



IEEE Guide for the Use of IEEE Std 1641, Standard for Signal and Test Definition

IEEE Standards Coordinating Committee 20

Sponsored by the
IEEE Standards Coordinating Committee 20 on
Test and Diagnosis for Electronic Systems

1641.1TM

IEEE
3 Park Avenue
New York, NY 10016-5997, USA
30 April 2007

IEEE Std 1641.1TM-2006

IEEE Guide for the Use of IEEE Std 1641, Standard for Signal and Test Definition

Sponsor
**IEEE Standards Coordinating Committee 20 (SCC20) on
Test and Diagnosis for Electronic Systems**

Approved 16 November 2006
IEEE-SA Standards Board

Abstract: Guidance in the use of the signal and test definition (STD) standard is provided. STD provides the means to define and describe signals used in testing. This guide describes how to implement, apply, and use a set of common basic signals to form complex signals usable across all test platforms.

Keywords: ATE, ATLAS, ATML, ATS, automatic test equipment, signal definitions, test definitions, test requirements, test signals, unit under test, UUT

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2007 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 30 April 2007. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

Print: ISBN 0-7381-5266-8 SH95589
PDF: ISBN 0-7381-5267-6 SS95589

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

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

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

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

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

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

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

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

This introduction is not part of IEEE Std 1641.1-2006, IEEE Guide for the Use of IEEE Std 1641, Standard for Signal and Test Definition.

IEEE Std 1641™-2004, known as the signal and test definition (STD) standard, defines a method of accurately defining signals and their timing as used in test procedures and requirements. This guide supplements IEEE Std 1641-2004 and is part of the document set.

This guide has been prepared to help all users of IEEE Std 1641-2004. The STD standard may be used within any test discipline and with any carrier language, and the examples provided in this guide should be seen as typical.

This guide explains how each of the layers in the STD standard is built up on the preceding (lower) layer and tells the user how to use the layer or layers that are important to a specific application. It describes how a signal may be created from basic signal components (BSCs) or other signals. It also shows the application and measurement of signals using a text-only format.

The need for a guide arose from the experience of users in the creation of signals and tests using IEEE Std 1641-2004. This experience showed that further information in the implementation and application of signal definitions was required. The purpose of this guide is to provide guidance in the technique of implementation, application, and usage of the basic signals defined in IEEE Std 1641-2004 to create signal definitions and test requirements. This is seen as particularly important in promoting the use of a relatively new and unambiguous method of describing signals. This guide is not intended to be used as an instruction manual for IEEE Std 1641-2004 nor as a substitute for formal training, but by its nature it should find some application in the training environment.

The initial clauses are intended as a brief introduction to the application of STD. Subsequent clauses concentrate on the description of signals used in test. Stimulus signals, conditioning elements, and the acquisition of information from response signals are all covered.

The STD standard does not specify any specific carrier (test program sequencing) language, and this guide does not provide any advice about the selection of a suitable carrier language. It does show STD used within typical carrier languages and includes examples of the definition and use of signal models in different environments.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/interp/index.html>.

Patents

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

Participants

At the time this guide was completed, the Test and ATS Description Working Group had the following membership:

Ashley Hulme and Ion A. Neag, Co-Chairs

Malcolm Brown
Matt Cornish
Keith Ellis

Chris Gorringer
Arthur Hann
Mike Harrison

Jim Langlois
Spike Milligan
Narayanan Ramachandran

The following members of the individual balloting committee voted on this guide. Balloters may have voted for approval, disapproval, or abstention.

S. K. Aggarwal
Adewole C. Akpose
Egor I Alekseev
Ali Al Awazi
Saber Azizi-Ghannad
Charles L. Barest
Hugh Barrass
Martin Baur
Barry L. Beaster
Malcom Brown
Danila Chernetsov
Keith Chow
Tommy P. Cooper
Matt Cornish
Luis M. Coronado
Charles B. Cotton
Joseph Czapski
Thomas J. Dineen
Carlo Donati
David B. Droste
James Dumser
Heiko Ehrenberg
Keith Ellis
William B. Frank
Devon L. Gayle
Fernando Genkuong
Ian C. Gifford
Sergiu R. Goma
Chris Gorringer
Arnold M. Greenspan
Randall C. Groves
A. W. Guy
Timothy E. Harrington
Mike Harrison

Gary A. Heuston
Werner Hoelzl
Dennis K. Holstein
Bernard Homes
Dennis Horwitz
Ronald W. Hotchkiss
James D. Huddleston III
Ashley Hulme
Arshad Hussain
Atsushi Ito
Anand Jain
Anthony A. Jeffree
Sheila A. Johnston
Efthymios G. Karabetsos
Osamu Karatsu
Piotr Karocki
Stuart J. Kerry
John Knowles
Jim Kulchisky
Jeremy A. Landt
Jim Langlois
Solomon Lee
Charles A. Lennon, Jr.
Michael Lerer
Daniel A. Lindberg
William Lumpkins
G. L. Luri
Jon S. Martens
Michael J. Maytum
Edward M. McCall II
Earl J. Meiers
Francisco J. Melendez
Gary L. Michel
Scott Misha
Mukund U. Modi

Georges F. Montillet
Ion A. Neag
Michael S. Newman
Charles Kamithi Ngethe
Chris L. Osterloh
Gregory D. Peterson
Ann Mary Pine
Cam K. Posani
Maurice M. Reintjes
Peter Richardson
Robert A. Robinson
David A. Rohacek
Daniel E. Rossi
Randall M. Safier
Marco Scorrano
Mike Seavey
Bogdan Seliger
John W. Sheppard
Jack Sherman
Shono Takashi
Dustin E. Soria
Roger J. Solowada
John H. Spare
Luca Spotorno
Manikantan Srinivasan
Steven E. Swanson
Douglas Taylor
James E. Timperley
Thomas A. Tullia
Srinivasa R. Vemuru
Charles M. Whitaker
Ernesto Jorge Wiedenbrug
Bruce R. Youmans
Oren Yuen

When the IEEE-SA Standards Board approved this guide on 16 November 2006, it had the following membership:

Steve M. Mills, *Chair*
Richard H. Hulett, *Vice Chair*
Don Wright, *Past Chair*
Judith Gorman, *Secretary*

Mark D. Bowman
Dennis B. Brophy
Joseph Bruder
Richard Cox
Bob Davis
Julian Forster*
Joanna N. Guenin
Mark S. Halpin
Raymond Hapeman

William B. Hopf
Lowell G. Johnson
Herman Koch
Joseph L. Koepfinger*
David J. Law
Daleep C. Mohla
Paul Nikolich

T. W. Olsen
Glenn Parsons
Ronald C. Petersen
Gary S. Robinson
Frank Stone
Malcolm V. Thaden
Richard L. Townsend
Joe D. Watson
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan H. Cookson, *NIST Representative*

CONTENTS

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
2. Normative references.....	1
3. Definitions, acronyms and abbreviations	2
3.1 Definitions	2
3.2 Acronyms and abbreviations	3
4. Historical background	3
4.1 Review of ATLAS.....	3
4.2 Decision to define a new standard	4
4.3 Requirements for STD standard	4
4.4 Inclusion of support for Automatic Test Markup Language (ATML).....	4
5. Features of the STD standard	5
6. Hierarchy of signal definitions	5
7. Process overview	7
7.1 General	7
7.2 TSF model as signal template.....	7
7.3 Typical TSF model	10
7.4 Using TSFs in test requirement or program.....	11
7.5 Signal measurement.....	12
8. Creating signals using STD standard.....	12
8.1 Overview	12
8.2 Physical types	15
8.3 Signals	16
8.4 Using signal graphs to create a signal.....	20
8.5 Documenting signal descriptions.....	24
8.6 Test requirement capture	29
8.7 Using TPL to create test requirements.....	38
9. Defining measurements with STD.....	59
9.1 General	59
9.2 Sensors.....	60
9.3 Measurement maps	64
9.4 Generic measurement	65

9.5 Intrinsic measurements	68
9.6 Different valid measurement methods	70
9.7 Reference signals description	73
10. Basic signal components (BSCs).....	74
10.1 Introduction of new BSCs	74
10.2 BSC interfaces	74
10.3 Diagrammatic representation of BSCs.....	75
10.4 SignalFunctions and events	77
11. Test Signal Framework (TSF).....	93
11.1 Introduction to TSF	93
11.2 Building TSF signal models using BSCs.....	93
11.3 Examples of source signal models.....	93
11.4 Signal models with preset internal attributes	102
11.5 Examples of signal models that process input signal.....	106
11.6 Example of signal models that include connection BSCs.....	111
11.7 SignalDelay TSF model for SWEEP	114
11.8 TSF model for linear sweep using frequency modulation (FM) BSC	116
11.9 TSF model for logarithmic sweep using FM BSC.....	117
11.10 TSF attributes mappings and formulae	118
11.11 Synchronization of signal model	119
11.12 Gating a signal model	129
11.13 Use of XML to specify TSF signal model information	131
11.14 Use of IDL to specify TSF signal model information	141
12. Test Procedure Language (TPL).....	145
12.1 Introduction to TPL	145
12.2 Use of ATLAS with STD	146
12.3 Simple test requirement in TPL.....	146
12.4 Further test requirement in TPL.....	154
12.5 Examples of test statements in TPL.....	162
12.6 Quantities, units, and unit symbols.....	171
13. Signal Modeling Language (SML).....	172
13.1 Introduction to SML	172
13.2 Using SML to define a BSC	173
Annex A (informative) Glossary	179
Annex B (informative) Generic measurement.....	181
B.1 Introduction.....	181
B.2 Dependence upon reference signal.....	181
B.3 Measurements on complex signal (square wave).....	186
Annex C (informative) Implementation examples	188
C.1 Implementing multiple STD environments.....	188
Annex D (informative) Bibliography	190

IEEE Guide for the Use of IEEE Std 1641, Standard for Signal and Test Definition

1. Overview

1.1 Scope

This guide provides application information and guidance for users who write, develop, implement, and support test requirements, signal definitions, and signal responses using IEEE Std 1641™-2004, known as the signal and test definition (STD) standard.¹ Examples of the definition and use of signal models in different environments are included.

1.2 Purpose

The purpose of this guide is to explain how to implement signal definitions and test requirements in conformance with IEEE Std 1641-2004. It also provides background information, tutorial support, and examples of signal definitions and test requirements for users of the STD standard.

2. Normative references

The following referenced document is indispensable for the application of this guide. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1641-2004, IEEE Standard for Signal and Test Definition.²

¹ For information on references, see Clause 2.

² IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (<http://standards.ieee.org/>).

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this guide, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B6]³ should be referenced for terms not defined in this clause.

3.1.1 attribute: A property value that is used to define signal characteristics or behavior.

3.1.2 base class: A class from which another class inherits attributes or properties.

3.1.3 basic signal component (BSC): Lowest level building block used to define signals.

3.1.4 carrier language: A native computer language used to support signal and testing definition (STD) signals (whether in the form of test statements or graphical representations) and to provide all other programming requirements. STD signals will need to be converted into carrier language code before program execution can commence.

3.1.5 connection: **(A)** The application of a signal to a unit under test (UUT). **(B)** The physical contact through which the signal passes.

3.1.6 dynamic signal: A signal whose definition changes over time, by use of the control interface. These changes must be initiated by using one of the signal method calls or by changing the interconnections of a signal model.

3.1.7 event stream: A signal comprising a stream of active/inactive pulses generated from signal models.

3.1.8 gated event stream: An event stream used to gate a basic signal component (BSC) or test signal framework (TSF).

3.1.9 Interface Definition Language (IDL): A machine-compilable language that is used to describe the interfaces that software objects call and object implementations provide. The language provides a neutral way to define software interfaces.

3.1.10 physical: Pertaining to the natural characteristics of entities and the natural laws of science.

3.1.11 procedural: The part of a signal and test definition (STD) test requirement that defines the tests in the manner and order required for testing.

3.1.12 reserved word: A specified keyword whose meaning and use are fixed by the semantics of the language in which it is reserved. It cannot be used for any other purpose than the one defined for the language.

3.1.13 signal graph: *See:* **signal model.**

3.1.14 signal model: A collection of test signal frameworks (TSFs) or basic signal components (BSCs) interconnected to define a signal. Used to define the model part of a TSF (TSF model) or a signal component used with the Require method. *Syn:* **signal graph.**

³ Numbers in brackets correspond to the numbers in the bibliography in Annex D.

3.1.15 SignalFunction: The name of the base class for all classes that provide signals.

3.1.16 static signal: A signal whose definition does not change over time. All basic signal components (BSCs) and test signal frameworks (TSFs) are static signal definitions.

3.1.17 system: A set of interconnected hardware and/or software entities that achieves a defined objective by performing specified functions.

3.1.18 test signal framework (TSF): A combination of basic signal components (BSCs) to form more complex signals.

3.1.19 template: A pattern or design that establishes the outline, dimensions, or process for subsequent users or implementers.

3.1.20 value: The quantitative magnitude of a signal attribute.

3.2 Acronyms and abbreviations

ADC	analog-to-digital converter
AM	amplitude modulation
ASCII	American Standard Code for Information Interchange
ATE	automatic test equipment
ATLAS	Abbreviated Test Language for All Systems
ATML	Automatic Test Markup Language
ATS	automatic test system
BSC	basic signal component
COTS	commercial off-the-shelf
DME	distance measuring equipment
FM	frequency modulation
IDL	Interface Definition Language
ILS	instrument landing system
PM	phase modulation
rms	root-mean-square
SI	Système International d'Unités (International System of Units)
SML	Signal Modeling Language
STD	signal and test definition
TPL	Test Procedure Language
trms	true root-mean-square
TSF	test signal framework
UUT	unit under test
XML	Extensible Markup Language

4. Historical background

4.1 Review of ATLAS

The STD standard is the culmination of a radical review of ATLAS (i.e., Abbreviated Test Language for All Systems) and the requirement to create truly portable test requirements. ATLAS, in both its military and commercial subsets, has been very successful in providing test requirements independent of test equipment since the late 1960s. A carefully specified test requirement using ATLAS may be converted into test programs (with a minimum of effort) that may then execute on different test equipment. Furthermore,

ATLAS test requirements are easily read by users who are familiar with the English language and test-engineering-oriented terms.

