the standard does not require this package to exist, nor does the standard require that the declarations of private types be of any particular form. The standard merely requires that a particular implementation of a private type correctly capture the semantic requirements imposed by this document.

Aside from the language structures imported from VHDL, this standard consists of a number of package declarations. These packages, called WAVES standard packages, have an interlocking relationship with packages (with standard names) provided by the organization that integrates WAVES datasets into its activities. This interlocking relationship requires that no knowledge concerning the internal structure of private types crosses this boundary (defined either by the WAVES standard package or by the organization packages). VHDL provides no mechanism for enforcing hiding information between packages. The private type mechanism provides this information-hiding mechanism; the declaration of the internal subtype prevents the access of the internal structure of these types in VHDL.

3.2.2 WAVES Level 2

WAVES Level 2 restricts or prohibits certain VHDL constructs. The restrictions are as follows:

- 1) An identifier may be declared only once within any scope. The visibility rules must determine at most one possible meaning for the name of any user-defined subprogram.
- 2) Identifiers declared in standard VHDL packages or in the WAVES packages are reserved words. Reserved words may not be redefined in a WAVES design file.
- 3) If a WAVES standard package is named in a context clause, then the entire package shall be made directly visible with the “.all” notation.
- 4) Files shall be of mode *in*.
- 5) An interface signal declaration may only use the type names **Waves_Port_List** or **Waves_Match_List**.
- 6) The type **Waves_Port_List** is private and is only permitted in an interface signal declaration with a mode of *inout*.
- 7) The type **Waves_Match_List** is private and is only permitted in an interface signal declaration with a mode of *in*.
- 8) Objects of type **Waves_Port_List** or **Waves_Match_List** may not appear in an expression and may not be referenced except as an actual designator in a subprogram call.
- 9) The types **Event_Value**, **Frame**, **Frame_List**, **Frame_Set**, **Logic_Map**, and **Frame_Set_Array** are private and are only permitted at the following places:
 - a) In a constant declaration
 - b) In an interface constant declaration
 - c) As the return type name of a function
- 10) The types **Event_Time_Basis** and **Delay_Time_Basis** are private.
- 11) The library name **Waves_Std** may not appear except in a *library* clause or as the prefix to the package name **Waves_Standard**.

NOTE — The first restriction prohibits overloading of subprograms. The restriction on overloading is specified so that a semantically correct WAVES file does not change meaning when considered as VHDL.

The restrictions on private types mean that the implementation of these types may vary from system to system without affecting the meaning of a WAVES dataset.

The restrictions on objects of type **Waves_Port_List** and **Waves_Match_List** mean that the only operation permitted on such objects is to pass an incoming interface object as an actual parameter to a subprogram call.

The restrictions on **Delay_Time_Basis** and **Event_Time_Basis** do not preclude the use of these typemarks in type declarations, nor do they preclude the use of objects of these types in aggregate literals.

Other semantic restrictions follow from the syntax restrictions:

- (1) Because file type definitions are not permitted, the only valid type for a file object is *Text*, as defined in the standard VHDL *Std.TextIO* package.
- (2) Because the syntax of *association_element* and *element_association* is restricted, named association is not permitted.

3.2.3 WAVES Level 1

WAVES Level 1 further restricts VHDL semantics as follows:

- 1) Subprogram parameters shall be of mode *in* except for the following cases:
 - a) Parameters of type **Waves_Port_List** shall be of mode *inout*.
 - b) Parameters of type **File_Slice** shall be of mode *inout*.
- 2) All objects other than formal parameters shall be initialized at the time of declaration.

3.3 Standard Packages

There are two standard VHDL packages, *Std.Standard* and *Std.TextIO*. Every WAVES package is placed in the context of *Std.Standard.all*, as indicated in IEEE Std 1076-1987 [1].

3.3.1 WAVES Level 2

WAVES Level 2 waves files may not reference the function *Std.Standard.Now*.

