

IEEE ABBET™

IEEE Standard for Software Interface for Resource Management for A Broad-Based Environment for Test (ABBET)

Sponsor

IEEE Standards Coordinating Committee 20

Approved 19 March 1998

IEEE-SA Standards Board

Abstract: The services needed to access and manage descriptive information about resources in an automatic test system (ATS) are covered. This information includes data about the automatic test equipment (ATE) instruments, switching, and the test subject adapter. This standard is a component of the ABBET set of standards.

Keywords: A Broad-Based Environment for Test (ABBET), automatic test equipment, automatic test system, resource management

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

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

ISBN 0-7381-0180-X

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. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

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

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

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

Comments on standards and requests for interpretations should be addressed to:

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

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

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (508) 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 1226.3-1998, IEEE Standard for Software Interface for Resource Management for A Broad-Based Environment for Test (ABBET).]

In the course of developing IEEE Std 1226-1993, A Broad-Based Environment for Test (ABBET), it was determined that multiple standards would be required to define the various ABBET functions and interfaces. These standards consist of the Overview and Architecture Standard, numbered 1226, and a set of related component standards, numbered 1226.1 through 1226.x. This component standard, numbered 1226.3, documents the software interfaces defined by ABBET to test equipment configuration information.

At the February, 1991, ABBET meeting, it was decided to document the automatic test equipment (ATE) system and instrument control in a single component standard. Deliberations subsequent to that meeting, however, revealed that combining the groups of interfaces into a single component standard would force implementations of the instrumentation interfaces to conform to a subset of that standard. Therefore, it was concluded that the interface groups should be documented in separate component standards, and that these standards would naturally divide at the interface between Test Equipment Description Language (TEDL) and Resource Description Language (RDL). At the Scottsdale, Arizona, meeting in May, 1991, this decision was changed, as it was decided to create three component standards for the ATE system layer: one for the resource management interfaces (referencing TEDL and RDL), one for the instrument interfaces (referencing SCPI, CIIL, etc.), and one for the instrument control bus interfaces (referencing IEEE 488, VXI, Ethernet, etc.).

This standard is intended to document the interfaces to services to ATE descriptive information that would allow bridging the gap between an ABBET ATLAS level test procedure and an ATE (referred to in this standard as “ATE interface services”).

Participants

At the time this standard was completed, the ABBET Subcommittee of the IEEE Standards Coordinating Committee (SCC20) had the following membership:

Guy Adam	Jay Grove	William P. Neblett
Fabio Angeli	Joel Guerra	David L. Nichols
James D. Baker	Wes Hand	Hans Obermeier
Jean-Yves Barbier	Arthur Hann	Leslie A. Orlidge
Darrell Bartz	Gary L. Hardenburg	Ed Perkins
Anne Bautista	Robert G. Hayes	Jean Pouilly
Aldo Bernal	Leonard Haynes	N. Ramachandran
Michael Blair	Bill Heather	William E. Reeves
Greg Bowman	John E. Heiser	Bill Rodriguez
Joanne Buckley	Shankar Hemmady	David Roggendorff
Toni Bunsen	Ashley Hulme	David Rollince
Michael Carpenter	Ulla Jacobson	Eric Sacher
E. Elaine Cashar	Patrick Johnson	Howard Savage
Alexander Cherfas	Andrew Joskowski	Helmut Scheibenzuber
Richard D. Churcher	Al Karalius	Rick Schippang
Gregory W. Darnell	Mark Kaufman	Michael L. Seavey
Bernard Dathy	Lars-Olov Larsson	John W. Sheppard
Danny C. Davis	Hans Lecluyse	Lee A. Shombert
Hal Davis	Robert Loretz, Jr.	William R. Simpson
Jose DeLaCruz	Michael Lougue	Butch Sneade
Bernd Dinklage	Richard Maguire	Joseph J. Stanco
Bernard Dugas	Michael A. Malesich	Jack Taylor
Jerome Elisha	Michael S. Martin	Kirk D. Thompson
Keith Ellis	Gayle Matysek	Janette Tuttle
Michael T. Ellis	Mark McCooey	Jorg Urban
Paul Erath	Robert L. McGarvey	Kenneth Van
Anthony Fitzpatrick	Harry McGuckin	J. Richard Weger
William Brit Frank	Scott Miller	Daniel H. Weiss
Michael Golas	James M. Mills	William A. Whitaker
Roger S. Goldberg	Mukund U. Modi	Will W. Young
Arnold M. Greenspan		Jehuda Ziegler

In addition, the following people made significant contributions to this document:

Maria Bostic	Ron DeBlanc	John Leuthe
Dez Cass	John Goddard	Rafael Martinez
Jack Cross	Chris Gorringer	Charles Morris
Robert J. Curry	T. Winston Griffin	Ron Taylor
David L. Dawson	Gary Hinkle	Mark J. Thullen
	Glenn Laflin	

The following persons were on the balloting committee:

Guy Adam	Arnold M. Greenspan	Leslie A. Orlidge
James D. Baker	Gary L. Hardenburg	Richard J. Patrick
Jean-Yves Barbier	Robert G. Hayes	Dave Pawson
Jeffrey S. Barkley	John E. Heiser	Dennis C. Peterson
Anne F. Bautista	J. Luis Hernandez	Alain Plassart
Michael Blair	Catherine Hogans	Jean Pouilly
Greg Bowman	Ashley Hulme	William E. Reeves
Alan L. Bridges	Paul B. Knese	David Roggendorff
David M. Brooks	Scott A. Kotler	Eric Sacher
Theodor Calin	James L. Lewis	Howard Savage
E. Elaine Cashar	Richard J. Maguire	Michael L. Seavey
Charles R. Childress	Michael A. Malesich	John W. Sheppard
Richard D. Churcher	Michael S. Martin	William R. Simpson
George A. Clarius	Rafael Martinez	Philip T. Smith
James E. Cottrell	Gayle Matysek	Joseph J. Stanco
Fred L. Cox	Robert L. McGarvey	Robert Stasonis
Robert J. Curry	Thomas J. McGrath	Joseph L. Stautberg
Gregory W. Darnell	Douglas W. McNabb	Kirk D. Thompson
Bernard Dathy	James M. Mills	Louis Y. Ungar
Danny C. Davis	Mukund U. Modi	J. Richard Weger
Somnath Deb	Charles Morris	Daniel H. Weiss
Bernd Dinkluge	James Nagy	William A. Whitaker
Horst D. Ey	Richard E. Neese	Ronald T. Wolfe
William Brit Frank	David L. Nichols	Werner K. Wolz
Alfred S. Gilman	Hans Obermeier	Will W. Young
Arthur H. Greenberg		Robert W. Zimbardi