ATLAS does have some shortcomings, and the STD standard has been created to overcome them. The major areas where improvements were required were determined to be as follows:

- The definitions of signals should be mathematically rigorous and not open to interpretation.
- The language structure should be less rigid and able to take advantage of modern object-oriented techniques.
- The method for extensibility should be straightforward and fully portable.
- The time taken before new technologies could be supported should be minimized or preferably eliminated.

4.2 Decision to define a new standard

During the review process, it was determined that it would be difficult to revise the existing C/ATLAS standard, i.e., IEEE Std 716™-1995 [B10], to include the required improvements. The decision was made to formulate a new standard to resolve these issues. This standard would stand alongside C/ATLAS (and may be used to define signals used in C/ATLAS requirements). It was also decided that the initial version of the new standard would include information to relate the new signal descriptions to the signals described in the C/ATLAS standard.

4.3 Requirements for STD standard

The key features that came from the review of ATLAS resulted in a series of requirements for the STD standard.

The principal requirements for the STD standard were determined to

- a) Provide the means to unambiguously define all signals.
- b) Provide a sound mathematical foundation for the definition of signals so that signals could be accurately exported, re-created, and used directly by different test equipment.
- c) Enable signals to be defined by providing an architecture that permitted the combination of existing signals.
- d) Support the collection of signals with similar application domains into readily extensible libraries.
- e) Provide a means whereby signals could be defined, used, and controlled in several programming languages.
- f) Enable test programs to be written for any test system that supported the same STD signals.

4.4 Inclusion of support for Automatic Test Markup Language (ATML)

During the time that the STD standard was being developed, the ATML was initiated. The purpose of ATML is to define a collection of Extensible Markup Language (XML) schemas that allow automatic test equipment (ATE) and test information to be exchanged in a common format adhering to the XML standard (see IEEE P1671™ [B11]). This led to the inclusion of XML schemas for the basic signal components (BSCs) and test signal frameworks (TSFs) in the STD standard, providing a basis for test requirements written using ATML to be supported by the same signal definitions.

5. Features of the STD standard

The key features of the STD standard are as follows:

- a) *Definition of signals.* STD has a rigorous mathematical and definitive foundation for all of its signal components. In other words, any signal defined using this standard will be the same regardless of what equipment is used.
- b) *Modern language structures.* STD does not specify a programming language. Any carrier language is acceptable. Its suitability is determined by its support of the minimum requirements as described in the STD standard.
- c) *Extensibility.* All signals are defined mathematically based on BSCs. New signals may be described by combining existing signals and BSCs. Thus, any desired signal can be described, and there is no limit on the extensibility of signals supported by the standard.
- d) *Implementation of new technologies.* The STD standard supports the implementation of new technologies by providing users with the ability to describe their own signals.
- e) *Relationship with the existing C/ATLAS standard.* The STD standard includes enhanced support for the existing C/ATLAS standard [B10] by describing typical examples of C/ATLAS signals (i.e., Nouns and Noun Modifiers) in a TSF library.
- f) *Flexibility of use.* The STD standard not only supports traditional textual test requirements and sequences, using the Test Procedure Language (TPL), but also supports modern graphical programming techniques, diagnostic driven programming techniques, and other environments where the test signals and timing need to be separated from the program sequencing and signal routing instructions and implementations.

6. Hierarchy of signal definitions

The STD standard uses a hierarchy of definitions to describe any signal. This hierarchy is illustrated in Figure 1, which shows the four STD layers. Most users will not need to familiarize themselves with every level in the hierarchy. The lowest level, the Signal Modeling Language (SML) level, will be used only by certain specialists within an organization. It may be used for the introduction of a new keyword to cover a new test application not currently embraced. The SML definitions are presented using Haskell 98 [B13].

The level above the SML is the BSC layer. This layer provides all the basic building blocks that enable more complex signals to be constructed. BSCs create SignalFunction base classes and BSC subclasses of up to four levels. A BSC may be used on its own as a signal to be applied to a unit under test (UUT). For example, a simple ac signal may require no more than is provided with the sinusoid BSC. If the ac signal requires further parameters, then the sinusoid BSC will have to be combined with one or more other BSCs to provide the overall signal desired.

Combining one or more BSCs into a more complex model results in a TSF model. TSF models compose the next level in the hierarchy. These represent the majority of the signals required by a test requirement to be applied or measured at the interface with a UUT. The TSF models belonging to a particular test technology may be grouped together into a TSF library. The STD standard provides a typical TSF library in an informative annex (Annex E of IEEE Std 1641-2004). This annex presents examples of many of the signals described in C/ATLAS. It should be noted that TSF signals are not necessarily exclusive to a single TSF library. For example, a dc signal may be in many different libraries. TSF signals (in different libraries) with the same name do not have to be the same signal because the signal definition is carried with the TSF class definition.

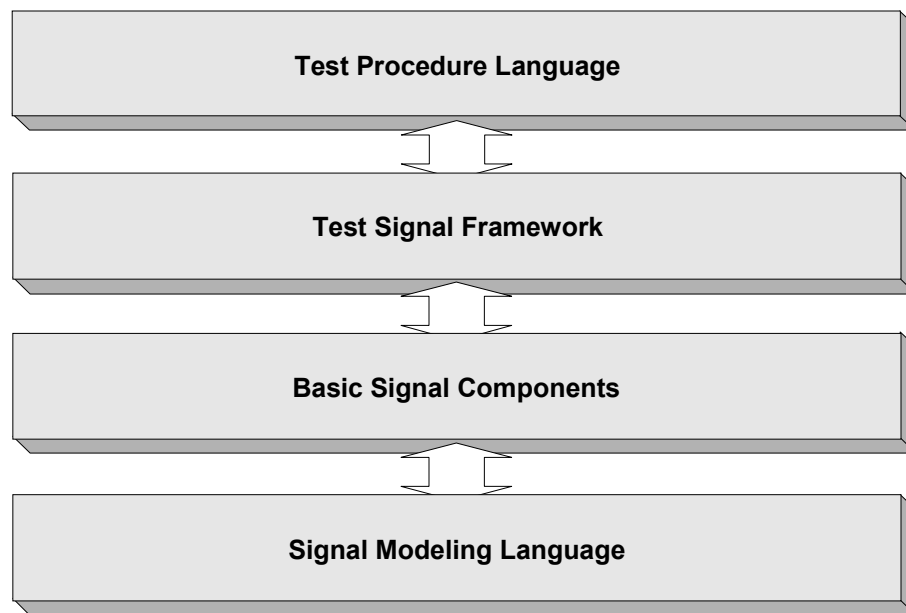


Figure 1—The four STD layers

It is the structured hierarchy of these three lower levels that ensures the portability of test requirements supported by the STD standard. As every signal applied or measured is founded upon the same lower level definitions, all implementations of the same test requirement should produce the same results.

The upper layer is the TPL layer. This layer provides the same programming facilities as any other test programming language, but it also includes signal-oriented descriptions that were a feature and an advantage of ATLAS. The same restricted programming features that ATLAS contained, however, do not bind this layer.

The STD standard defines how signal statements should be written; in other words, all implementations of the STD standard will provide the same signal for an identical definition. The programmatic part of a test requirement may be written in any suitable programming language. This language is referred to as the *carrier language* in the STD standard because it is the programming language that “carries” the signal statements. Annex H of IEEE Std 1641-2004 shows how TPL statements are mapped into a typical carrier language.

The TPL is not necessarily restricted to a text-based program definition environment. Users are free to use a graphical programming environment should that be more relevant to their application.

The body of the STD standard concentrates upon the application of the STD standard to the electrical signal domain. The content of the STD standard can be applied to a much wider scope of signal definition, however, because the signal-defining terminology is not restricted to the electrical signal domain. For example, pneumatic, hydraulic, and mechanical signals can also be defined by the BSCs. The signal values and their dimensions have to be appropriate to the nonelectrical signal domain.

7. Process overview

7.1 General

One of the primary use cases for the STD standard is to capture test requirements and create test programs. The STD standard has a variety of methods that could be used for this task, and initially it can be daunting to identify a process using the STD standard. This clause introduces some best practices that can be followed for this activity, and users can refine these practices to fulfill their individual requirements.

A key feature of the STD standard is the ability to create user signal components and maintain them within a library environment. Within the STD standard, these signal components are called *TSF models*, and a collection of TSF models used for a particular application, technology, or project is a *TSF library*. TSF models can be used to support stimulus signals, events, measurements, and the processing of existing signals.

It is also possible to define a complete test, including all the timing and location information, within a single template signal or TSF model.

7.2 TSF model as signal template

A TSF model may be considered as a reusable signal template with parameters, signal inputs, and signal outputs. The parameters are known as *attributes* and will be one of the types described in 7.2.1. The signal model (template) may be reused by providing different values for the attributes.

7.2.1 Types of TSF attributes

The following attributes can be found in a TSF model:

- a) *Control attributes*. A control attribute is mapped onto a model property. It provides information about the characteristics of a signal.

Example: voltage 10 V. The resource provides a signal with a voltage of 10 V.

- b) *Capability attributes*. An attribute that is not mapped onto any signal model is regarded as a capability attribute and may be thought of as a requirement attribute. It holds information about the capability of the resource required to supply the signal

Example: current 2 A. The resource must be capable of supplying a signal that may draw up to 2 A.

- c) *Value attributes*. A value attribute represents a value obtained from the signal model. These attributes are normally read-only.

Example: when an interface attribute is mapped onto a model measurement attribute. The resource must be capable of providing measurement information about a signal, such as the amplitude of the signal.

- d) *Expression or formula attributes.* An expression or formula attribute is mapped to or from a model property via an expression.

Examples:

- Type conversion, such as “period = 1/freq”
- Measurement analysis, such as “mode = HistMode (Measurements)”
- Text manipulation

7.2.2 Creating a TSF

These TSF models may be defined in XML as shown in the following examples:

- a) Provide the name of the library component.

```
<TSF name="AC_SIGNAL" uuid=...>
```

- b) Specify TSF base type.

```
<xs:extension base="SignalFunctionType" />
```

- c) Define attributes in the <interface> section.

```
<xs:attribute name="ac_ampl" type="Physical">
```

- d) Specify any default attribute values.

```
default="0 rad"
```

- e) Specify any restrictions to attribute values.

```
<xs:maxInclusive value="2*pi rad"/>
```

- f) Define the behavior model in the <model> section.

```
<Signal Out="AC_Signal" ...>  
  <Constant name="DC_Offset" amplitude="dc_offset"/>  
  <Sinusoid name="AC_Component" amplitude="ac_ampl"  
    frequency="freq"  
    phase="phase"/>  
  <Sum name="AC_Signal" In="DC_Offset AC_Component"/>  
</Signal>
```

- g) Specify any attribute formula.

```
(period="1/freq")
```

- h) Assign value attributes formula.

```
(default="rms.measurement/carrier.amplitude")
```

Many of the above examples are shown in the following XML description of an AC_SIGNAL.

```
<TSF name="AC_SIGNAL" uuid="{CA4F7FD8-D05E-11D6-860D-00010214C4D2}">
  <interface>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
      <xs:element name="AC_SIGNAL">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="SignalFunctionType">
              <xs:attribute name="ac_ampl" type="Physical">
            </xs:attribute>
              <xs:attribute name="dc_offset" type="Physical" default="0">
            </xs:attribute>
              <xs:attribute name="freq" type="Frequency">
            </xs:attribute>
              <xs:attribute name="phase" default="0 rad">
            </xs:attribute>
              <xs:simpleType>
                <xs:restriction base="PlaneAngle">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="2*pi rad"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</interface>
<model>
  <Signal Out="AC_Signal" xmlns="STDBSC" >
    <Constant name="DC_Offset" amplitude="dc_offset"/>
    <Sinusoid name="AC_Component" amplitude="ac_ampl"
      frequency="freq"
      phase="phase"/>
    <Sum name="AC_Signal" In="DC_Offset AC_Component"/>
  </Signal>
</model>
</TSF>
```

Additional tools may be required to deal with formulae within a signal model:

- Formula translations allow TSF models to manipulate attribute values.
- Formula translations may be used to specify measurement analysis.
- The formulae formats are normally in Haskell 98 [B13], which may not be the most suitable form. This situation leads to a requirement for formula support for other unspecified formulae and for other analysis tools, and this requirement should be documented in the TSF library documentation.

Further details about the creation and use of TSF models are provided in Clause 11.

7.3 Typical TSF model

Figure 2 represents a model for a typical TSF model. The example used is for a SYNCHRO, which is described in more detail in 11.6. This model shows the input parameters (attributes) at the top and the signal output on the right. The attributes shown are control attributes. Figure 3 shows the output waveforms derived from the TSF model.