WAVES Level 2 waves files may only reference the following objects from the *Std.TextIO* package:

- The type *Line*
- The type *Text*
- The procedure *ReadLine*
- The overloaded procedures *Read*
- The function *EndFile*

3.3.2 WAVES Level 1

WAVES Level 1 waves files may only reference the following object from the *Std.TextIO* package:

- The type *Text*

NOTE — Within a WAVES Level 1 waves file, an object of type *Text* may only be passed as an argument to a subprogram and may not be referenced in any other manner. The procedure **Read_File_Slice** is provided to users of the WAVES Level 1 standard; this procedure reads a file in the WAVES Level 1 external format and returns the equivalent file slice.

3.4 Structure

WAVES waves files define a waveform generator procedure. The totality of waves files shall provide certain required declarations and shall conform to package visibility restrictions. The required declarations may be provided in a waves file that is physically part of the dataset or in a referenced waves unit. All declarations shall appear within a package declaration.

NOTE — These requirements apply to WAVES Level 2 and WAVES Level 1 equally except for the requirements in 3.4.7.

3.4.1 Waves_Standard

The WAVES standard package **Waves_Standard** resides in the library **Waves_Std**.

3.4.2 Logic_Value

The enumerated type **Logic_Value** shall be declared. Within the same package declaration the function **Value_Dictionary** shall be declared. This function shall accept a single input parameter of the type **Logic_Value** and return an **Event_Value** object.

Visibility to the standard package **Waves_Interface** is not permitted at the point of the **Logic_Value** declaration.

NOTE — It is a consequence of the requirement that the **Value_Dictionary** function return an object of type **Event_Value** that the contents of the standard package **Waves_Standard** shall be directly visible at the point of the **Value_Dictionary** declaration.

3.4.3 Waves_Interface

The WAVES standard package **Waves_Interface** is included in a WAVES dataset through a *waves_unit_reference* statement in the header file. At the point of analysis of the **Waves_Interface** package, the required declaration **Logic_Value** shall be directly visible.

NOTE — The WAVES semantic and visibility restrictions imply that the type **Logic_Value** cannot be declared within a package in which declarations depending on the package **Waves_Interface** also appear.

3.4.4 Pin_Codes

The string constant **Pin_Codes** shall be declared within a package declaration. This constant may include any character from the enumerated type *Character* in package *Std. Standard*; however, no character may appear more than once. The string shall contain at least two characters. Lowercase letters are distinct from uppercase letters.

NOTE — Functions returning **Frames**, **Frame_Lists**, and **Frame_Sets** may be declared in the same package declaration as **Pin_Codes**. At the point of the function declaration, the contents of the standard package **Waves_Interface** must be directly visible.

At the point of any subprogram body that returns values of the type **Frames**, **Frame_Lists**, or **Frame_Sets**, the type **Logic_Value** must be directly visible.

3.4.5 Test_Pins

The enumerated type **Test_Pins** shall be declared in a package declaration. There shall be one enumeration value for each pin specified by the dataset.

The constant **Pin_Direction** declaration is optional but, if present, shall have a type of **Direction_List** and have one entry for each enumeration value in the type **Test_Pins**. This constant provides a summary of the direction information for each pin in the dataset. For every pin in the dataset, the information in the constant shall be consistent with all event value directions specified for that pin in the dataset, as follows:

- **Unspecified** is consistent with any direction
- **Unknown** is consistent with **Stimulus** or **Response**
- **Stimulus** is consistent with **Stimulus** only
- **Response** is consistent with **Response** only
- **Compound** is consistent with **Stimulus**, **Response**, and simultaneous **Stimulus** and **Response**.

The **Pin_Direction** declaration, if present, shall be in the same package declaration as the **Test_Pins** declaration.

3.4.6 Waves_Objects

The WAVES standard package **Waves_Objects** is included in a WAVES dataset though a *waves_unit_reference* statement in the header file. At the point of analysis of the **Waves_Objects** package the required declarations **Logic_Value**, **Test_Pins**, and **Pin_Codes** shall be directly visible. The contents of the package **Waves_Interface** shall be directly visible.