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

Richard J. Holleman, *Chair*

Donald N. Heirman, *Vice Chair*

Judith Gorman, *Secretary*

Satish K. Aggarwal	James H. Gurney	L. Bruce McClung
Clyde R. Camp	Jim D. Isaak	Louis-François Pau
James T. Carlo	Lowell G. Johnson	Ronald C. Petersen
Gary R. Engmann	Robert Kennelly	Gerald H. Peterson
Harold E. Epstein	E. G. "Al" Kiener	John B. Posey
Jay Forster*	Joseph L. Koepfinger*	Gary S. Robinson
Thomas F. Garrity	Stephen R. Lambert	Hans E. Weinrich
Ruben D. Garzon	Jim Logothetis	Donald W. Zipse
	Donald C. Loughry	

*Member Emeritus

Contents

1. Overview.....	1
1.1 Scope.....	1
1.2 Purpose.....	1
1.3 Application.....	1
2. References.....	2
3. Definitions and acronyms	2
3.1 Definitions	2
3.2 Acronyms.....	5
4. Description.....	5
4.1 Objectives	5
4.2 Overview.....	5
4.3 Relationship to ABBET architecture	6
4.4 Relationship to reference documents	9
5. Conformance.....	9
5.1 Minimum resource management interface functionality	9
5.2 Subsets	9
5.3 Extensions.....	9
5.4 Approach for implementation of subsets	10
5.5 Approach for implementation of extensions.....	10
6. Specification	11
6.1 Conventions	11
6.2 Virtual resource management services	13
6.3 Test equipment configuration management services.....	26
6.4 Real resource management services	32
Annex A (informative) Examples.....	47
Annex B (informative) Object-oriented analysis.....	62
Annex C (normative) Interface Definition Language (IDL) specification	71
Annex D (informative) Bibliography	86
Annex E (informative) Test foundation framework.....	87

IEEE ABBET™

IEEE Standard for Software Interface for Resource Management for A Broad-Based Environment for Test (ABBET)

1. Overview

The set of standards called “A Broad-Based Environment for Test (ABBET)” was developed to serve as a standards set for product testing. ABBET specifies a comprehensive environment for integrating design data, test strategies and requirements, test procedures, test results management, and test system implementations.

1.1 Scope

This standard is a component of the ABBET set of standards and covers the services needed to access and manage descriptive information about resources in an automatic test system (ATS). This information includes data about the automatic test equipment (ATE) instruments, switching, and the test subject to ATE adapter.

1.2 Purpose

This standard has a dual purpose. The first purpose is to define the interfaces to services that obtain descriptive capability and control information about an ATS. The second purpose is to define the interfaces to services that perform virtual and real resource management.

1.3 Application

This standard is primarily intended for use by ATS customers, ATE system developers and maintainers, and test program set (TPS) developers and maintainers. This standard will also be of interest to ABBET tool developers and test instrument developers.

2. References

This standard shall be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision will be considered for incorporation into this standard.¹ Additional documents used in the development of this standard are listed in Annex D.

IEEE Std 100-1996, IEEE Standard Dictionary of Electrical and Electronic Terms.²

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

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

OMG CORBA/IIOP Revision 2.1, The Common Object Request Broker: Architecture and Specification.³

IEEE P1226/D15.3, Draft Standard for A Broad-Based Environment for Test (ABBET) Overview and Architecture.⁴

IEEE P1226.4/D2.1, Draft Standard for Software Interface for Instrument Drivers for A Broad-Based Environment for Test (ABBET), 15 May 1996.

IEEE P1226.11/D4.1, Draft Standard for Test Resource Information for A Broad-Based Environment for Test (ABBET), 15 May 1998.

3. Definitions and acronyms

3.1 Definitions

IEEE Std 1226-1993 provides a complete glossary with definitions of all the terms used in the ABBET set of standards and guides. A clear understanding of the following terms with respect to an ATE system is particularly important in order to understand this ABBET component standard.

3.1.1 allocate: The process of identifying a resource for a particular task and making it unavailable to other users. The allocate process can occur during or prior to test execution. If a virtual resource is involved, the process of physical mapping is also included. *Contrast:* **deallocate.**

3.1.2 automatic test equipment (ATE): Test, measurement, diagnostic, and automation equipment that executes a sequence of manual or automated instructions that performs tests of functional or static parameters in order to evaluate the degree of performance degradation, or to perform fault isolation of unit malfunctions. Also included in ATE are the software packages required for the equipment to perform its functionality.

3.1.3 automatic test system (ATS): Includes the ATE as well as all support equipment, support software, test programs, and interface adapters.

¹The capabilities of evolving reference standards will be integrated as they serve to improve the ABBET environment.

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

³This document is available from the Object Management Group (OMG), 492 Old Connecticut Path, Framingham, MA 01701, USA.

⁴Numbers preceded by P are IEEE authorized standards projects that were not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining drafts, contact the IEEE.

3.1.4 behavior: A relationship between the inputs and the outputs of a product. As specified in ABBET terms, however, it is neither implementable nor testable until it has been specified in terms of signals that are observable or controllable. With respect to this standard, behavior is the observable and controllable signals that are used as inputs or outputs of a resource or path.

3.1.5 class: A description of a group of objects (or types) with similar properties, common services, common relationships, and common semantics.

3.1.6 composite resource: A specific type of real resource. It describes a collection of individual resources, or instruments, that provide some behavior when used collectively. Information known about the individual resources that make up the composite resource includes connectivity and the mapping of their individual resource ports onto the composite resource ports. *See also:* **real resource**.

3.1.7 connection: The virtual connection of a virtual resource that connects a test subject port to an external port of the test equipment configuration. A connection can be stated in ATLAS, for example, in the following manner: CNX HI P1-1. In this example, P1-1 is the test subject pin, and HI is the ATLAS reference to a port on the test equipment configuration.