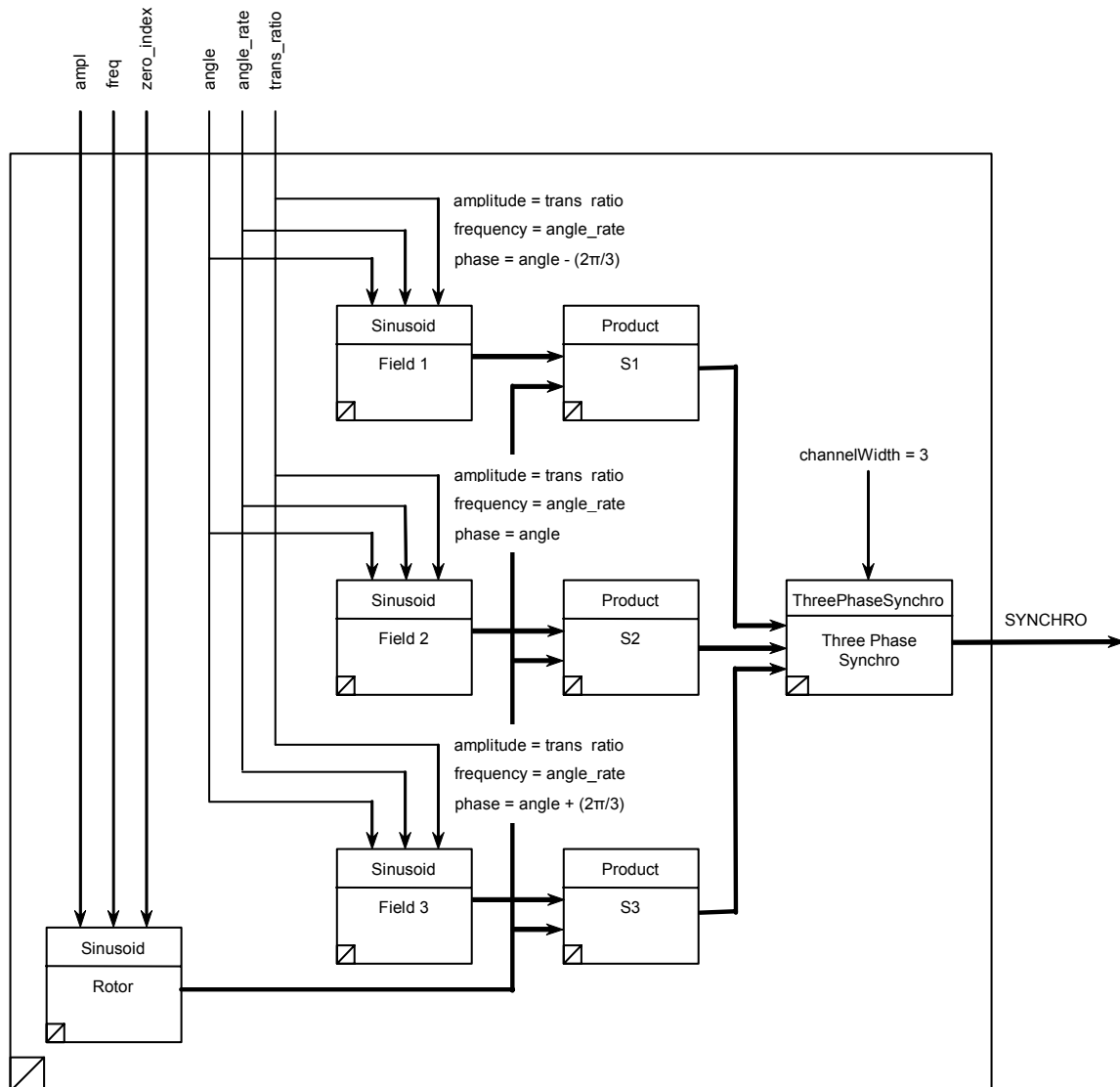


Figure 2—Annotated SYNCHRO signal TSF model

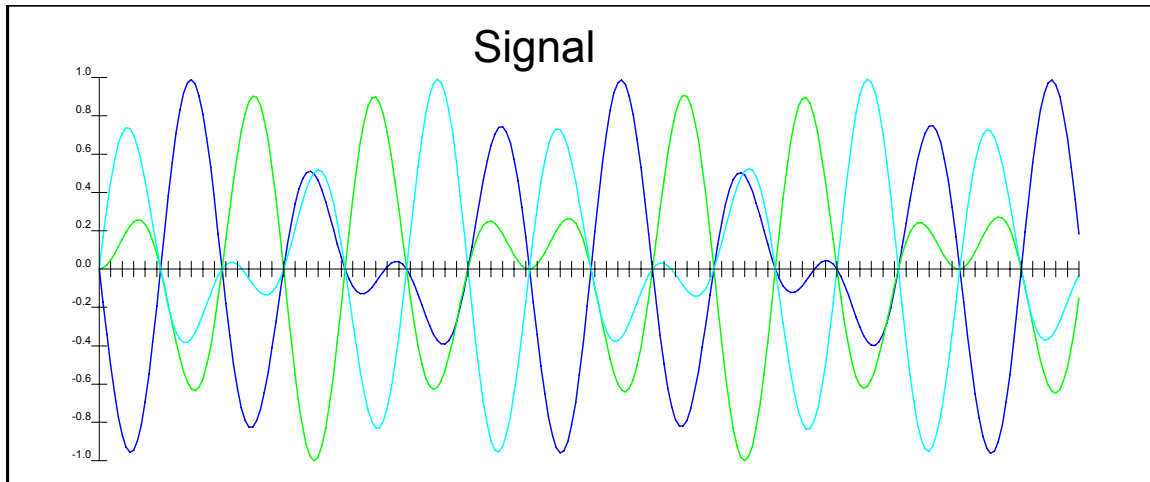


Figure 3—Output from SYNCHRO TSF model

7.4 Using TSFs in test requirement or program

Signal templates may be used in a test requirement or program by adding the test parameters to the template to create an instance of a signal. The reason to use a template is to enable the signal definition to be reused.

The STD standard does not mandate how the requirement is to be captured. It is possible to use the TPL as defined in Annex H of the STD standard. This method allows a more traditional textual definition of a program in a simple format; however, it requires a process to convert the TPL program into the test program native language (see 8.7)

The test program may also be written directly in the native language using the signals instantiated from BSCs or the TSF models (see 8.7.2.3).

Both these approaches provide the same test requirement in a procedural (dynamic) form, i.e., “do this; then do that; etc.”

Test requirements may also be captured in a nonprocedural form for use in an information or declarative (static) system, e.g., using XML descriptions such as ATML or other nonprocedural environment (see 8.7.3.6). The STD standard itself uses the nonprocedural method to define test requirements, as seen in the behavioral model of a TSF (Figure 2).

The STD standard allows the creation of “anonymous TSFs,” which represent a signal’s behavior without any interface definitions, using the Require method and XML signal definition. Anonymous TSFs support a default interface that provides a “Document Object Model” interface utilizing late binding (IDespatch).

Example:

```
mySig.DC_Offset.dc_offset = "1V"  
mySig.AC_Component.freq = "10MHz"
```

These anonymous TSFs can be used to create and define nonprocedural test requirements and can be used interchangeably within a procedural test requirement definition.

The test requirement may then be built up in a procedural or declarative manner.

7.5 Signal measurement

Thus far, TSF models have been discussed as signals. Signals and, therefore, TSFs can be equally considered sources, events, or measurements. This flexibility allows TSF models to encapsulate measurements or even complete tests. One major feature of STD measurements is known as *generic measurement*. This principle combines with TSF models so that any TSF model naturally defines its measurement requirements. It does not define the measurement method or the approach to achieving it.

Key features of generic measurements with a TSF model are the following:

- a) Generic measurement provides an inverse function for any signal library component to measure any of its control attributes.
- b) Because users can define library components, the generic measurement is naturally extended and can be used to measure any control attribute of such signal.
 - 1) As an example of this principle, consider the reading back of an RS232 message.
`<Measure As="rs232" attribute="data_word" In=.../>`
 - 2) Consider the reading back of a list of pulses from a PULSED_AC_TRAIN.
`<Measure As="PULSED_AC_TRAIN" attribute="pulse_train" In=.../>`
 - 3) Consider the measurement of the rise time of a square(ish) wave.
`<Measure As="Trapezoid" attribute="riseTime" In=.../>`
 - 4) The use of the generic measure is not restricted to measuring single attributes. Consider an example of performing multiple measurements.
`<Measure As="Sinusoid" attribute="ac_ampl phase" In=.../>`

Generic measurement is discussed in more detail in 9.4.

8. Creating signals using STD standard

8.1 Overview

The STD standard provides a means to describe and model the test requirements needed to test a UUT. In describing these test requirements, the external behavior of a test subject is defined unambiguously, by modeling the signals, attributes, and values that pertain to the UUT.

The STD standard defines entities known as *SignalFunctions*. New SignalFunctions can be created by combining existing SignalFunctions. The behavior and specification of new SignalFunctions so created is defined (in part) by their hierarchical decomposition and their subcomponent SignalFunctions. At the lowest level, SignalFunctions are BSCs. At this level, their behavior and specification are defined by the STD standard.

Interaction between SignalFunctions occurs only via the information that passes between them (see Figure 4).

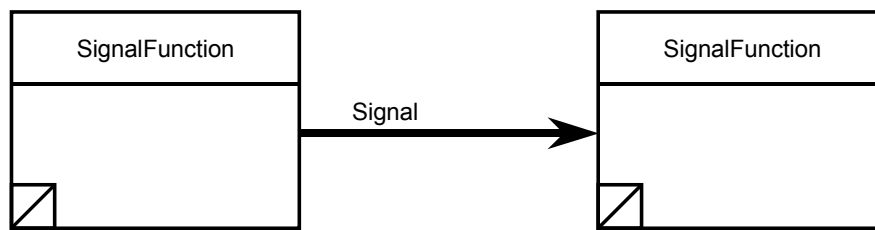


Figure 4—Signal flow between two SignalFunctions

The information transfer between SignalFunctions falls into two categories, continuous and discrete. The continuous transfer of information represents signals, and the discrete transfer of information represents events.

The behavior that a SignalFunction exhibits is controlled through attributes. The behavior is a combination of the class of the SignalFunction, such as Sinusoid (Voltage, Time), and the values of its attributes, such as frequency, amplitude, and phase. As well as describing a source of a signal, SignalFunctions can measure an observable behavior of the signal by producing values. Conceptually, control attributes help define the signal whereas value attributes provide measurable aspects of the signal. Put another way, control attributes represent inputs to the SignalFunctions whereas value attributes represent outputs from SignalFunctions. All attributes are heavily typed using the same basic types.

Figure 5 is used to illustrate a generic SignalFunction component. The concept of a SignalFunction is the key reusable component within the system. The SignalFunction provides the signal interface that transfers a signal, its attributes, and its values to other components. The SignalFunction has a reference to an input signal (**In**). The SignalFunction operates on the input signal and produces an output signal (**Out**). The behavior may also be through events (**Sync** and **Gate**). In addition, the SignalFunction has a collection of attributes that affect its behavior. An STD model depicts the passing of parameters to other components pictorially as lines.

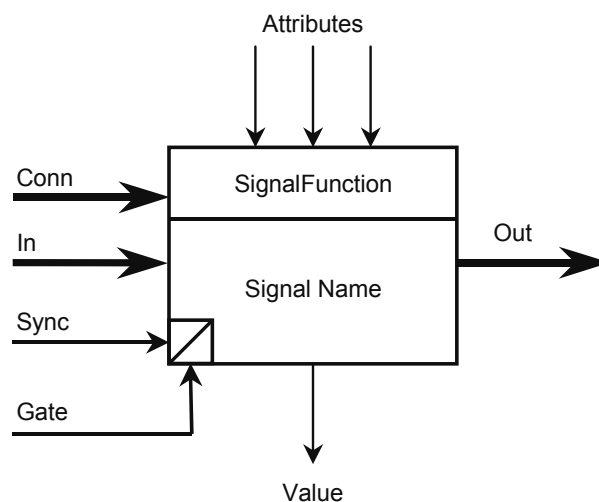


Figure 5—SignalFunction diagram

The fundamental SignalFunctions are BSCs and have formal definitions that mathematically define both the function and behavior. The concept of the STD standard is that a network of these SignalFunctions can be constructed based on a consistent set of rules of composition. These rules and the SignalFunction formal definitions allow the building of complex SignalFunctions, which are formally defined via their base components. Using these SignalFunctions and attributes, a model may be created that defines the complete signal (see Figure 6).

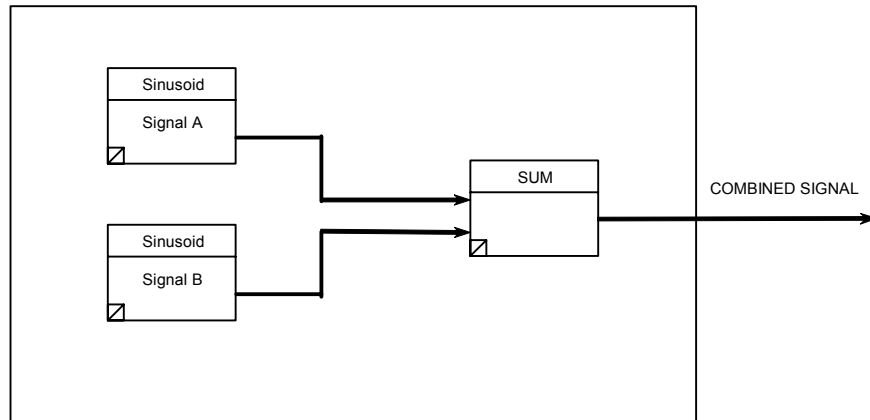


Figure 6—Simple signal summing two sinusoidal waveforms

By assigning the reference input signal to another SignalFunction output, it is possible to build up complex signal descriptions. These signal definitions become available to the application through the high-level signal model interfaces.

The STD standard deals directly with the concept of SignalFunctions and their interaction and defines the allowed set of attributes and values for each SignalFunction and their allowed types. It is the role of the carrier language program to deal with the values of every attribute and interpret the values correctly, including performing any analysis, computation, or evaluating expressions on the attributes.

In practice, some test requirements may not be fully defined without the need of some carrier language algorithms, in which case the test requirements may be nondeterministic across different language platforms. In these cases, it is important to realize that the signal descriptions will have fully deterministic behavior that is ensured by strict rules concerning event concurrency and state updates and that the nondeterministic nature lies with the different implementations of carrier languages, such as speed or variable precision.

SignalFunctions may be defined in several ways, including XML definitions, TSFs, or a carrier language utilizing the Interface Definition Language (IDL). When using a carrier language with IDL, each instance of a SignalFunction is created by a supplied application, but the SignalFunction's behavior and interface are defined through the STD standard. The purpose of the carrier language is to name and request the creation of all the necessary SignalFunctions, interconnect all of the SignalFunctions, set the attributes of the SignalFunctions, initiate the top-level sequencing (thereby, in effect, pressing the GO button), and interpreting the results.

Test specifications are built using static (functional specification) descriptions or dynamic (algorithmic) descriptions. This process necessitates that the carrier language program perform two tasks:

- a) Modeling, i.e., providing a static description of the SignalFunctions and their interconnections.
- b) Control, i.e., providing a dynamic use of the model by changing attributes and interpreting values.

8.2 Physical types

The STD standard defines a whole collection of quantity types, under a base type called Physical. These are the general engineering terms such as Time, Voltage, Resistance, and Power as well as some others that are less well used such as Amount of Information, Luminescence, Pressure, Heat, etc.

They are all equally valid, although the examples used in this guide are predominantly based on electrical signals with the default dependent (type) and independent (reftype) variables, being Voltage and Time. In general, any Physical type may be used.

The STD standard also defines the base Physical types that can be used to define any quantity in terms of a combination of unit symbols, e.g., kB/s. The definition of these physical quantities is outside the scope of the STD standard. However, when used, their definition shall match the appropriate SI⁴ or IEEE standard definition.

All physical types may take engineering unit prefixes as illustrated in the examples pF, nm, μ W, mA, cm, kg, and MHz.