3.4.7 Waveform Generator Procedure

A waveform generator procedure shall be declared within a package declaration and is identified with a *waveform_id* statement in the waves header file. Each such procedure represents a distinct waveform.

The first parameter of a waveform generator procedure is required and shall be of type **Waves_Port_List**.

The waveform generator procedure may represent a *non-match mode* waveform or a *match mode* waveform. A match mode waveform requires a parameter of type **Waves_Match_List**, which shall immediately follow the **Waves_Port_List** parameter.

The waveform generator procedure may represent a *nonparametric* waveform or a *parametric* waveform. A parametric waveform is one that takes additional parameters beyond the required *Waves_Port_List* and the optional **Waves_Match_List**. These parameters shall have a class of **constant**.

A WAVES Level 1 dataset may not define a parametric waveform generator procedure.

NOTE — It is a consequence of the declarations in the standard packages that at the place of the waveform generator procedure declaration the contents of the standard package **Waves_Objects** must be directly visible. At the place of the corresponding procedure body the contents of the standard packages **Waves_Interface** and **Waves_Objects** must be directly visible.

The WAVES semantic restrictions require that the **Waves_Port_List** parameter have a class of *signal* and a mode of *inout*. If there is a **Waves_Match_List** parameter, then the WAVES semantic restrictions require that the class of the parameter be *signal* and that the mode of the parameter be *in*.

4. External Files

A WAVES dataset may contain external files that are not in the syntax of waves files. The dataset shall also supply a procedure that reads the external file and converts the data in the external file to standard WAVES objects. The meaning of an external file is therefore dependent on the supplied reader.

4.1 WAVES Level 1

A single external file format is available for WAVES Level 1 datasets.

4.1.1 Syntax

White space is permitted between any of the tokens in the following syntax. The syntax element *cr* denotes an end of line. IEEE Std 1076-1987 [1] does not define the meaning of the end of line. This standard follows this, and the term *cr* will similarly remain undefined.

```

level_1_external_file →
    { <file_slice> <cr> } ◇

file_slice →
    { <file_slice_command> } ; ◇

file_slice_command →
    <quoted_string> | <skip_command> | <timing_command> ◇

skip_command →
    = <integer> ◇

timing_command →
    : <integer> | : <time> ◇

quoted_string →
    " <character> [ <non_blank_string>
    | <non_special_character> <non_blank_string> ◇

```

The syntax definitions for *integer* and *time* are given by the VHDL syntax. A *non_special_character* is any printing character that is not otherwise recognized as part of the syntax. A *non_blank_string* is a sequence of printing characters.

A *file_slice* may span multiple records but every *file_slice* shall begin on a new record. Blank records in the file are permitted.

A comment begins with a comment operator and runs to the end of the record. The comment operator is a single percent character ("%").

The value of a *skip_command* is the value of the integer that follows the equals sign. The value of a *timing_command* is the value of the integer or time that follows the colon. The value of a *quoted_string* that begins with a double quote ("") is the string formed from the character immediately following the quote and the following nonblank string. The value of a *quoted_string* that does not begin with a double quote is the nonblank string itself.

NOTE — The special characters are "=", ":", ",", "%", and " ".

The definition of a quoted string is such that embedded special characters are treated as part of the string. Therefore, a space or end of record is required to delimit the right-hand side of the quoted string.

Spaces may be needed to delimit, for instance, the *integer* in a *skip_command* from a leading digit in a *quoted_string*.

4.1.2 Standard Meaning

Within a WAVES waves unit a file slice from an external file is stored in the following structure:

```
type File_Slice is record
  CODES                :STD.TEXTIO.LINE;
  FS_TIME              :TIME;
  FS_INTEGER           :INTEGER;
  END_OF_FILE          :BOOLEAN;
end record;
```

The procedure **Read_File_Slice**, declared in the package **Waves_Objects**, reads a file slice from an external file in the Level 1 syntax and stores it in a **File_Slice** object according to the following rules. These rules are given as actions to be taken at the reduction of each nonterminal.