3.1.8 connection set: A multiple list of connections used by a resource to connect multiple test subject ports to multiple output ports in the test equipment configuration. A connection set can be stated in ATLAS, for example, in the following manner: CNX HI P1-1 LO P1-2. In this example, the set is comprised of two connections, HI P1-1 and LO P1-2. Additionally, in rare cases, a list of connection sets can occur for resources that change connections during the implementation of their behavior, such as when checking for multiple ports. A set of connection sets can be shown in ATLAS by more than one set of HI, LO, or other test equipment configuration port designators linked to test subject ports.

3.1.9 deallocate: The process of releasing a resource from a particular task and making it available to other users. This is the reverse process of allocate. *Contrast:* **allocate**.

3.1.10 individual resource: A specific type of real resource that can be used by itself or be used jointly with other individual resources to make up a composite resource. The purpose of an individual resource is to generate or measure a class of signal. Information known about the individual resource includes its control interface and its current state. *See also:* **composite resource** and **real resource**.

3.1.11 instrument: An individual item that is synonymous with an individual resource. *Syn:* **individual resource**.

3.1.12 language-independent specification: The format for describing the services that are not tied to a specific computer language, syntax, or implied semantics.

3.1.13 object: A member (instance) of a class that encapsulates (i.e., hides the details of) state and operations.

3.1.14 path: Describes a contiguous routing through a test equipment configuration. Each path begins with a port and ends with a port. A path may include multiple switch connections and/or paths.

3.1.15 path set: Involves two or more paths that are needed to support a specific real resource. Many real resources have multiple ports that are active simultaneously and require multiple paths for the required connections. In addition, sets of path sets, or multiple path sets, may be required by a specific real resource.

3.1.16 physical mapping: The process of mapping a virtual resource to one or more real resources.

3.1.17 port: Describes the interface points on the test equipment configuration, the interface points on an individual resource, the interface points on a composite resource, the interface points on a virtual resource, and the interface points on a test subject.

3.1.18 real resource: An abstract supertype. The subtypes of real resources consist of individual resources or composite resources, both of which describe physical instruments. Unlike virtual resources, instances of the subtypes of real resources physically appear in a test equipment configuration.

3.1.19 resource: An abstract supertype in a test equipment configuration. A resource consists of virtual resources and real resources (even the test equipment configuration itself can be a resource). A resource is described by a set of ports and an optional set of behaviors.

3.1.20 service: A user-requested operation or runtime call whose behavior and interface is standardized.

3.1.21 signal: The observable behavior produced by or associated with a test procedure, resource, or path.

3.1.22 switch: A specific type of instrument or individual resource that is used for connecting or breaking an electric circuit.

3.1.23 switch connections: Routes through a switch that require a specific type of instrument or individual resource. A switch connection can be a part of a defined path.

3.1.24 test: A step in a test procedure that requires specific capabilities and produces an observation or response.

3.1.25 test equipment configuration: An integrated set of real resources. This composition includes all paths allowed by the interconnections of the real resources and all virtual resources that have been created using real resources. *See also:* **real resource**.

3.1.26 test foundation framework: A comprehensive set of object classes that supports the development of test programs and diagnostic data elements. The test foundation framework is defined in IEEE Std 1226-1993.

3.1.27 test method: Defines the algorithm and functionality of a test object.

3.1.28 test object: An object that represents a test of a specific behavior.

3.1.29 test procedure: Defines the implementation of a test method.

3.1.30 test program: Contains the coded sequences of a test procedure that, when executed, will provide a set of instructions sufficient to automatically ascertain the operational condition of a test subject and detect and isolate faults within a test subject.

3.1.31 test resource information model (TRIM): Identifies entities and the relationships between the entities of a test equipment configuration. See IEEE Std 1226-1993.

3.1.32 test subject: The physical object or type of physical object that is the focus of attention or target for the development of tests and diagnostics.

3.1.33 unit under test (UUT): Any system, subsystem, assembly, or subassembly undergoing test.

3.1.34 virtual resource: A specific type of resource that defines a representative interface and test behavior between a test equipment configuration and a test subject. More specifically, a virtual resource defines a connection or connection set between the external ports of a test equipment configuration and a test subject, as well as a behavior required to test a specific function of the test subject. A virtual resource makes no assumptions about any real resources or conditions necessary to realize the behavior.

3.1.35 virtual resource group: A virtual resource group defines a collection of virtual resources that are required simultaneously in order to test a specific function of a test subject.

3.2 Acronyms

ABBET	A Broad-Based Environment for Test
ATE	automatic test equipment
ATLAS	Abbreviated Test Language for All Systems
ATS	automatic test system
CORBA	Common Object Request Broker Architecture
IDL	Interface Definition Language
OMG	Object Management Group
OO	object oriented
OOA	object-oriented analysis
RM	resource management
TAF	test application framework
TEC	test equipment configuration
TFF	test foundation framework
TPS	test program set
TRIM	test resource information model
UUT	unit under test

4. Description

This clause provides a description of the relationships of this standard to IEEE P1226/D15.3 and its associated standards.

4.1 Objectives

Resource management (RM) provides access to descriptive information about test resource behaviors, test resource interconnections, and the connection of the test subject to the ATE. RM also provides all services needed to perform virtual to real resource mapping, virtual resource management, test equipment configuration management, and real resource management.

4.2 Overview

This standard specifies the interfaces that will enable an ATE independent test program to be implemented on an ATE system. ATE independent implementations may utilize the services specified in this standard to provide a transportable means to bridge the gap between the test procedure and the ATE. Examples of the use of the services defined in this standard can be found in Annex A.

A test program contains requirements for the virtual resources that are needed to test a specific test subject. The services described in this standard map the virtual resources to the real resources on a particular ATE. The test resource information model (TRIM) for this standard is referenced in IEEE P1226/D15.3.

The process of defining the services in this standard involved an object-oriented analysis (OOA) in conjunction with the TRIM. The OOA defines a set of class and object diagrams that identifies classes, objects, and their relationships. These diagrams and an explanation of each are found in Annex B of this standard. From the OOA, the services contained in this standard were defined and appear in Clause 6.