When a Physical quantity is defined, e.g., 2 V, the following can also be used:

- A qualifier, such as inst, pk, trms, pk_pk, av
- A level of uncertainty, introduced by errlmt, $\pm$, etc
- A range over which the quantity may change

If this additional information is not provided, then the interpretation is that these values are unimportant. In other words, if no uncertainty value is defined, any accuracy or stability will do.

Examples:

- “25 °C” (or “25 degC”), meaning that on average the temperature should be 25 °C
- “trms 2 V $\pm$ 5% range MAX 10 V,” meaning that the true root-mean-square (trms) value must be within 5% of 2 V
- “pk 2 W errlmt 100 mW range 1 W to 3 W,” meaning that the peak power must be within 100 mW of 2 W

The range value in the last two examples indicates the extremes that the attribute value may take. Therefore, for a control attribute, it will never be programmed outside these values, and for a measurement, the value will be within the range.

⁴ Système International d’Unités (International System of Units).

Within engineering, there are several terms where it is more convenient to express the value using a different set of units. The STD standard supports this practice by allowing mappings between different units onto the defined quantity. In all cases, the quantity, regardless of how it is defined, represents the same single value; therefore, defining “32 °F” or “0 °C” defines the same value.

Some quantities provide mappings between units, e.g., a frequency value of 1 MHz is identical to a value of 1 μ s because frequency has a mapping of 1/Time. In some cases, there are relationships between mapped quantities that require additional information. An example of this situation is Power and Voltage where a value for the load resistance is required.

Only the mappings defined in the STD standard should be used because abuse of this feature makes signal verification difficult and masks user mistakes within signal definitions.

- Frequency = 1/Time
- Time = 1/Frequency
- Power = Voltage $\times$ Current
- Power = Voltage $\times$ Voltage / Resistance
- Power = Current $\times$ Current $\times$ Resistance
- Voltage = Current $\times$ Resistance

8.3 Signals

8.3.1 General

Electrical engineers often visualize signals as Voltage vs. Time; i.e., a signal is always some voltage changing in time. For most electrical signals, this view is reasonable. The question that should be asked is the following:

Are these types the best way to describe the required signal?

The STD standard allows a signal to be defined in any combination of Physical types. A signal identifies the relationship between the dependent variable (type) and independent variable (reftype). The concept is that any signal exhibiting the specified relationship will be adequate. The implication of this concept is that there may be many waveforms that will meet the STD definition of the signal. This implication falls in line with the intent of the standard:

- a) Do not over-constrain a requirement.
- b) Do not pin down a signal definition to a particular implementation, instrument, or resource.

A simple dc voltage, something that may be expressed as 5 V dc, would be described (using XML) as

```
<Constant amplitude="5 V" />
```

or more fully

```
<Constant type="Voltage" reftype="Time" amplitude="5 V" />
```

This example defines a signal whose relationship between Voltage and Time is such that the voltage amplitude is a constant 5 V. It is illustrated in Figure 7.

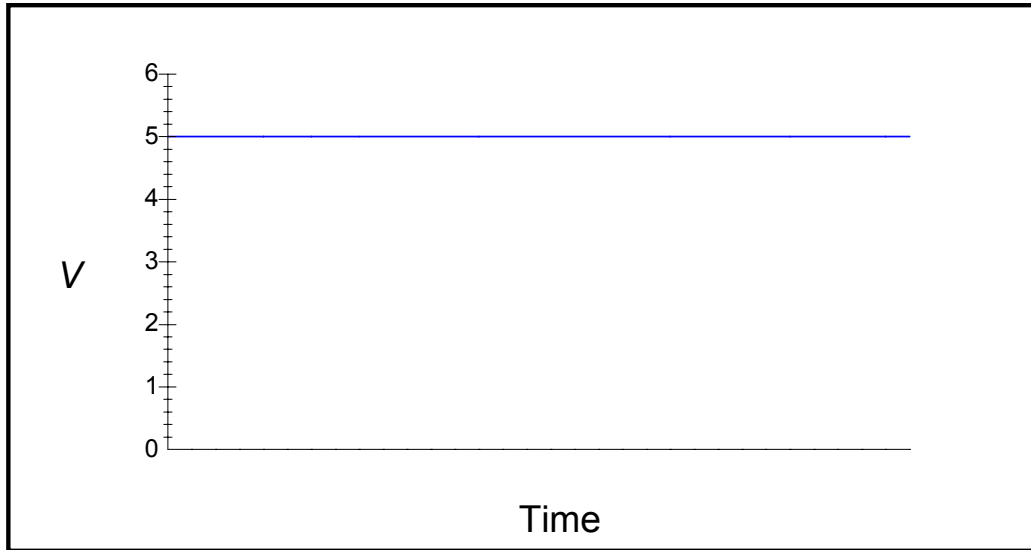


Figure 7—Constant (Voltage, Time) signal

8.3.2 Effect of qualifiers on signal definition

The definition could be expressed in a different way. The following definition could be used to describe the identical signal:

```
<Constant amplitude="trms 5 V" />
```

In both cases, the definitions describe a constant voltage with respect to time where the (constant) value of the amplitude of the voltage is expressed as trms 5 V, rather than a constant trms voltage with respect to time.

It is important to remember that using a qualifier does not change the type of the signal being defined. It just allows a quantity to be expressed in a different form.

Consider the following signal:

```
<Sinusoid type="Voltage" reftype="Time" frequency="1 MHz ±0.1 Hz"  
amplitude="trms 2 V" />
```

This example defines a signal whose relationship between Voltage and Time (the default types) is a sinusoid where the amplitude of an accurate (± 0.1 Hz) 1 MHz sinusoid is such that its rms value is 2 V. Therefore, the amplitude pk is 2.8284 V ($2\sqrt{2}$), the pk_pk value is 5.6569 V ($4\sqrt{2}$), etc. The phase of the signal starts at 0° , the default value of the phase attribute. This signal is illustrated in Figure 8.

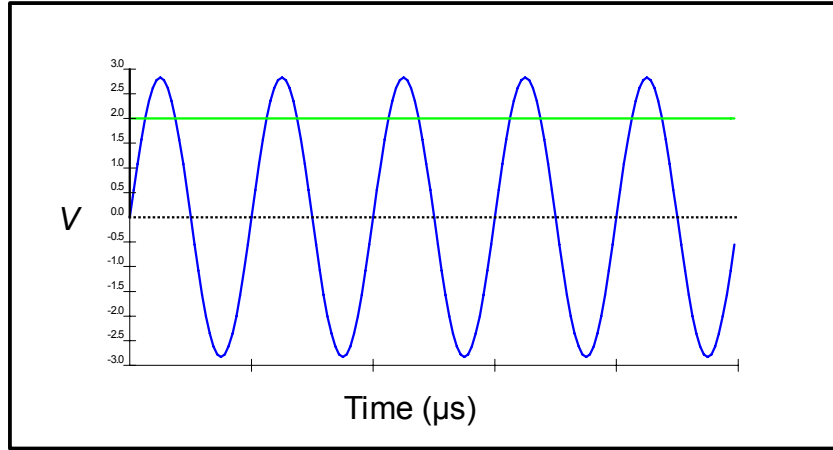


Figure 8—Sinusoid (Voltage, Time) signal

8.3.3 Phase information within signal

This example may be extended to illustrate how phase is handled by the STD standard. The signal was specified with a defined phase. If two such sinusoid signals with different phase attributes were to be summed, the resulting signal would be defined as follows:

```
<Sum In="s1 s2" />
<Sinusoid name="s1" frequency="1 MHz" amplitude="1 V" />
<Sinusoid name="s2" frequency="3 MHz" amplitude="1 V" phase="45 deg" />
```

Phase is relative between signals within the same time frame; therefore, the second sinusoid signal (s2) has a phase 45° different from the first signal (s1). This case is illustrated in Figure 9, which shows the two signals, one combined in phase and the other with a 45° phase shift.

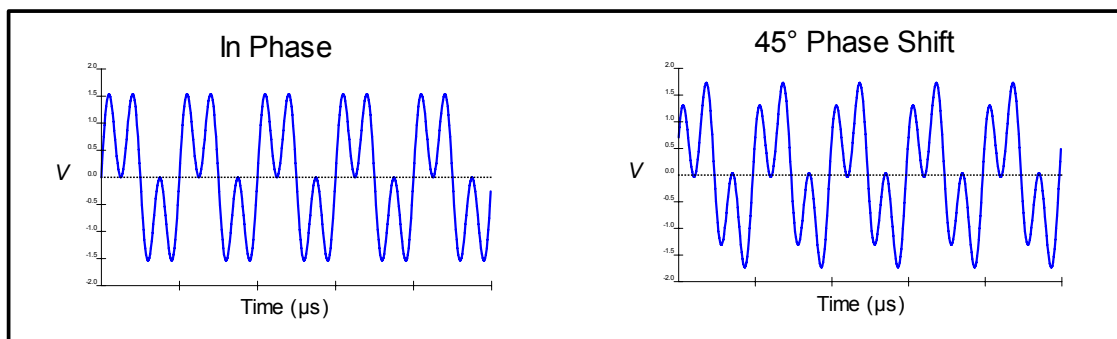


Figure 9—Example showing phase relationship on a signal

A signal may pass through different time frames. For example, when a signal goes through any derived STD connection object, the phase information of the time frame is lost. In other words, the relative phase of the signal is maintained, but the specific phase of the time frame is not guaranteed. A consequence of this

situation is that if a simple single sinusoid is applied to the UUT, defining the signal as sinusoid (0° initial phase angle) or cosine (90° initial phase angle) has little effect because at the connection the waveform that emerges is not guaranteed to start at the time t_0 . Using the signal example above, both waveforms in Figure 10 are valid at the UUT.

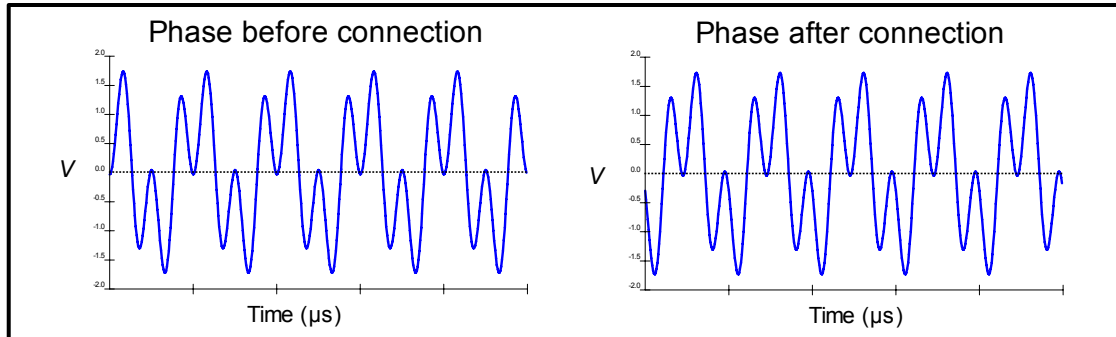


Figure 10—Effect of connection: both waveforms valid for signal

8.3.4 Signals with insufficient information

Consider signals that are expressed in terms of Power.

```
<Sinusoid type="Voltage" frequency="1 MHz" amplitude="trms 2 mW" />
```

This example defines a signal whose relationship between Voltage and Time is a sinusoid where the voltage amplitude of the sinusoid is such that the trms power (voltage $\times$ current) is 2 mW. Because it is unlikely that the signal can affect the impedance of the system, it will need to adjust the voltage, or know in advance what the resistance load is, in order to achieve this goal. The signal definition, however, does not change.

```
<Sinusoid type="Voltage" frequency="1 MHz" amplitude="inst_max 2 μW" />
```

This example defines a signal whose relationship between Voltage and Time is a sinusoid where the voltage amplitude of the sinusoid is such that the inst_max power (voltage $\times$ current) is 2 μ W. If an impedance of 50 Ω is assumed, the maximum instantaneous power of 2 μ W would indicate that a voltage amplitude of the signal at that instant would be $\sqrt{2 \mu\text{W} \times 50 \Omega}$, or 10 mV. Figure 11 shows the waveforms for instantaneous voltage and power.

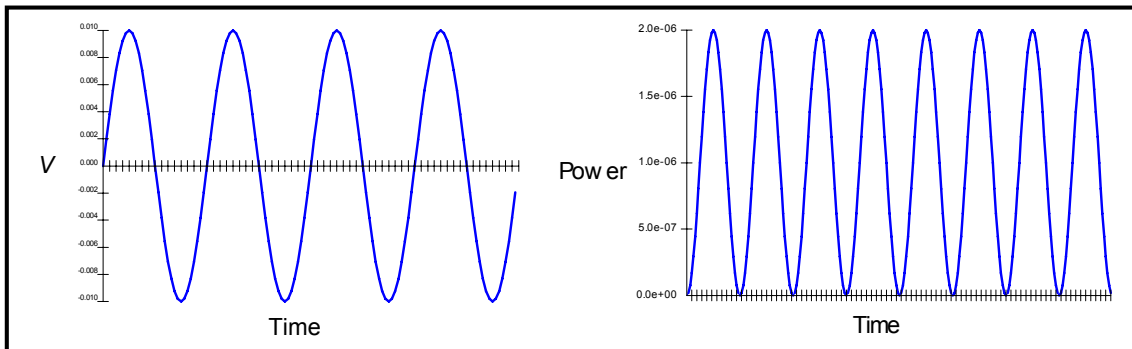


Figure 11—Relationship between instantaneous voltage and power

The above examples show that either the signal voltage has to be dynamically adjusted to arrive at the required power or the impedance must be known in advance of creating the signal. It should be noted that it may be difficult or even impossible to achieve the specified signal, e.g., if the impedance is not a linear or open circuit. Specification of stimulus signals that rely on the characteristics of the UUT for their implementation is not recommended.

All of the following describe the same signal (assuming a 50 Ω impedance).

```
<Sinusoid frequency="1 MHz" amplitude="inst_max 2  $\mu$ W" />
<Sinusoid frequency="1 MHz" amplitude="pk_pk 2  $\mu$ W" />
<Sinusoid frequency="1 MHz" amplitude="pk 1  $\mu$ W" />
<Sinusoid frequency="1 MHz" amplitude="trms 1.2247  $\mu$ W" />
<Sinusoid frequency="1 MHz" amplitude="inst_max 10 mV" />
<Sinusoid frequency="1 MHz" amplitude="inst_min -10 mV" />
<Sinusoid frequency="1 MHz" amplitude="pk_pk 20 mV" />
<Sinusoid frequency="1 MHz" amplitude="pk 10 mV" />
<Sinusoid frequency="1 MHz" amplitude="pk_neg -10 mV" />
<Sinusoid frequency="1 MHz" amplitude="trms 7.0711 mV" />
<Sinusoid frequency="1 MHz" amplitude="10 mV" />
```

Consider this example in which the signal is defined in terms of Power.

```
<Constant type="Power" reftype="Frequency" amplitude="0.1  $\mu$ W" />
```

This example defines a signal whose relationship between Power and Frequency is constant, i.e., at all frequencies, a power of 0.1 μ W exists. This example is a description of white noise. In this example, the key properties that are required have been defined, but not the method of turning it into a traditional voltage-time waveform. This ability represents a major strength when defining signals because in reality there may be many types of waveforms that will do the job and it must be possible to include them all.