Prior to the beginning of the read, the *current_position* is set to the beginning of the *codes* field.

```
level_1_external_file →
  { <file_slice> <cr> } ◇
```

Action: If a file slice is present, return from the procedure with the given assembled file slice. If no file slice is present, return an end of file record to the caller.

```
file_slice →
  { <file_slice_command> } ; ◇
```

Reduction rule only.

```
file_slice_command →
  <quoted_string> | <skip_command> | <timing_command> ◇
```

Reduction rule only.

```
skip_command →
  = <integer> ◇
```

Action: Skip to the position in the *codes* string indicated by the integer by updating the *current_position*.

```
timing_command →
  : <integer> ◇
```

Action: Set the *fs_integer* field to the value of the integer.

```
timing_command →
  : <time> ◇
```

Action: Set the *fs_time* field to the value of the given time.

```
quoted_string →
  " <character> [ <non_blank_string> ]
  | <non_special_character> <non_blank_string> ◇
```

Action: Starting from the *current_position*, place each character in the *quoted_string* into successive positions in the *codes* field, updating *current_position* as necessary. If the procedure is called with the pin list *active_pins*, use *active_pins* to index into the *codes* field; otherwise, index from left to right.

NOTE — It is an error if a character is stored when the current position exceeds the length of the *codes* field. When called with an *active_pins* parameter, it is an error if a character is stored when current position exceeds the length of the *active_pins* parameter.

Blank lines and comments do not affect the end-of-file condition. That is, an external file may end in a series of comments.

An external file in the Level 1 format may be read, in WAVES Level 2, by a procedure other than **Read_File_Slice**. In that case, the meaning of the characters in the external file is determined solely by the procedure reading the external file.

5. Execution Model

A WAVES dataset may be regarded as being written in a sequential programming language. The semantics of the structures of this programming language are identical to the semantics of the equivalent structures in VHDL. A WAVES dataset is thus wholly simulatable within the VHDL environment. Simulation of a WAVES dataset in a VHDL environment requires package bodies corresponding to the specifications of the WAVES standard packages. These package bodies may be altered as necessary by different installations. The results of executing the subprograms in the package bodies shall match the execution model given below.

The waveform produced by the dataset is a result of interaction between the dataset and its environment, regardless of whether that environment is VHDL, some other simulation environment, or a test environment. This interaction may be captured formally as a *trace*. A full description of trace theory may be found in the text *Communicating Sequential Processes* by C. A. Hoare [2]. For the purposes of this discussion, the following summary is given.

5.1 Traces

A *trace* is a sequence of events. Events may occur in one process only or may be shared by two processes. In a WAVES dataset, exactly two processes are postulated: the dataset itself and the environment of the dataset. Events that occur in one process only may be executed at the discretion of that process. Events shared between two processes must be executed by both processes in cooperation.

The action of a WAVES dataset may be described formally using a vocabulary of seven events:

Set_Value (Value, Pin, Time) This event marks the initiation (in time) of an association between the **Event_Value** Value describing signal conditions and the signal identified by Pin.

Advance_Time (Time) This event advances the current time to the specified time.

Start_Match (Pin, Time) This event marks the initiation of a matching time window.

Stop_Match (Pin, Time) This event marks the termination of a matching time window.

Match (Pin, Bool, Time) This event extracts from the environment its success or failure in conforming to the dataset-specified signal conditions.

Handshake (Pin, Value, Time) This event extracts from the environment the time of a transition on Pin from not-Value conditions to Value conditions.

Tag (String, Pin) This event occurs within the dataset and marks the trace for the given pin with the given string.