A language-independent implementation of the services is shown in Annex C, written in the Interface Description Language (IDL) defined in OMG CORBA/IIOP Revision 2.1. These service representations are computer processible and identify classes, objects, methods, and method parameters.

Clause 6 is divided into three sections as follows:

- a) Virtual resource management services
- b) Test equipment configuration management services
- c) Real resource management services

The virtual resource management services describe services that are available at the virtual test level. These services are divided into services for individual virtual resources and services for groups of virtual resources. It is entirely possible to write a test program using only the services specified in the virtual resource management services subclause (6.2) of this standard. For example, a test programmer may know that an ac signal with a -10 dBm power and a 100 MHz frequency is needed at test subject pin J1-1. This information is conveyed to the virtual services in this standard via the test procedure, which may or may not be written in ATLAS. The examples in Annex A of this standard utilize ATLAS, but this is arbitrary. The test-specific information may be used to create and manipulate a virtual resource via the services defined in the virtual resource management subclause (6.2) of this standard. The virtual resource can be mapped to real resources and, eventually, used to command the real resources to perform the functions needed by the test programmer. An example of this can be found in more detail in Clause A.1 of Annex A. It is important to note that the virtual resource creation and initialization can occur dynamically, during test execution, or statically, at some time prior to test execution.

The virtual resource group services are a subset of the services that are provided in the virtual resource management services subclause (6.2) of this standard. The discriminating factor for the services associated with the virtual resource group is simultaneity. The virtual resource group services let the test programmer group virtual resources together and indicate that all of them must be available at the same time in order for testing to occur. An example that uses the virtual resource group services can be found in Clause A.3 of Annex A.

The test equipment configuration management services describe services to manipulate and search information related to the configuration of a particular set of test equipment. These services can be utilized by the implementation of the virtual resource management services, or they can be accessed directly by tester configuration tools or by the test program itself. An example of a tool that may want to access the test equipment configuration services is a test subject/tester analysis tool. This tool could analyze a particular tester configuration, and could identify whether it could be used to test a specific test subject. The analysis would be based on the behaviors of the test subject that need to be tested. Clause A.9 in Annex A describes just such a tool.

In addition, a test program could directly use the test equipment configuration management services to find a set of real resources and paths that satisfy a particular test subject behavior to be tested. Once a list of real resources and paths are found, a decision can be made on which to use, and these real assets manipulated through the real resource management services of this standard. Clause A.5 in Annex A further describes this scenario.

The real resource management services describe services to find and allocate physical resources and paths. These services are usable from implementations of the virtual resource services or directly from the test program. Clause A.6 in Annex A describes, in detail, how a test program can utilize the real resource management services. In particular, if the test program knows the tester's real resources and paths, the services in the real resource management subclause (6.4) of this standard can be used to allocate, deallocate, and manipulate information about them.

4.3 Relationship to ABBET architecture

The primary purpose of A Broad-Based Environment for Test (ABBET) is to provide for a broad-based environment for test that encompasses the total product life cycle. To do so, an integrated set of interfaces to

test related information and services are defined that support the information and implementation needs of this test environment for automated testing applications.

The ABBET architecture is defined by IEEE P1226/D15.3. It defines the existence of interfaces between software components. Each of these interfaces is, in turn, specified in separate ABBET component standards, such as the software interface defined by this standard. The relationship of this standard to IEEE P1226/D15.3 is defined in terms of the ABBET architecture that is defined within IEEE P1226/D15.3.

4.3.1 Architecture levels

The ABBET architecture consists of a set of interfaces that are created from a single class hierarchy that is created in a foundation infrastructure. The foundation provides the basic frame of reference, both to basic classes and to primary interfaces. These can be specialized at different levels to create general test environments (frameworks) or specific test applications. Whereas a library is normally a collection of subroutines, a framework is a collection of classes. Both are usually designed to provide required functionality in a specific and limited application domain.

Figure 1 diagrams the ABBET architecture. The discussion that follows describes each of the following major elements:

- a) Foundation framework
- b) Extension framework
- c) Application

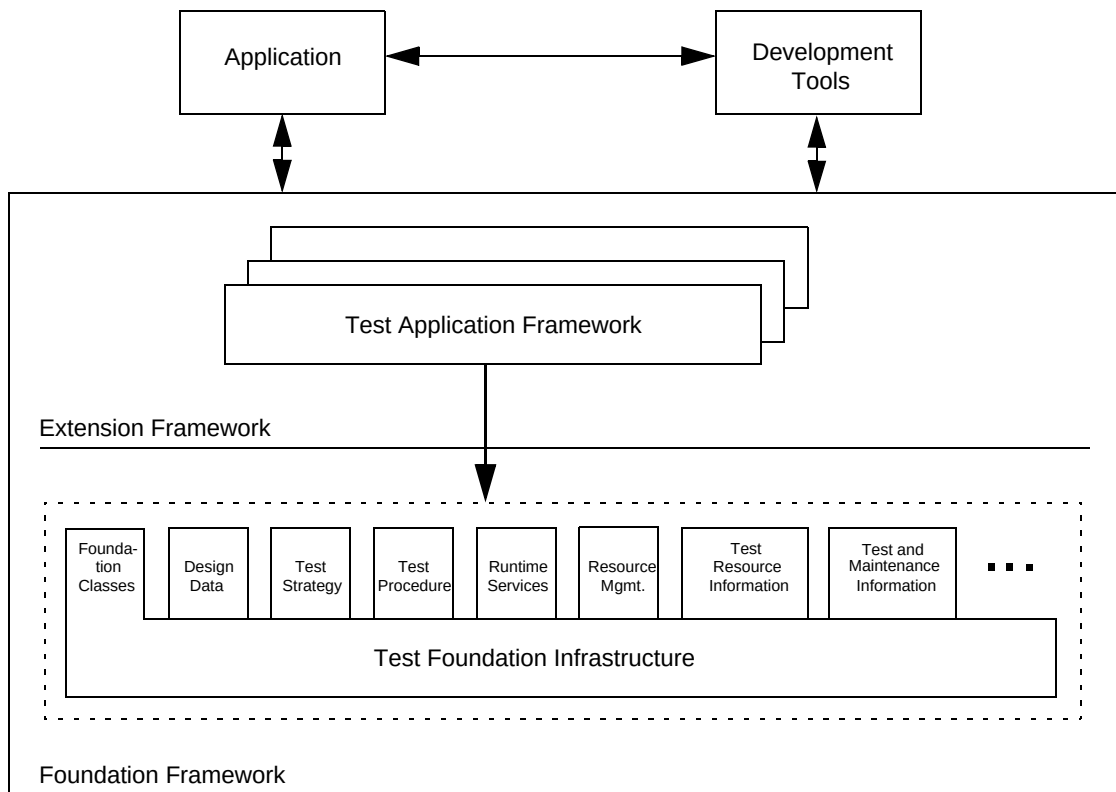


Figure 1—ABBET architecture

4.3.2 Foundation framework

Any single organization can produce applications for its own needs and according to its own policies, technical expertise, and available test facilities. These applications are useful within that organization, but they usually cannot be shared with others until a basis for that sharing has been achieved. ABBET provides the standards upon which a foundation for object sharing can be built.

The foundation framework is organized as a set of interfaces, with each interface related to one or more ABBET component standards or to external standards maintained by either the IEEE or another recognized body. Basic interfaces appropriate to various aspects of a product life cycle are defined for each product family. The foundation framework contains the basic classes applicable to the components regardless of the application. The other interfaces in the foundation framework use and inherit from the foundation classes.

4.3.3 Extension framework

Foundation classes are too generic and abstract to be of direct use in test application development. However, they provide the starting point for additional specialization that leads to useful test applications. As new classes are developed, new interfaces are defined that select and group objects in support of a product family, mission, or application type. The resulting group is called a test application framework (TAF). A TAF is a reusable collection of classes that is designed to meet the needs of a specific application area. Each TAF provides services to support a specific category of test subjects, technologies, resources, or requirements.

Each class within a TAF fits into the class hierarchy created in the foundation infrastructure. Once the component classes of an application framework have been declared within a test procedure or program, they can be used as if they were a fundamental part of the system.

The extension framework level contains one or more TAFs. TAFs can be moved into the foundation framework if the application is of wide interest and greater accessibility is desired. For example, RM was originally considered a test application framework. It was moved to the foundation framework because of its widespread uses and the need for greater accessibility by other interfaces in this framework.

4.3.4 Application

The ABBET architecture supports the application level by providing direct access both from test development tools and TAFs.

An application may use one or more foundation frameworks to provide access to the classes needed to implement the application. The ABBET object model avoids any naming conflicts between various foundation framework classes and subclasses through the use of OMG IDL module definitions (see OMG CORBA/IOP Revision 2.1). Each module within the foundation framework has a unique module name. Since an OMG IDL module provides for fully qualified names to be used, each class and method of a class within the module has a unique name. Therefore, only the module names are required to be managed. This mechanism should be applied to any extension framework also.

4.3.5 Relation to ABBET standards

The TRIM provides the information structure needed for a repository of test system information. It can be realized as a database of test instrument capabilities and test system configuration data. Generally, the data in such a model is changed only when an instrument is updated or swapped from an existing configuration. The TRIM permits the information and its access methods to be shared among test systems, vendors, and users, thereby minimizing redundant work.

The RM component provides the primary interfaces needed to fully manage resources found within a test system. It is also a part of the foundation framework. An implementation of this component may use other

interfaces or classes defined in the foundation framework such as the TRIM or the interface to instrument drivers or instrument communication buses.

An implementation of this standard is intended to work with or without an implementation of the TRIM. This standard defines services for the creation and utilization of physical objects identified in the TRIM, such as paths, composite resources, and individual resources. Each of the objects in this standard has an initialization service with a name parameter. The purpose of the initialization services in this standard is to link the IEEE Std 1226.3-1998 level objects with their lower-level equivalent objects. If a TRIM implementation is not included, then these same objects will be defined in an implementation-dependent manner. In either case, the IEEE Std 1226.3-1998 level objects become linked to the lower-level objects in order to perform the services in this standard.

4.4 Relationship to reference documents

OMG CORBA/IIOP Revision 2.1 contains the IDL definition that is used to develop the IDL binding for the services in this standard. IEEE Std 1226-1993 is used as a reference for some of the ATLAS-based scenarios and comments used throughout this standard.

5. Conformance

This clause defines the requirements for conformance with this standard. It defines the following:

- a) The minimum functionality required
- b) Allowed subsets and extensions
- c) The approach for implementing the subsets and extensions

An implementation is in conformance with this standard if it provides the minimum required RM functionality described in 5.1. It may also be tailored to one of the allowed subsets and/or extended as specified in 5.2 and 5.3.

Any claim of conformance shall specifically list those functions that are not supported and those extensions that have been made.

5.1 Minimum resource management interface functionality

An implementation shall be in conformance with this standard if it supports one or more of the allowable subsets, as defined in 5.2.

5.2 Subsets

Table 1 identifies the allowed subsets and groups of allowable subsets, along with the required services for each. All services within a subset must be implemented in order to be compliant.

A subset is defined as an implementation in which not every service is included for commercial or technical reasons.

5.3 Extensions

A test system implementation shall be in conformance with this standard if it supports extensions as defined in the following paragraph.

Table 1—Resource management allowed subsets

Subset	Service groups	Paragraph number
1	Virtual resource	6.2.1– 6.2.4, 6.3.1.11*
2	Virtual resource Virtual resource group	6.2, 6.3.1.10*, 6.3.1.11*
3	Test equipment configuration Real resource management	6.2.2 - 6.2.4, 6.3.1 (except 6.3.1.10 and 6.3.1.11), 6.4
4	Real resource management	6.2.4, 6.4
5	Virtual resource Test equipment configuration Real resource management	6.2.1–6.2.4, 6.3.1 (except 6.3.1.10), 6.4
6	Virtual resource Real resource management	6.2.1– 6.2.4, 6.4
7	Virtual resource Virtual resource group Real resource management	6.2, 6.3.1.10*, 6.3.1.11*, 6.4

*This service is not required for an object-oriented (OO) implementation. If a non-object-oriented implementation is performed, this service is required as well as the create, destroy, and initialize services of test equipment configuration (TEC).