8.4 Using signal graphs to create a signal

8.4.1 Introduction

BSCs are specified to be the basic building blocks required to create any signal. A TSF model is a higher level grouping of building blocks designed to create reusable blocks of signal functionality or measurement definition, as an arrangement of BSCs and/or TSFs, connected together to form a signal. A TSF model is also known as a signal graph or signal model.

There are a number of different ways in which to express signal graphs. They may be shown graphically, which is useful for visualization; XML provides the benefits of a mark-up language, such as ready parsing; and IDL provides an interface to software objects implementing signal control functionality. In all cases, the functional SML underpins all BSCs to provide consistent definition across all implementations.

8.4.2 Sources and conditioners

Sources are BSCs that create a signal according to the values of the attributes associated with it. Conditioners are BSCs that take one or more input signals and output a signal depending upon the relationship of the input signals and/or the value of the associated attributes.

Most BSCs have attributes, which are used to configure the behavior of that component. Examples of attributes are amplitude, frequency, phase, etc.

Attributes have default values that simplify their use should the designer not need to manipulate that attribute. For example, the design in Figure 12 shows sinusoid components where the phase attribute has been ignored. In this case, the result is no phase offset. Figure 12 demonstrates an additional format to present the same information. This format is also used in Figure 13, Figure 14, and Figure 15 in 8.4.3, 8.4.4, and 8.4.5, respectively.

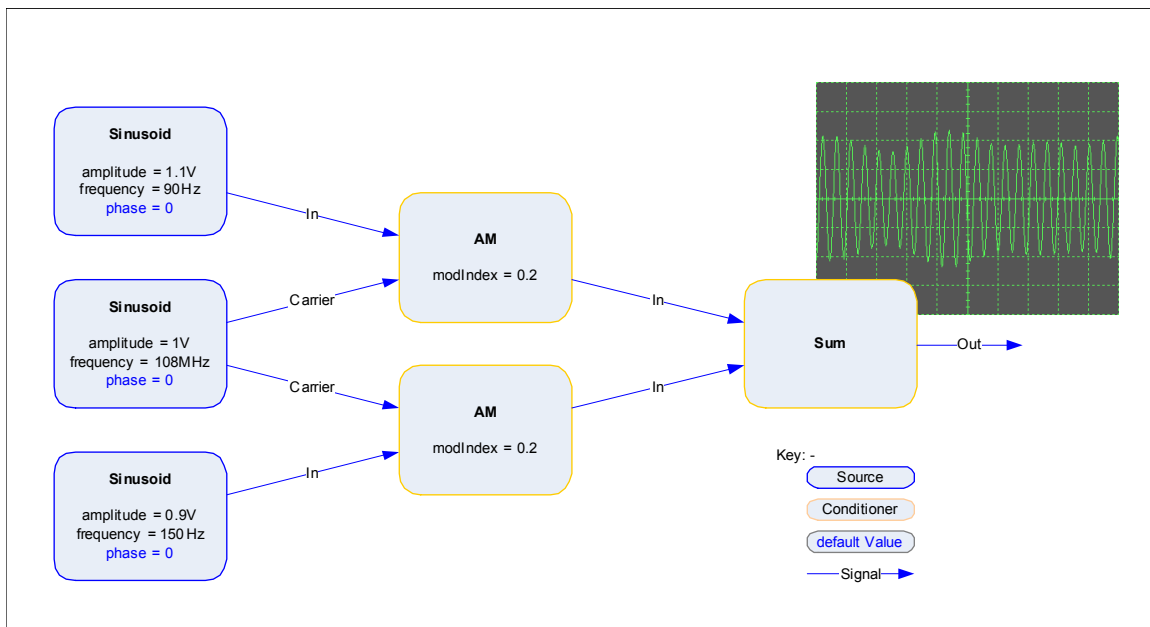


Figure 12—Sources and conditioners

The **In** connection is used to indicate that the signal generated by one BSC **Out** connection should be passed to another. This statement is true for almost all BSCs. An exception to this rule is made for the case of the carrier input to the amplitude modulation (AM) BSC, where a distinction between two different In connections must be made. Usually, as for the case of the sum conditioner, no such distinction is required.

The signal graph shown in Figure 12 represents an instrument landing system (ILS) signal. One sinusoid BSC provides a carrier signal into the two AM BSCs. The other two sinusoids provide the modulation signals for the two AM BSCs. This setup results in two separate AM signals with the same carrier frequency but with different modulation signals. These two signals are then added in the sum BSC to provide the composite signal.

8.4.3 Events and timing

Much like signal sources and conditioners, event BSCs provide and condition event signals. Events can be used to synchronize or gate signals.

- Synchronizing a signal restarts the BSC or TSF, e.g., time becomes zero.
- Gating a signal turns on the operation, i.e., for a source or conditioner, the output is on, and for a measurement, the measurement is performed over the gate.

In addition to default values, the Clock event source provides an example of a default unit. In this case, if no unit is specified, hertz are presumed. In the case of the clock BSC, it is possible to specify clockRate in seconds or hertz. The component model will automatically convert one unit to the other.

The signal graph shown in Figure 13 represents random bursts of a decaying sinusoid. The sinusoid **Out** signal is modified by the exponential conditioner, which will produce an exponentially decaying sinusoid. The effect of the **Sync** signal from the clock will be to restart the exponential each time the **Sync** event occurs, resulting in a series of decaying sinusoids. Notice also that the clock provides a signal into the ProbabilityEvent (labeled ProbabilityBased Event in Figure 13); this results in an output from the ProbabilityEvent for 50% of the inputs. The **Gate** signal from the ProbabilityEvent is used to gate the exponential BSC on and off, resulting in the random bursts of a decaying sinusoid.

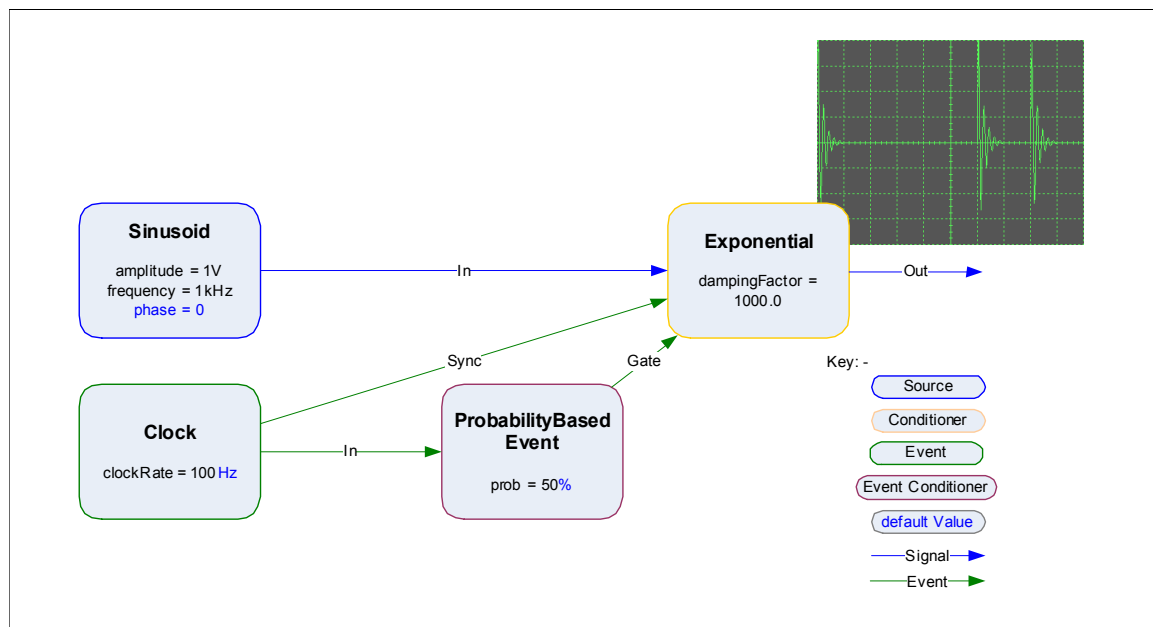


Figure 13—Events

8.4.4 Sensors and measurements

Sensors perform combinations of three main functions:

- a) A monitor function for raising an event when a particular condition is met
- b) Measurements of a required value or values of a signal
- c) Comparison of the measurement with upper and lower limits to yield a pass/fail result

Figure 14 shows a sinusoid being measured by a sensor BSC that provides the root-mean-square (rms) value of the input signal. The signal from the SingleRamp BSC is monitored by a sensor BSC that will register the maximum instantaneous value of the ramp signal. In addition, as the instantaneous value reaches 0.9 V, the monitor function will raise an event that is used to gate the sinusoid BSC on.

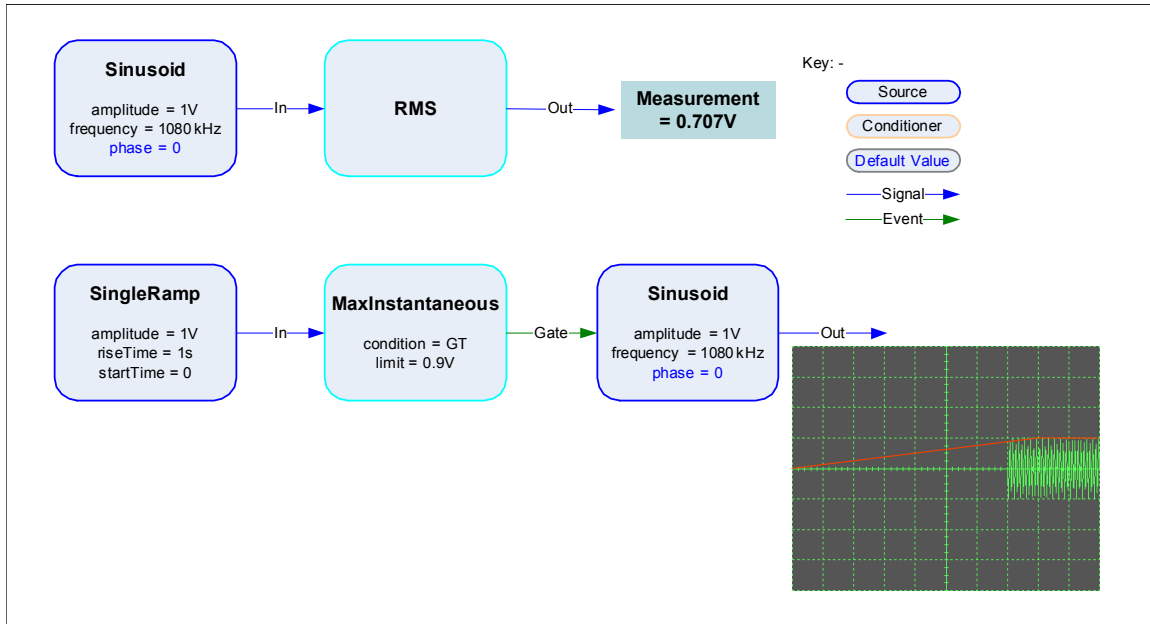


Figure 14—Measurements

When taking a measurement, the attribute value of gateTime has the following effect:

- a) +ve value: The gateTime is the specified value in time (seconds).
- b) zero: The measurement resource determines the required time.
- c) -ve value: The gateTime is determined by the gate input.

Prior to any measurement, the signal can be conditioned, e.g., by filtering the signal or demodulating the signal. Conditioning allows measurements to be made on processed signals.

Parallel measurements can be defined by using sensors in parallel. As an example to measure both rms and instantaneous frequency, it is needed only to add sensors in parallel, for both rms and instantaneous (frequency).

As an alternative measurement method, the STD standard provides the generic measure sensor. This method allows multiple TSF or BSC attributes to be measured simultaneously. The measurement may represent a multidimension series of results that can be further analyzed.

Example:

```
<Measure As="Sinusoid" attributes="amplitude phase" />
```

This example provides the simultaneous amplitude and phase measurement of the input signal. The use of user-defined TSFs and the generic measure sensor allows a wide range of measurements to be defined by the standard.

8.4.5 Connections and locations

Connections provide the method to map signals from signal graphs to real pins on a real UUT.

The STD standard provides a further group of BSCs called *connections*. Connections are used to define a point in the signal graph where the signal is required to appear in the real world, typically, as an electrical signal. The attributes of a connection are primarily pin names used to describe a physical pin or location, i.e., what would commonly be the UUT (see Figure 15).

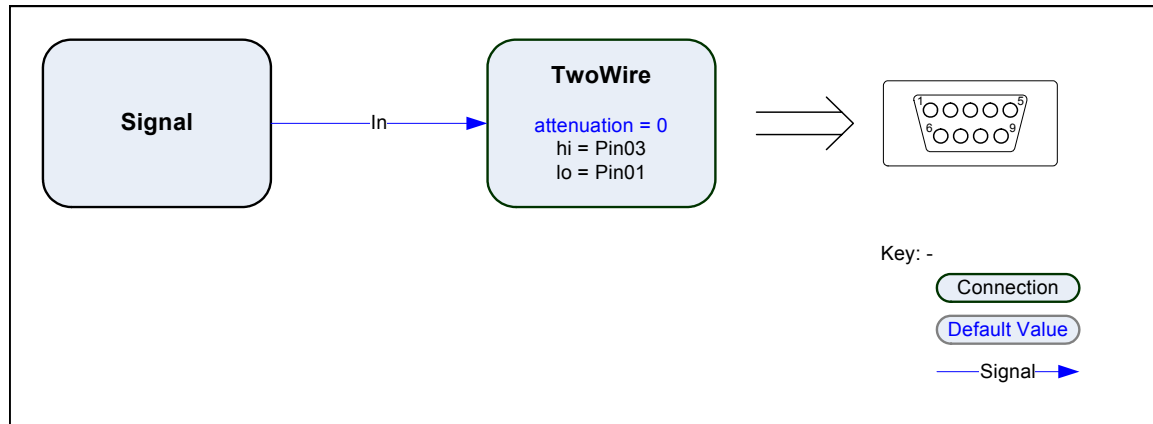


Figure 15—Connections

An additional aspect of using a connector is that any implied phase or synchronous information with respect to $t=0$ is lost across the connector. Phase and synchronism between signal components is maintained. This aspect allows TSFs to define signals that have ambiguous phase or synchronous characteristics without over-constraining a solution. As an example, consider a pulsed ac signal where the phase of the ac signal to that of the pulse may be unknown. By separating the components by a connector (with no pin information), this ambiguity is defined.