The dataset adds events to the trace through the use of operations that appear as subprograms defined by the WAVES standard packages. These subprograms are **Apply**, **Match**, and **Tag**. The **Apply** procedures create *Set_Value*, *Handshake*, and *Advance_Time* events. The **Match** procedures create *Start_Match* and *Stop_Match* events. The **Match** functions create *Match* events. The **Tag** procedures create *Tag* events. Each subprogram has several versions to allow variation in the form of parameter passage. The definitions of all versions are given below.

NOTE — The sub programs declared in the WAVES standard packages take advantage of overloading permitted in VHDL. The user-defined subprograms in WAVES files may not.

This standard does not specify the realization of the subprograms **Apply**, **Match**, and **Tag**. These procedures, along with the precise declarations of the types **Waves_Port_List** and **Waves_Match_List**, may vary from implementation to implementation. The WAVES standard specifies conditions on the actions of any realization of these subprograms in terms of the effect of these operations on the waveform trace, which is sufficiently abstract to apply to both simulation and test environment realizations.

5.2 Events

The detailed specification of each of the event classes is provided in the following subsections.

5.2.1 Set_Value (Value, Pin, Time)

The value aspect of this event is a predicate or descriptor of signal conditions known as an **Event_Value**. It associates the conditions of the **Event_Value** with the signal associated with the particular pin at the identified time and for any subsequent time until there is another *Set_Value* event for this pin. This event occurs within the dataset.

Prior to any *Set_Value* events on a pin, the value associated with the pin is the **Event_Value** associated with the leftmost **Logic_Value**.

NOTE — Event Values are defined in detail in 6.1.

This event is generated by the **Apply** operation under the control of values specified in objects of type **Logic_Value** at times under the control of objects of type **Event_Time**. The **Event_Time** type is defined in 7.2.1. The **Apply** operation is defined in the following Subprogram Semantics section.

The selection of a precise time for each *Set_Value* event, within the constraints set up by timing tolerances expressed in the waves data, is to be construed as a function of the realization of the dataset process. This is true whether the realization is a simulation subprogram or a test program compiler. Environmental influence on the waveform timing is accomplished through the *Handshake* event.

5.2.2 Advance_Time (Time)

This event advances the current time to the specified time. It is executed within the dataset and is used as a marker to the environment. The new time shall not be less than the old time.

NOTE — This event is generated by the **Apply** operation under the control of the specifications provided in objects of type **Delay_Time**. This type is defined in 7.2.1.

5.2.3 Start Match (Pin, Time)

Each pin in **Test_Pins** shall have, in the realization of the environment of a match mode WAVES dataset, the functional equivalent of two binary states for matching. There is a one-bit Boolean register to hold the match result and a one-bit control flag that takes on the values (*Hold*, *Sample*).

Prior to any *Start_Match* events, the match result register contains the value False and the control flag contains the value Hold.

While the control flag is *Sample*, the environment saves in the results register the logical “and” of the prior result and the truth value of the statement, “The signal conditions observed by the environment conform to the requirements as stated in the dataset.” The required signal conditions, for the evaluation of this statement, are only those that have a relevance of *required* and a direction of *response*. At any time, the current dataset event value is the event value in the most recent *Set_Value* event on the subject pin.

This event is shared between the dataset and its environment. The dataset realization of a **Match** procedure call with a control parameter of *Sample* is to offer the environment a *Start_Match* event at that time for each designated pin.

The environment realization of *Start_Match* is, when offered such an event by the dataset, to inspect its matching control flag for that pin. If the prior value of that flag is *Hold*, the event is accepted. The environment sets the control flag to *Sample*, sets the result register to True, and initiates results-register updating as of the current time.

NOTE — Multiple calls to the **Match** procedure that all have a control value of *Sample* will not generate multiple *Start_Match* events. Only the first such call results in an event in the trace.

5.2.4 Stop_Match (Pin, Time)

This event closes a matching time window.

This event is shared between the dataset and its environment. The dataset realization of a **Match** procedure call with a control parameter of *Hold* is to offer the environment a *Stop_Match* event at that time for each designated pin.