Whenever an existing RM behavior, as defined by the most current, approved version of this standard, can satisfy an application requirement, then existing RM services shall be used. If services are created that provide a legitimate new RM behavior required for the implementation of resource management applications, then any such services shall be identified as extensions.

5.4 Approach for implementation of subsets

All subsets of this standard shall be in accordance with 5.2.

No adaptation or distortion of the service interface(s) or description(s) of this standard is allowed.

A subset can only be created by selecting the required services, as defined in Table 1 of the most current, approved version of this standard. The subset shall be complete (i.e., each of the services indicated in the subset must be implemented).

5.5 Approach for implementation of extensions

Should behavior be required that is not supported by the services defined in this standard, an ABBET Submittal Form (available from the ABBET subcommittee) shall be completed and submitted to the SCC20 ABBET subcommittee for consideration as new material to be incorporated into this standard. All extensions used in an implementation shall be clearly and explicitly identified as changes to this standard.

Extensions are allowed to utilize the standard services; however, the standard services are not allowed to be dependent on the extensions. Users shall be made aware, through clear and explicit documentation, that applications utilizing the extensions will not be portable to other compliant implementations of RM.

6. Specification

6.1 Conventions

This clause defines the conventions used to describe the services for this standard. Services are grouped by class. For each class, there is a state diagram that describes the methods for state transition. For each service of a class, there is a definition that includes parameters, a list of the types of parameters, a description of the service action, and a description of the completion status returned. Classes are grouped into three segments:

- a) Virtual resource management
- b) Test equipment configuration management
- c) Real resource management

6.1.1 Specification layout

The following shows the structure for the specification of the services.

Format:

```

6.x First segment
6.x.1 First class of this segment
6.x.1.1 State diagram for this class
6.x.1.2 First service of this class
6.x.1.2.1 Interface for this service
6.x.1.2.2 Description for this service
6.x.1.2.3 Completion status for this service
6.x.1.3 Next service of this class
:
:
6.x.2 Next class of this segment
:
:
6.y Next segment

```

Example:

```

6.2 Virtual resource management services
6.2.1 [Virtual Resource]
6.2.1.1 State model
6.2.1.2 [Virtual Resource] create
6.2.1.2.1 Service interface
6.2.1.2.2 Description
6.2.1.2.3 Completion status
6.2.1.3 [Virtual Resource] initialize
:
:
6.2.2 [Connection]

6.3 Test equipment configuration manage-
ment services

```

6.1.2 State diagram

A state diagram is provided for each class. These diagrams describe the states for each object in a class along with the state transitions that occur as a result of service performance. This serves as a top-level validation that sufficient services have been defined within each class. Each object has an initial state of nonexistent, which means that no object of that class has been created. The create service causes a state transition to the exist state. The exist state occurs when an identifier is established for an object of a given class. The initialize service causes a state transition to some class-dependent state that has a named identifier, and possibly other attributes. Separate states for existence and initialized are required to conform to the parameterless constructors and destructors defined in IDL.

6.1.3 Service definition

Each service is defined by providing the interface definition with parameters and their types, a natural language description of the service, and a completion status. The formal IDL service definitions are provided in Annex C.

6.1.3.1 Interface and description

This paragraph identifies the format used to describe the language-independent specifications throughout Clause 6. The initial entry of a service name is preceded by the class name enclosed in square brackets. Within the service interface, the object identifier upon which the service is acting is enclosed in triangular brackets preceding the service name. If the service operates on a class instead of an object (such as create), no preceding identifier is applicable. The service interface within each service describes the input and output parameters. Each parameter is listed as an input or an output followed by the specified identifier and its type.

Format:

Service:	service name	
Input:	input parameter name;	input parameter type
Output:	output parameter name;	output parameter type

Example:

Service:	<virtual_resource_id>.perform_service	
Input:	service_id	service_to_perform
Input:	parameters	input_parameters
Output:	measured value ⁵	data

6.1.3.2 Type definitions

The parameters are one of the types listed below. Those types that are defined as “a unique object identifier” are the result of creating an object of their associated class. This identifier is used to call all the services of the class, applied to a unique instantiation. For the enumeration, the actual values are identified in the services where they are utilized.

behavior_id	A unique behavior object identifier
connection_id	A unique object identifier
connection_set_id	A unique object identifier
data	A unique object identifier
criteria	Enumerated type to indicate search criteria
input_parameters	A unique object identifier
object_id	A unique object identifier
path_id	A unique object identifier
path_set_id	A unique object identifier
port_id	A unique object identifier
real_resource_id	A unique object identifier
real_resource_set_id	A unique object identifier
service_to_perform	Enumerated type to indicate an action to perform
set_of_behavior_id	A unique object identifier
set_of_connection_ids	A unique object identifier
set_of_path_id	A unique object identifier
set_of_path_sets_id	A unique object identifier
set_of_real_resources_id	A unique object identifier

⁵Optional parameter.

status	Enumerated type to indicate availability
string	A sequence of character data, with a minimum length of 0 and no defined maximum length
test_equipment_configuration_id	A unique object identifier
virtual_resource_group_id	A unique object identifier
virtual_resource_id	A unique object identifier

6.1.3.3 Completion status definitions

The completion status subclause of each service identifies all possible error conditions that may be raised by the service. It is assumed a successful completion will exhibit no error or will simply return an expected output value. During the error condition, the class name and service name will be included with the raised error to identify the error's occurrence. When an error occurs, the system shall be left in the calling state. No partial completion of a service is permitted.

For different language implementations, the completion status is realized according to the IDL to language mapping. It may be implemented as an exception, as in Ada, or as a returned or queried error code, as in C. No implementation restriction is made in this clause of the standard.

The completion statuses are listed in alphabetical order, with a brief description of their meaning:

Invalid behavior:	indicates that the behavior_id provided is invalid or not created.
Invalid connection:	indicates that an invalid connection_id has been specified.
Invalid connection set:	indicates that the connection set is invalid or not created.
Invalid name:	indicates an illegal name has been used.
Invalid parameters:	indicates that the object has received invalid parameter values.
Invalid path:	indicates that the path provided is invalid or not created.
Invalid path set:	indicates that the path_set identifier provided is invalid or not created.
Invalid port:	indicates that an invalid port_id has been specified.
Invalid real resource:	indicates that the real resource identifier does not exist.
Invalid real resource set:	indicates that the real resource set provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid service id:	indicates that the service_id parameter is invalid.
Invalid set of path set:	indicates that a set_of_path_set_id provided is invalid or not created.
Invalid set of path sets:	indicates that the set of path sets is invalid.
Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
No behavior:	indicates that no behavior is associated with this path.
Unsuccessful operation:	indicates that the service has failed.