8.5 Documenting signal descriptions

The STD standard is an interface standard that identifies how to model signals. It is not a computer language. Remember that the TPL describes only signals and requires the support of a carrier language.

In order to allow consistent access to signal definitions across multiple languages, it is necessary to describe the STD classes using a language-neutral method. This approach allows each independent language to define how it accesses STD classes. In effect, each language reads the language-neutral description and converts it into something it can use (e.g., in the case of C++, a header file is used containing client classes). The medium chosen for the definition of the signal definitions is the IDL, which is defined in the Distributed Computer Environment (DCE) Specifications [B3] controlled by The Open Group.

The STD standard provides interface definitions detailing the names of all the properties that the STD classes possess. In addition, the standard provides the semantics of what each property does. The STD standard also defines the rules of use and composition between these elements and effectively defines interfaces to nonsignal types that are used by signals.

8.5.1 BSC definitions

The STD standard defines a set of fundamental SignalFunctions (i.e., the BSCs). These fundamental classes represent the elementary components available in STD and are used to define all higher level signals. For

each BSC, there exists a description in English and an SML definition to form the basis of a formal description. The SML definitions underwrite the BSCs in a rigorous mathematical way.

As well as defining the BSCs, the STD standard contains higher level signal models defined in a TSF library. The TSF provided in Annex E of IEEE Std 1641-2004 defines signals from the existing C/ATLAS standard [B10].

Clause 10 of this guide provides a more detailed description of the BSCs.

8.5.2 TSF model descriptions

TSF models are built up using the STD BSCs and by reusing existing TSF models. The TSF models represent static model descriptions with an interface that represents its attributes. A TSF model has an interface description and a static “network list” description. These may be shown in a diagram and as tables.

For each of the TSF classes, an IDL interface definition is provided. No SML definition is required because the TSF classes are constructed only from STD BSCs or other TSF models. Because the fundamental classes (BSCs) are formalized and traceable to SML definitions, the higher level signals are also defined.

The use of TSFs allows users to define their own signals and then use them in the same way as any of the BSCs. The STD standard does not mandate the use of TSFs; in fact, applications concentrate on the underlining capability of BSCs. However, TSFs provide a very powerful tool to define and support legacy signals as well as provide a vehicle for capturing domain specific signals.

It should be noted that the signal model attributed to a TSF only identifies that the TSF model can be defined; it does not mandate any particular implementation. This feature allows TSF implementations to be optimized and enables different TSF definitions to be compared.

The key use of TSF models is that, once defined, they provide an ideal vehicle through which to describe resource capability. The strengths of this approach are that resources can be configured for specific signal domains and users can match test requirement to capability. The downside is that each resource’s capability needs to be defined in terms of the TSF models it can support. In traditional instruments, this offers a practical solution; however, when considering synthetic instruments or looking at combining instruments into a complex resource, this approach requires that all the TSF models are known and mapped into resource capability.

8.5.2.1 Interface properties

The different types of property for a TSF model include the following:

- Control
- Measurement
- Input
- Output
- Expression or formula

The interface must include all inputs for the model as shown in Table 1.

Table 1—Interface properties

Description	Name	Type	Default	Range
Attribute 1	attr_name1	Physical		
Attribute 2	attr_name2	Frequency		
Attribute 3	attr_name3	Ratio		
Input Signal	input_Signal	SignalFunction		

This interface properties table provides the following information:

- Description is a short, helpful description describing the attribute.
- Name is the attribute's name with initial lowercase letter and no spaces. This information becomes the name of the attribute in the interface.
- Type is the primary type, such as a signal type, a Physical type, or a basic data type. An attribute can be of only one type, which may be of type Variant. Associated with the type may be a direction relative to the SignalFunction. Types have initial uppercase letter and no spaces.
- Default is the value given to the attribute if no value is provided by the user.
- Range is a constraint that limits the range of the values for an attribute.

The output signal and the sync and gate inputs are not shown in the interface properties table. This omission does not mean that they are not present.

Therefore, for a simple signal like an ac signal, the interface properties table could be as shown in Table 2.

Table 2—Simple interface (AC_SIGNAL)

Description	Name	Type	Default	Range
AC Signal amplitude	ac_ampl	Physical		
AC Signal frequency	freq	Frequency		

In a TSF interface properties table, the lack of a default entry indicates that the user is expected to provide a value. Normal engineering considerations require that, for an ac signal, the user will (as a minimum) state the amplitude and frequency of the signal. If the values are not provided, the default attributes of the supporting BSC will be assumed. Omitting these parameters from a signal is bad practice, as it is then not clear at a glance what the values are intended to be. Users are not expected to remember (or care about) the default attributes of a BSC.

8.5.2.2 Model description

Each TSF model has a signal model description in the form of a network list describing the interconnections. The model description table has the form shown in Table 3.

Table 3—Model

Name	Type	Terminal	Inputs	Output	Formula
Name	BSC Class	Signal [Out]		OUTPUT_SIGNAL	
		bsc_attr_1	attr_name1		
		bsc_attr_2	attr_name1		
		bsc_attr_3	attr_name1		
		bsc_attr_4			0

where

Name	is a short, helpful description identifying the specific instance of the BSC.
Type	is the class or subclass of the BSC in the model.
Terminal	is the name of the attribute or signal of the BSC.
Input	is the name of the input from the interface properties table.
Output	is the name of the BSC output signal.
Formula	provides information about the attribute, such as how the value is fixed or a calculated value if the attribute does not have an entry in the interface properties table.

For the same simple ac signal the model description table could be as shown in Table 4.

Table 4—Simple model (AC_SIGNAL)

Name	Type	Terminal	Inputs	Output	Formula
AC Signal	Sinusoid	Signal [Out]		AC_SIGNAL	
		amplitude	ac_ampl		
		frequency	freq		
		phase			0 rad

The name of the TSF model provides the class of the SignalFunction it describes. For multicomponent models, the **Sync** event input is generally connected to the last SignalFunction **Sync** event. By convention, the name of the last SignalFunction component has the same name as the TSF model.

8.5.2.3 Graphical presentation

Each TSF model diagram shows the network list in a graphical presentation where the TSF is contained within an interface box. Each component is placed within the interface box, and its class and name identified. The interconnections between signals and between attributes are shown as lines with arrows.

The inputs and outputs to the TSF interface box are the same as for an individual BSC as shown in Figure 16.

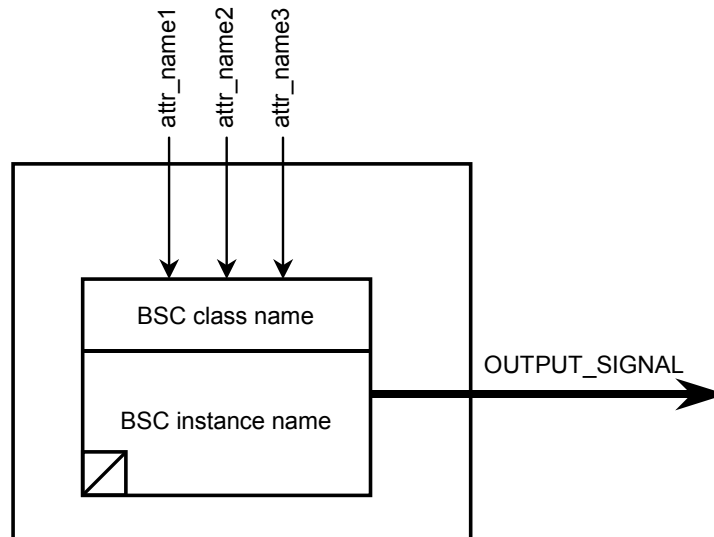


Figure 16—Generic TSF model diagram

The TSF model diagram for the simple example of an ac signal would be as shown in Figure 17.

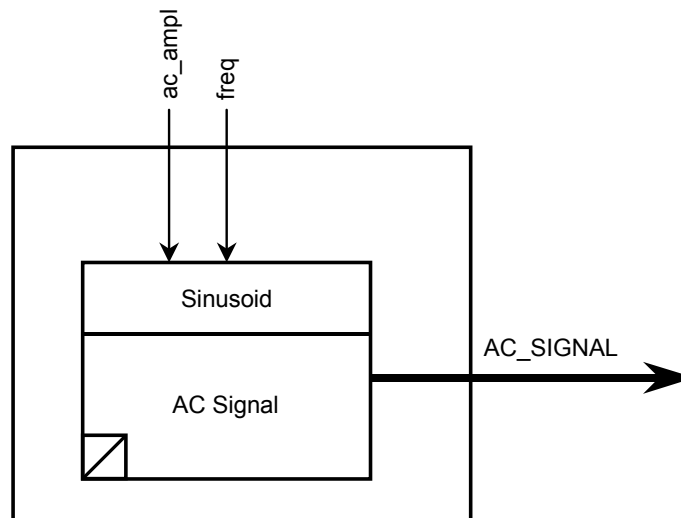


Figure 17—Simple TSF model (AC_SIGNAL)

Clause 11 of this guide provides a more detailed description of the creation and use of TSF models together with examples.

8.5.3 Interfacing STD signal descriptions to carrier language

The STD standard allows almost any language to be used as the carrier language for a signal definition environment. To achieve this feature, the signal definitions have to be made available to the carrier language in a form that is compatible with the carrier language. As previously discussed, the STD standard provides a description of the SignalFunctions, together with their controllable properties, in a language-neutral form, which can be mapped into carrier⁵ language environments. This language-neutral form (IDL) allows computer language environments (such as C/C++ or Basic) to generate their own header and input files and allows access to these signal building blocks. Some language development environments are powerful enough that they can use the IDL description directly, extend their programming features, and eliminate the need for additional interfacing files.

The STD standard allows the use of any of the commercial off-the-shelf (COTS) languages to define common signal descriptions for UUT testing. These signal descriptions, together with the programmatic information, become the UUT test requirements (see Figure 18).

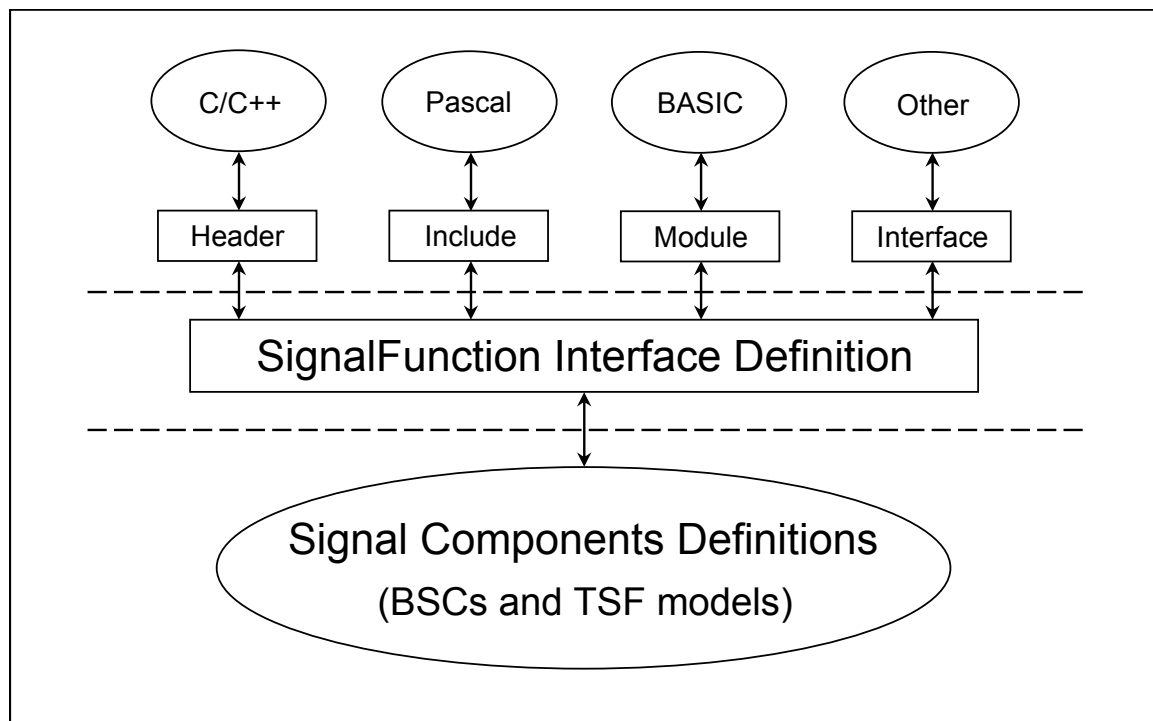


Figure 18—Multiple languages using common signal component definitions

8.6 Test requirement capture

Test requirement capture requires both of the following:

- The definition of the signals required to be applied and measured (signal definition part)
- The definition of the test sequencing and any calculations to be made (programmatic part)

⁵ There are certain criteria a carrier language must meet to support this feature.

The programmatic part is written directly in the chosen carrier language and does not require further explanation here. The signal definition part has to be extracted from the STD signal definitions and converted into carrier language code. Up to this point, the discussion has been restricted to the presentation of signal data as BSC or TSF elements. This subclause of this guide illustrates how the information may appear in a typical carrier language (see Figure 19).