The environment realization of *Stop_Match* is, when offered such an event by the dataset, to inspect its matching control flag for that pin. If the prior value of that flag is *Sample*, the event is accepted. The environment performs one last comparison- and-result update at the time of this event. It then sets the control flag to *Hold*. If the control flag value is already *Hold*, the event is rejected and not placed in the trace.

NOTE — This buffer subsequently remembers the result as of the *Stop_Match* time until the next *Start_Match* event on this pin. The time of this event is included in the matching time window.

Multiple calls to the **Match** procedure that all have a control value of *Hold* will not generate multiple *Stop_Match* events. Only the first such call results in an event in the trace.

5.2.5 Match (Pin, Bool, Time)

This event is shared by the dataset and its environment.

The dataset realization of a **Match** function call is to offer the environment a set of two events with a fixed Pin and Time, and Boolean values of both True and False.

The environment realization of this event is to select one of these depending on the contents of its matching results buffer for that pin at the current time. The results buffer is updated to the current time before deciding which offered event to match.

NOTE — The control flag is unaltered by the Match event.

5.2.6 Handshake (Pin, Value, Time)

This event extracts an edge time from the environment for dataset timing.

This event is shared between the dataset and the environment.

The dataset realization of an **Apply** call requiring a handshake delay is to offer the environment a set of events for all times starting at the current time with specific values of Pin and Value.

The environment realization of this event is to select the offered event with the earliest time available such that the truth-value of the statement, “The environment signal conditions on Pin conform to the event value in Value” transitions from False to True at the event time.

NOTE — Coarse synchronization to the environment can be achieved by the *Match* function sensing environmental signal levels, but precise synchronization requires the use of a **Delay_Time** specifying a handshake delay.

5.2.7 Tag (String, Pin)

This event occurs within the dataset and marks the trace for the given pin with the given string. The pin field may be missing, in which case the trace is marked for the waveform as a whole.

A **Tag** event is included in the execution trace for the purposes of documentation. It is executed as the result of a call to the **Tag** procedure. It has no effect on the generation of events by the dataset; however, it may have an effect on the environment. This effect may be installation specific. Some tag values have specified effects, which are listed in 6.2.

5.3 Subprogram Semantics

The operations **Apply**, **Tag**, and **Match** are defined above in terms of the events they add to the execution trace of the dataset. The families of overloaded subprograms that implement these operations are described in this subsection.

5.3.1 Apply Operation

Every **Apply** procedure realizes an **Apply** operation using the following information:

- A list of event specifications for each pin. The list for any or all pins may be empty, that is, contain no events; a pin for which no events are specified is said to have an empty list.
- A **Delay_Time** value governing the time to elapse prior to the time of the next **Apply** operation.
- An indication as to which pins are active in this **Apply** operation. This set may be empty, contain some pins, or contain all pins.

Separately declared overloaded **Apply** procedures within the WAVES standard packages accept this information in different data forms, but they all resolve their input parameters to the standard information and take the following actions with that information.

Event times are reduced to unique, absolute times. For a given event, this may require reference to the current time or to the time of another event specified within the same **Apply** operation. Any time compatible with the tolerances specified in the governing **Event_Time** object may be selected as the unique time of an event. Where the time of one “dependent” event is referenced to the time of another “independent” event, the unique time of the independent event is selected first, and the offset and tolerances of all dependent events are interpreted relative to that selected time.

For a given pin, the unique times selected in the resolution process shall be nondescending.

Set_Value events are generated for every event the event list for each pin in the active pins set.

Appropriate *Advance_Time* and *Handshake* events are generated. If the governing **Delay_Time** object requires a handshake delay, a *Handshake* event is executed, followed by an *Advance_Time* to the time of the *Handshake* event. Whether or not a handshake is involved, an *Advance_Time* compliant with the nominal delay and tolerances in the **Delay_Time** object is then executed.

A call to **Apply** with a handshake delay shall include the *waves_match_list* parameter.