6.2 Virtual resource management services

This subclause defines the functionality of services for the management of virtual test resources.

6.2.1 [Virtual Resource]

6.2.1.1 State diagram

Figure 2 represents the state diagram for the virtual resource class.

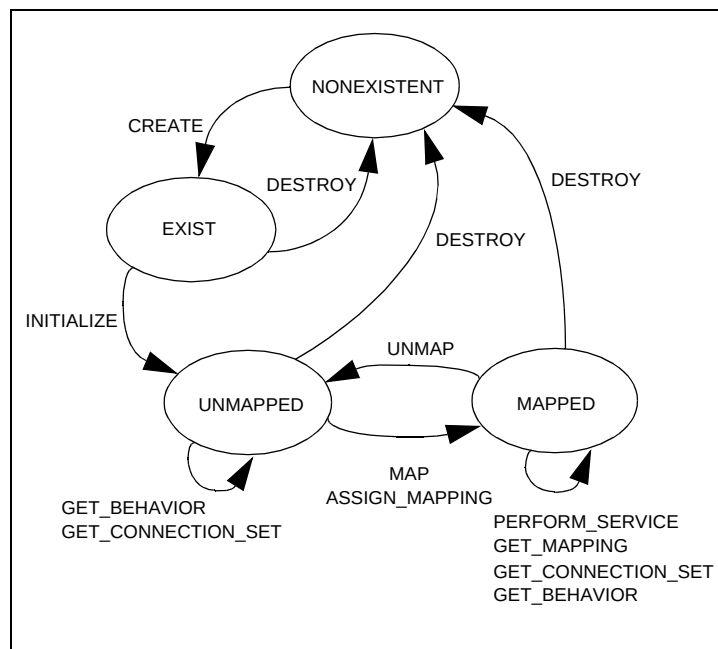


Figure 2—Virtual resource state diagram

6.2.1.2 [Virtual Resource] create

6.2.1.2.1 Service interface

Service: create

Output: virtual resource; virtual_resource_id

6.2.1.2.2 Description

This service shall create a virtual resource object and shall return a unique virtual resource object identifier. No object identifiers shall be duplicated and no duplicate virtual resource names shall be allowed. If the virtual resource cannot be created, a value indicating this shall be returned.

6.2.1.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.1.3 [Virtual Resource]

6.2.1.3.1 Service interface

Service:	<virtual_resource_id>.initialize	
Input:	name;	string
Input:	behavior;	behavior_id
Input:	connection set;	connection_set_id

6.2.1.3.2 Description

This service shall initialize a virtual resource object. A search for candidate resources and paths may be done optionally as a function of this service. No actual mapping or enabling of resources and paths, however, shall be done as a result of this service. The behavior input parameter is a behavior object identifier that represents a particular behavior, such as an ac signal at -5 dBm and 100 MHz. The connection set input parameter is the connection set that is associated with this virtual resource. A connection set is a pair of virtual port identifiers that create a virtual connection.

6.2.1.3.3 Completion status

Invalid name:	indicates that an illegal name has been used.
Invalid behavior:	indicates that the behavior_id provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid connection set:	indicates that the connection set is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.4 [Virtual Resource] destroy

6.2.1.4.1 Service interface

Service:	<virtual_resource_id>.destroy
----------	-------------------------------

6.2.1.4.2 Description

This service shall destroy a virtual resource by deleting the virtual resource object identified by its unique virtual resource identifier. No disabling of physical resources associated with the virtual resource shall occur.

6.2.1.4.3 Completion status

Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.5 [Virtual Resource] map

6.2.1.5.1 Service interface

Service:	<virtual_resource_id>.map
----------	---------------------------

6.2.1.5.2 Description

This service shall map a virtual resource by mapping and enabling its behaviors to specific physical paths. This service shall perform this service in one of two ways. First, if candidate real resources and paths have been found previously by the initialize service, this service shall choose among the candidates and perform the mapping. Second, if the candidate resources and paths have not been found, this service shall perform this function first, choose among the candidates, and then perform the mapping.

6.2.1.5.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.6 [Virtual Resource] unmap

6.2.1.6.1 Service interface

Service: <virtual_resource_id>.unmap

6.2.1.6.2 Description

This service shall unmap a virtual resource by releasing the resources and paths. The virtual instrument, however, is still available to be relinked with other resources and paths.

6.2.1.6.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.7 [Virtual Resource] assign mapping

6.2.1.7.1 Service interface

Service:	<virtual_resource_id>.assign_mapping	
Input:	resource mapping;	real_resource_id
Input:	connection mapping;	set_of_path_sets_id

6.2.1.7.2 Description

This service shall map specific real resources and real paths to a virtual resource. This service differs from the <virtual_resource_id> map service in that the virtual resource does not get to choose which real resources and path to use. The choosing function is typically performed by the virtual resource group, which has access to information for all the virtual resources within that group. The specific mapping information is then passed to this routine via input parameters.

The resource mapping is a real resource object identifier that represents a physical instrument or a physical composite instrument that is to be mapped to the specified virtual resource. The input parameter,

connection_mapping, is an object identifier that indicates a set of path sets that represent physical path sets that are to be mapped to the specified virtual resource.

6.2.1.7.3 Completion status

Invalid set of path sets:	indicates that the set of path sets is invalid.
Invalid real resource:	indicates that the real resource identifier does not exist.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.8 [Virtual Resource] get mapping

6.2.1.8.1 Service interface

Service:	<virtual_resource_id>.get_mapping	
Output:	resource mapping;	real_resource_id
Output:	connection mapping;	set_of_path_sets_id

6.2.1.8.2 Description

This service shall return the physical mapping of the virtual resource to a real resource and paths. The resource mapping output parameter is a real resource object identifier that has been mapped to the specified virtual resource. The connection mapping output parameter is a set of path sets that have been mapped to a specified virtual resource. A path set is a set of path object identifiers. If either output cannot be generated, a value that indicates this will be returned.