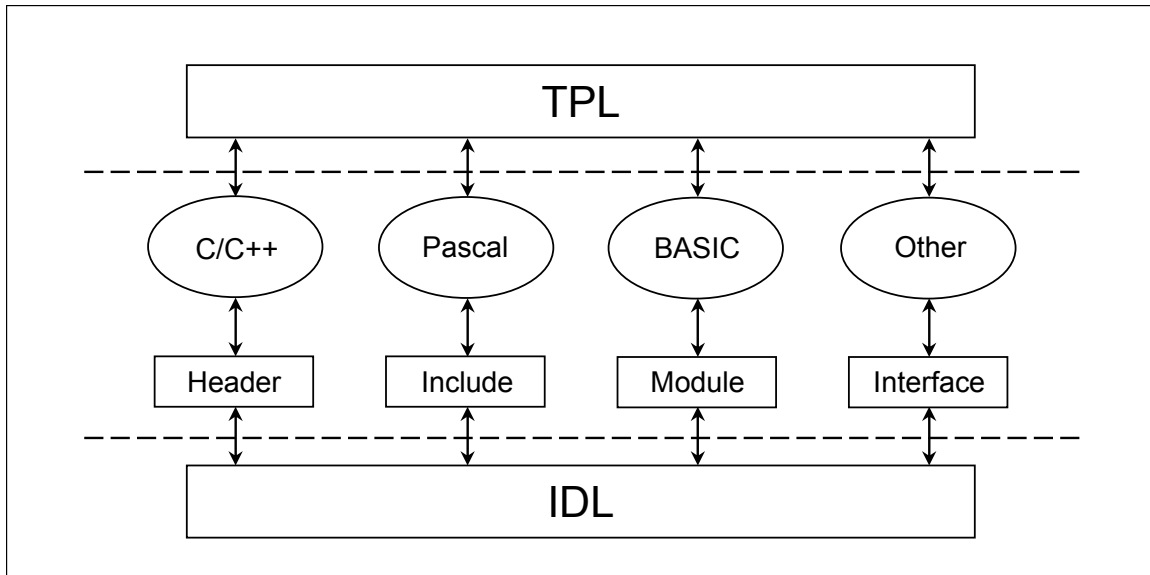


Figure 19—TPL used with any carrier language

The STD standard provides for both signal control and signal definition. Signal control is achieved programmatically. Signal definition is achieved in two ways by use of the signal building block templates:

- Using existing or predefined BSCs or TSF models from a library
- Building custom or unique signals directly using BSCs

8.6.1 Using BSC or existing TSF components

BSC and existing TSF components provide the simplest way of defining test requirements. They are used when the signal template has been defined. The test requirement becomes an instance of the TSF with the parameters entered to define a signal, a connection instance defining the location of the signal (at the UUT), and optionally an event instance defining any timing considerations.

This method is best illustrated by taking a simple example in which an ac signal with an amplitude of 1 mV and a frequency of 1 kHz is applied to the UUT. Within the carrier language, it is required to create an AC_SIGNAL object, instantiated with attributes and associated values (properties) of `ampl="1 mV"` and `freq="1 kHz"`, and then the signal has to be connected to the UUT. This process can be shown in a block diagram design (see Figure 20).

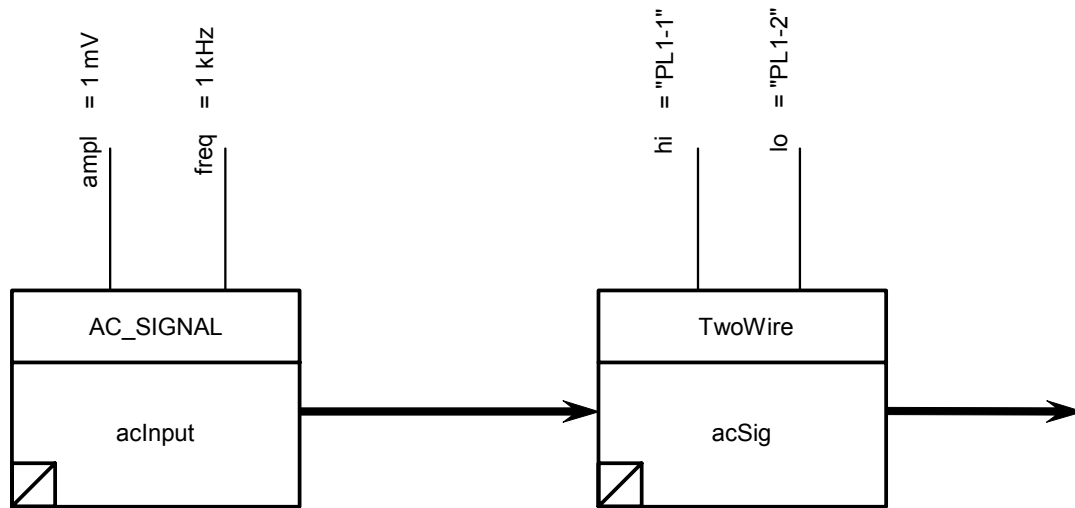


Figure 20—Simple block diagram design

8.6.1.1 TPL layer

```
Setup AC_SIGNAL ac_ampl 1 mV,
                freq 1 kHz
as source acInput;

Connect acInput(TwoWire) to hi PL1-1 lo PL1-2
as connection acSig;
```

8.6.1.2 Carrier language program (using IDL)

The test requirement can also be expressed directly in a carrier language using the IDL definitions from the STD standard. Here, the use of a ResourceManager object named STD is used, and it is assumed that it was previously declared and instantiated as a ResourceManager object in the carrier language.

```
'define acInput as an input
Set acInput = STD.Require("AC_SIGNAL")
  acInput.ac_ampl = "1 mV"
  acInput.freq = "1 kHz"

'define acSig as a UUT connection

Set acSig = STD.Require("TwoWire")
  acSig.hi = "PL1-1"
  acSig.lo = "PL1-2"

'assign the signal output to the connection input
Set acSig.In = acInput.Out

'turn the signal On
acSig.Out.Run
```

In the above example, the following statements apply:

- a) Require is a method of the resource STD, which returns a SignalFunction object of the type specified, i.e., "TwoWire" or "AC_SIGNAL".
- b) **In** and **Out** represent the signal input and the signal output to and from a SignalFunction.
- c) Run is a signal control method that turns on the signal. Run has the meaning of apply because apply is both signal setup and connection switching. A signal can be explicitly turned off, using the Stop signal control method.

8.6.1.3 Carrier language program (using C)

This test requirement may also be written using the C language. The following is a typical C implementation of the same test requirement.

```
#include "STDTSF.h"
#include "STDBSC.h"
#include <my_rte.h>

#ifndef errChk
#define errChk(fCall) if (error = (fCall), error < 0) \
{goto Error;} else
#endif

#ifndef CLEAR
#define CLEAR(__handle, __func) \
if(__handle) \
{ \
__func(__handle); \
__handle = 0; \
}
#endif

int main (int argc, char *argv[])
{

    int error = 0;
    ERRORINFO errorInfo;
    // Resource manager
    CAObjHandle STD = 0;
    // define acInput as an input
    CAObjHandle acInput = 0;
    CAObjHandle acAmpl = 0;
    CAObjHandle acFreq = 0;
    // define acSig as a UUT connection
    CAObjHandle acSig = 0;
    CAObjHandle iSignal = 0;

    if (Init MY_RTE (0, argv, 0) == 0)
        return -1; /* out of memory */

    // Create a resource manager
    errChk(STDBSC_NewIResourceManager (NULL, 1, LOCALE_NEUTRAL, 0, &STD));

    errChk( STDBSC_IResourceManagerRequire (STD, &errorInfo, "AC_SIGNAL",
CA_VariantEmpty (), &acInput));

    errChk( STDTSF_IAC_SIGNALGetac_ampl (acInput, &errorInfo, &acAmpl));
    errChk( STDBSC_IPhysicalSetmagnitude (acAmpl, &errorInfo, 1));
    errChk( STDBSC_IPhysicalSetunits (acAmpl, &errorInfo, "mV"));
    errChk( STDTSF_IAC_SIGNALSetByRefac_ampl (acInput, &errorInfo, acAmpl));
```

```
errChk( STDTSF_IAC_SIGNALGetfreq (acInput, &errorInfo, &acFreq));
errChk( STDBSC_IPhysicalSetmagnitude (acFreq, &errorInfo, 1));
errChk( STDBSC_IPhysicalSetunits( acFreq, &errorInfo, "kHz"));
errChk( STDTSF_IAC_SIGNALSetByReffreq (acInput, &errorInfo, acFreq));

errChk( STDBSC_IResourceManagerRequire (STD, &errorInfo, "TwoWire",
CA_VariantEmpty(), &acSig));
errChk( STDBSC_ITwoWireSetHI (acSig, &errorInfo, "PL1-1"));
errChk( STDBSC_ITwoWireSetLO (acSig, &errorInfo, "PL1-2"));

//assign the signal output to the connection input
errChk( STDBSC_ITwoWireSetByRefIn (acSig, &errorInfo, 0, acFreq));
//turn the signal On
errChk( STDBSC_ITwoWireGetOut (acSig, &errorInfo, 0, &iSignal));
errChk( STDBSC_ISignalRun (iSignal, &errorInfo, 0));

Error:

CLEAR (iSignal, CA_DiscardObjHandle);
CLEAR (acSig, CA_DiscardObjHandle);
CLEAR (acFreq, CA_DiscardObjHandle);
CLEAR (acAmpl, CA_DiscardObjHandle);
CLEAR (acInput, CA_DiscardObjHandle);
CLEAR (STD, CA_DiscardObjHandle);

return error;
}
```

8.6.1.4 XML version

The STD standard provides definitions so that the signal may be defined in XML format.

```
<Signal name="" Out="TW6" xmlns:tsf="STDTSF" xmlns="STDBSC" >
  <tsf:AC_SIGNAL name="AC6" ac_ampl="1mV" freq="1kHz" />
  <TwoWire name="TW6" lo="PL1-2" hi="PL1-1" In="AC6"/>
</Signal>
```

8.6.2 Building custom signals

At some point, a BSC or TSF library component that matches the required signal will not be available. If a noisy AC_SIGNAL was required because there is a need to measure the UUT tolerance to distorted signals, the AC_SIGNAL description used in 8.6.1 would not be suitable as it does not include a noise component. To overcome this problem, it is necessary only to define what a noisy signal is and then use it within the test requirement (see Figure 21). In this example, random or white noise will be assumed.

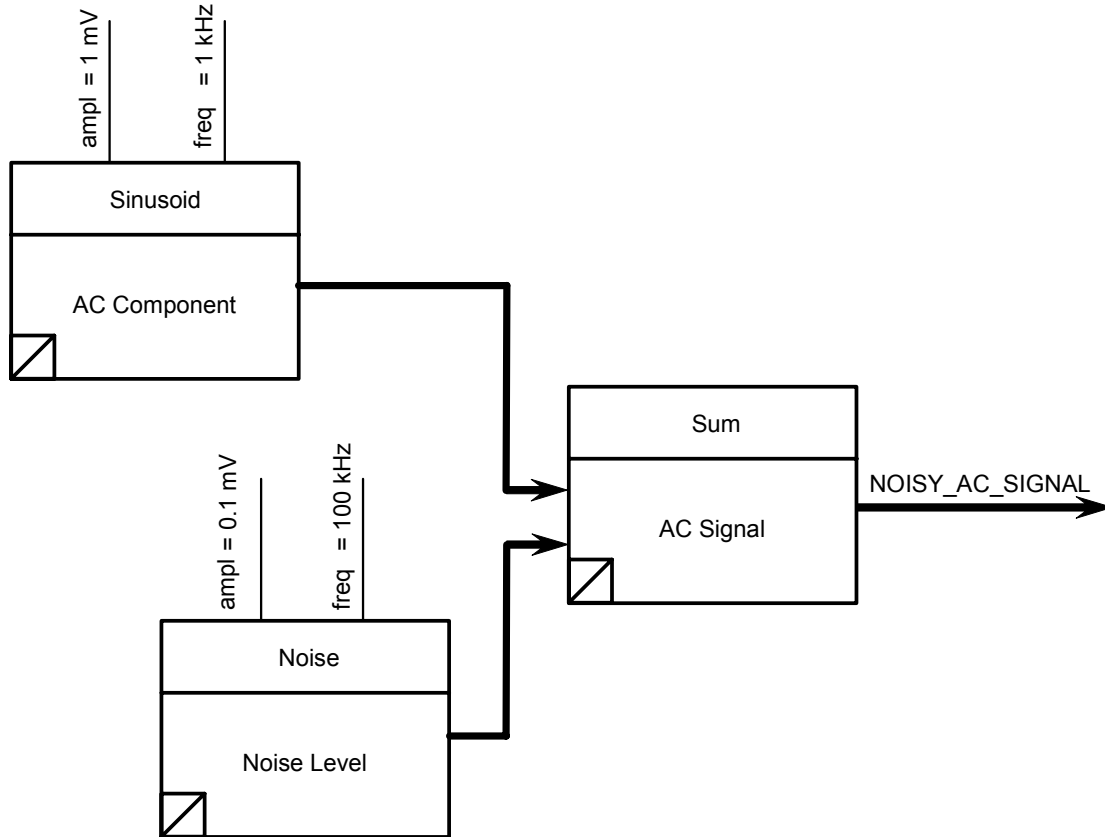


Figure 21—Noisy AC_Signal

Having defined such a signal, it can be either used directly in a test requirement or packaged into a TSF library for reuse. There is never any ambiguity of what the signal definition is because the mathematical definition in the STD standard provides a formal definition of the new signal.

8.6.2.1 TPL layer (using new TSF)

```
Setup NOISY_AC_SIGNAL ampl 1 mV,
    freq 1 kHz,
    noise_ampl 0.1 mV,
    noise_freq 100 kHz,
    as source AcInput;

Connect AcInput(TwoWire) to hi PL1-1 lo PL1-2
    as connection acSig;
```

8.6.2.2 Carrier language program (using IDL)

The test requirement can also be expressed directly in a carrier language using the STD standard and its IDL definitions.

```
'define acComponent as an input
Set acComponent = STD.Require("AC_SIGNAL")
    acComponent.ac_ampl = "1 mV"
    acComponent.freq = "1 kHz"
'define NoiseLevel as an input
Set NoiseLevel = STD.Require("RANDOM_NOISE")
```

```
NoiseLevel.ampl = "0.1 mV"  
NoiseLevel.freq = "100 kHz"  
  
'define noisyAcSig as the sum of two components  
Set noisyAcSig = STD.Require("SUM")  
    noisyAcSig.In(1)= acComponent.Out  
    noisyAcSig.In(2)= NoiseLevel.Out  
  
'define acSig as a UUT connection  
  
Set acSig = STD.Require("TwoWire")  
    acSig.hi = "PL1-1"  
    acSig.lo = "PL1-2"  
  
'assign the signal output to the connection input  
Set acSig.In = noisyAcSig.out  
  
'turn the signal On  
acSig.Out.Run
```

This can be expressed in an alternative (shorter) format using the XML format within the Require method (STD.Require):

```
'define Noisy AC_SIGNAL as sum of two components  
Set noisyAcSig = STD.Require("SUM")  
Set noisyAcSig.In(1) =STD.Require("<tsf:AC_SIGNAL ac_ampl='1mV' freq='1kHz' />")  
Set noisyAcSig.In(2) =STD.Require("<tsf:RANDOM_NOISE ampl='0.1mV' freq='100kHz' />")  
  
'define UUT connection  
Set acSig = STD.Require("<TwoWire hi='PL1-1' lo='PL1-2' />")  
  
'and connect signal to UUT input  
  
Set acSig.In = noisyAcSig.Out  
  
'Turn the signal On  
acSig.Out.Run
```

8.6.2.3 XML version

XML can also be used to define the new signal.

```
<Signal name="noisyAcSig" Out="TW6" xmlns:tsf="STDTSF" xmlns="STDBSC" >  
  <tsf:AC_SIGNAL name="AC6" ac_ampl="1mV" freq="1kHz" />  
  <tsf:RANDOM_NOISE name="NV8" freq="100kHz" ampl="0.1mV" />  
  <Sum name="Sum8" In="AC6 NV8"/>  
  <TwoWire name="TW6" lo="PL1-2" hi="PL1-1" In="Sum8"/>  
</Signal>
```

This complete XML signal definition can be used by the Require method.

```
Set noisyAcSig = STD.Require(...)  
noisyAcSig.Out.Run
```

8.6.2.4 C language version

The following is a typical C implementation of the same test requirement.

```
#include "STDTSF.h"
#include "STDBSC.h"
#include <my_rte.h>

#ifdef errChk
#define errChk(fCall) if (error = (fCall), error < 0) \
{goto Error;} else
#endif

#ifdef CLEAR
#define CLEAR(__handle, __func) \
if(__handle) \
{ \
__func(__handle); \
__handle = 0; \
}
#endif

int main (int argc, char *argv[])
{
    int error = 0;
    ERRORINFO errorInfo;
    // Resource manager
    CAObjHandle STD = 0;
    // define acComponent as an input
    CAObjHandle acComponent = 0;
    CAObjHandle acAmpl = 0;
    CAObjHandle acFreq = 0;
    CAObjHandle noiseLevel = 0;
    CAObjHandle noiseLvlAmpl = 0;
    CAObjHandle noiseLvlFreq = 0;
    CAObjHandle iSignal = 0;
    CAObjHandle noisyACSig = 0;
    CAObjHandle acCompOut = 0;
    CAObjHandle noiseLvlOut = 0;
    CAObjHandle noisyACSigOut = 0;
    //define acSig as a UUT connection
    CAObjHandle acSig = 0;

    if (Init MY_RTE (0, argv, 0) == 0)
        return -1; /* out of memory */

    // Create a resource manager
    errChk( STDBSC_NewIResourceManager (NULL, 1, LOCALE_NEUTRAL, 0, &STD));

    // define acComponent as an input
    errChk( STDBSC_IResourceManagerRequire (STD, &errorInfo, "",
    CA_VariantEmpty (), &acComponent));

    errChk( STDTSF_IAC_SIGNALGetac_ampl (acComponent, &errorInfo, &acAmpl));
    errChk( STDBSC_IPhysicalSetmagnitude (acAmpl, &errorInfo, 1));
    errChk( STDBSC_IPhysicalSetunits (acAmpl, &errorInfo, "mV"));
    errChk( STDTSF_IAC_SIGNALSetByRefac_ampl (acComponent, &errorInfo,
    acAmpl));

    errChk( STDTSF_IAC_SIGNALGetfreq (acComponent, &errorInfo, &acFreq));
    errChk( STDBSC_IPhysicalSetmagnitude (acFreq, &errorInfo, 1));
    errChk( STDBSC_IPhysicalSetunits (acFreq, &errorInfo, "kHz"));
```

```
errChk( STDTSF_IAC_SIGNALSetByReffreq (acComponent, &errorInfo, acFreq));

errChk( STDBSC_IResourceManagerRequire (STD, &errorInfo, "NOISE",
CA_VariantEmpty(), &noiseLevel));

errChk( STDBSC_INoiseGetamplitude (noiseLevel, &errorInfo,
&noiseLvlAmpl));
errChk( STDBSC_IPhysicalSetmagnitude (noiseLvlAmpl, &errorInfo, 0.1));
errChk( STDBSC_IPhysicalSetunits (noiseLvlAmpl, &errorInfo, "mV"));
errChk( STDBSC_INoiseSetByRefamplitude (noiseLevel, &errorInfo,
noiseLvlAmpl));

// define NoiseLevel as an input
errChk( STDBSC_INoiseGetfrequency (noiseLevel, &errorInfo,
&noiseLvlFreq));
errChk( STDBSC_IPhysicalSetmagnitude (noiseLvlFreq, &errorInfo, 100));
errChk( STDBSC_IPhysicalSetunits( noiseLvlFreq, &errorInfo, "kHz"));
errChk( STDBSC_INoiseSetByReffrequency (noiseLevel, &errorInfo,
noiseLvlFreq));

// define noisyAcSig as the sum of two components
errChk( STDBSC_IResourceManagerRequire (STD, &errorInfo, "SUM",
CA_VariantEmpty(), &noisyACSig));
errChk( STDTSF_IAC_SIGNALGetOut (acComponent, &errorInfo, 0, &acCompOut));
errChk( STDBSC_INoiseGetOut (noiseLevel, &errorInfo, 0, &noiseLvlOut));

errChk( STDBSC_ISumSetByRefIn (noisyACSig, &errorInfo, 1, acCompOut));
errChk( STDBSC_ISumSetByRefIn (noisyACSig, &errorInfo, 2, noiseLvlOut));

errChk( STDBSC_IResourceManagerRequire (STD, &errorInfo, "TwoWire",
CA_VariantEmpty(), &acSig));
errChk( STDBSC_ITwoWireSetHI (acSig, &errorInfo, "PL1-1"));
errChk( STDBSC_ITwoWireSetLO (acSig, &errorInfo, "PL1-2"));

errChk (STDBSC_ISumGetOut (noisyACSig, &errorInfo, 0, &noisyACSigOut));
//assign the signal output to the connection input
errChk( STDBSC_ITwoWireSetByRefIn (acSig, &errorInfo, 0, noisyACSigOut));
//turn the signal On
errChk( STDBSC_ITwoWireGetOut (acSig, &errorInfo, 0, &iSignal));
errChk( STDBSC_ISignalRun (iSignal, &errorInfo, 0));
```

Error:

```
CLEAR (iSignal, CA_DiscardObjHandle);
CLEAR (acSig, CA_DiscardObjHandle);
CLEAR (acCompOut, CA_DiscardObjHandle);
CLEAR (noiseLvlOut, CA_DiscardObjHandle);
CLEAR (noisyACSigOut, CA_DiscardObjHandle);
CLEAR (acFreq, CA_DiscardObjHandle);
CLEAR (acAmpl, CA_DiscardObjHandle);
CLEAR (acComponent, CA_DiscardObjHandle);
CLEAR (noiseLvlAmpl, CA_DiscardObjHandle);
CLEAR (noiseLvlFreq, CA_DiscardObjHandle);
CLEAR (noiseLevel, CA_DiscardObjHandle);
CLEAR (noisyACSig, CA_DiscardObjHandle);
CLEAR (STD, CA_DiscardObjHandle);

return error;
}
```

8.7 Using TPL to create test requirements

The TPL layer provides a mechanism for users who want to document test requirements in a textual format. The TPL provides verbs giving users the ability to manipulate test objects (such as signal definitions contained within the TSFs) in test procedures or signal sequences.

In order to be able to create a test requirement in any carrier language and yet still have a standardized method of defining signals, the STD standard defines this TPL. Users can define test requirements in terms of these TPL test objects embedded within the carrier languages. In this way, different languages may use their own syntax and language semantics and still define the same test requirement because they reference the same external test objects.

Within a TPL environment, there will be some translation mechanism to convert from this neutral format of the signal statements into the preferred carrier language format before the test statements can be compiled and executed. The examples in 8.7.1 through 8.7.3 provide a background of how the TPL maps into a typical carrier language.

8.7.1 Examples of test requirements using TPL

The following examples show different test requirements expressed in TPL, a COTS programming environment, and in ATLAS (specifically IEEE 716 C/ATLAS). The examples serve to illustrate how the STD standard can be implemented to define UUT test requirements using a variety of platforms and languages. The TPL examples include references to “preamble” and “procedural” parts, which have been included to assist with the comparison to the ATLAS versions.

8.7.2 DC amplifier example

In this example, the UUT test requirements are defined as follows:

- a) Apply power 28 V dc between pins PL1-1 and PL1-2.
- b) Apply power 20 V dc between pins PL1-3 and PL1-4.
- c) Apply a dc voltage of 0.5 V between pins J1-1 and J1-2.
- d) Measure the voltage across output pins J2-1 and J2-2.
- e) Calculate the voltage gain between the 0.5 V input and the measured output.

Figure 22 represents a diagram of the UUT.

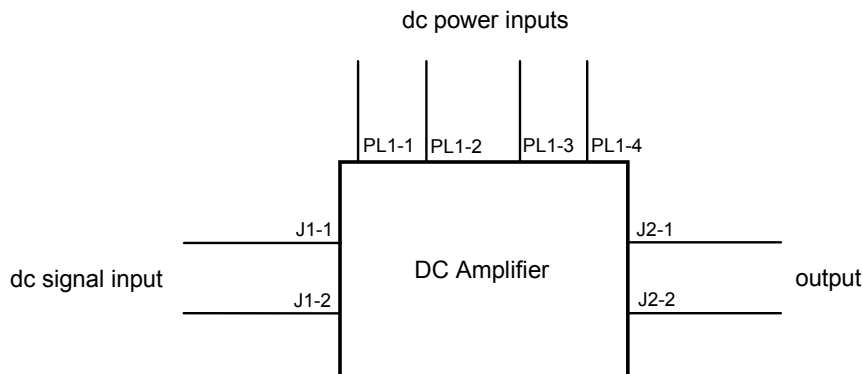


Figure 22—DC amplifier example

8.7.2.1 TPL within carrier language

This example shows how to express the dc amplifier requirement using an undefined carrier language. The lines of code in italic text would be expressed in a user-selected carrier language. The example test requirement shows how to mix TPL signal definitions with procedural and output statements using the carrier language.

```
/* *****  
/* Test Methodology: TPL  
/* UUT Type: DC Amplifier  
/* Model: DC-01 */  
/* *****  
/* In this Example:  
/* the Roman Text is Test Procedure Language Code  
/* the Italic Text is Carrier Language (including comments) */  
//  
/* *****  
/* Begin the preamble part */  
/* *****  
//  
single gain  
test PowerUp(single Vin-1, Vin-2)  
{  
/*<TPL>  
Setup DC_SIGNAL dc_ampl Vin1 range 16 V to 32 V  
as source Power1 ;  
Setup DC_SIGNAL dc_ampl Vin2 range 16 V to 32 V  
as source Power2 ;  
Connect Power1 to hi PL1-1 lo PL1-2;  
Connect Power2 to hi PL1-3 lo PL1-4;  
/<TPL> */  
return() ;  
}  
//  
test GainTest(single Vin)  
{  
single Vout;  
/*<TPL>  
Setup DC_SIGNAL ampl Vin range 0 V to 20 V  
as source SignalIn ;  
Setup DC_SIGNAL ampl range 0 V to 30 V  
as sensor dcVoltMeter ;  
Connect SignalIn to hi J1-1 lo J1-2;  
Connect hi J2-1 lo J2-2 to dcVoltMeter;  
Read dcVoltMeter into Vout;  
/<TPL> */  
return(Vout/Vin);  
}  
//  
/* *****  
/* Begin the procedural part  
/* Start testing the UUT */  
/* *****  
//  
output("Start of TPL Program 'DC_AMP' " ) ;  
powerUp(28.0, 20.0);  
gain=GainTest(0.5);  
output("Gain is",gain);  
/*<TPL>  
remove all;  
/<TPL>*/
```

```
output("End of TPL Program 'DC_AMP' " ) ;
//
/*****
/* End of testing
*****/
```

8.7.2.2 Embedded TPL in native language

This example shows an optimized requirement for the dc amplifier using a carrier language. It illustrates how Disconnect or Reset statements may be omitted at the end of a program. The TPL verbs create SignalFunctions and control them through their signal control interface. Within the TPL, signals are turned on with an internal statement like "<object>.Out.Run". To remove or disconnect the signal, an equivalent "<object>.Out.Stop" may be used. All SignalFunction objects are deleted when a program exits. The effect is that as a program exits, it tidies up all variables and references to SignalFunctions. This step, in turn, informs the SignalFunction implicitly that it is no longer used, then resets any signals, and thus disconnects or removes the signal or connection.

The TPL statements are introduced and annotated via carrier language comments, which for this carrier language are prefixed by the single quote character (').

```
*****
'    Test Methodology: TPL
'    UUT Type:         DC Amplifier
'    Model:            DC-01
*****
' In this Example:
'    the Roman Text is Test Procedure Language Code
'    the Italic Text is Carrier Language (including comments)
*****
' Begin the preamble part (containing declarative statements)
*****
'
Subroutine PowerUp Vin-1, Vin-2
'<TPL>
'Setup DC_SIGNAL ampl Vin-1 range 16V to 32V as source Power1;
'Setup DC_SIGNAL ampl Vin-2 range 16V to 32V as source Power2;
'Connect Power1 to hi PL1-1 lo PL1-2;
'Connect Power2 to hi PL1-3 lo PL1-4;
'</TPL>
End Subroutine
'
Subroutine PowerDown
'<TPL>
'Disconnect Power1;
'Disconnect Power2;
'Reset Power1;
'Reset Power2;
'</TPL>
End Subroutine
'
Function GainTest(Vin) as Double
'<TPL>
'Setup DC_SIGNAL ampl Vin range 0V to 20V as source Signal-In;
'Setup DC_SIGNAL ampl range 0V to 30V as sensor DC-VoltMeter;
'Connect SignalIn to hi J1-1 lo J1-2;
'Connect hi J2-1 lo J2-2 to dcVoltMeter;
'Read dcVoltMeter into Vout;
'</TPL>
GainTest = Vout/Vin
End Function
'
```