6.2.1.8.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.9 [Virtual Resource] perform service

6.2.1.9.1 Service interface

Service:	<virtual_resource_id>.perform_service	
Input:	service id	service_to_perform
Input:	parameters	input_parameters
Output:	measured value ⁶	data

⁶Optional parameter.

6.2.1.9.2 Description

This service shall cause the behavior associated with a particular virtual resource to occur. The service id input parameter identifies which physical activity is to occur. For example, the virtual resource may be set up as a source, but the service id will indicate if a setup or apply is to be done. The parameter input values are the specific values reserved for the virtual resource, and the optional measured value parameter is a value returned from the real resource. This service can only be called if the virtual resource has previously been mapped, which guarantees the associated behavior can be performed with specific real resources.

6.2.1.9.3 Completion status

Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Invalid service id:	indicates that the virtual resource cannot provide this function.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid parameters:	indicates that the virtual resource has received invalid parameter values.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.10 [Virtual Resource] get behavior

6.2.1.10.1 Service interface

Service:	<virtual_resource_id>.get_behavior	
Output:	behavior;	behavior_id

6.2.1.10.2 Description

This service returns the behavior id with which the virtual resource object was initialized.

6.2.1.10.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.11 [Virtual Resource] get connection set

6.2.1.11.1 Service interface

Service:	<virtual_resource_id>.get_connection_set	
Output:	connection set;	connection_set_id

6.2.1.11.2 Description

This service returns the connection set id with which the virtual resource object was initialized.

6.2.1.11.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.2.2 [Connection]

6.2.2.1 State diagram

Figure 3 represents the state diagram for the connection class.

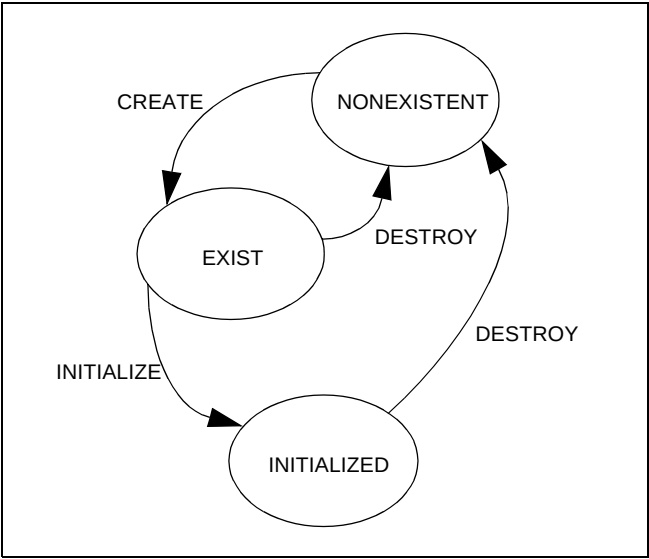


Figure 3—Connection state diagram

6.2.2.2 [Connection] create

6.2.2.2.1 Service interface

Service:	create	
Output:	connection;	connection_id

6.2.2.2.2 Description

This service shall create a connection. If the connection cannot be created, a value indicating this shall be returned. A single connection identifier is output, which is a reference identifier to access the connection in the future.

6.2.2.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.2.3 [Connection] initialize

6.2.2.3.1 Service interface

Service:	<connection_id>.initialize	
Input:	name;	string
Input:	from port;	port_id
Input:	to port;	port_id

6.2.2.3.2 Description

This service shall initialize a previously created connection with the two ports specified.

6.2.2.3.3 Completion status

Invalid name:	indicates that an illegal name has been specified.
Invalid connection:	indicates that an invalid connection_id has been specified.
Invalid port:	indicates that an invalid port_id has been specified.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.2.2.4 [Connection] destroy

6.2.2.4.1 Service interface

Service: <connection_id>.destroy

6.2.2.4.2 Description

This service shall destroy the connection by deleting its unique identifier.

6.2.2.4.3 Completion status

Invalid connection:	indicates that the connection provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.3 [Connection Set]

6.2.3.1 State diagram

Figure 4 represents the state diagram for the connection set class.

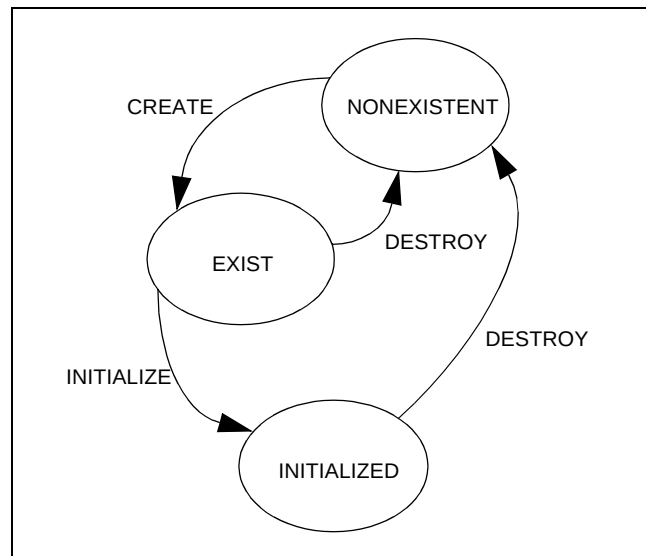


Figure 4—Connection set state diagram

6.2.3.2 [Connection Set] create

6.2.3.2.1 Service interface

Service: create

Output: connection set; connection_set_id

6.2.3.2.2 Description

This service creates an object of the class called connection set. If the connection set cannot be created, a value indicating this shall be returned. The connection set inherits from the connection set that is defined in the test foundation framework (TFF). Services that are inherited include add and remove; however, a locking mechanism is included to restrict users from inadvertently changing a connection set during utilization. Returned from this service is a connection set identifier that is used to access the set.

6.2.3.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.3.3 [Connection Set] destroy

6.2.3.3.1 Service interface

Service: <connection_set_id>.destroy

6.2.3.3.2 Description

This service shall destroy the connection set by deleting its unique identifier.

6.2.3.3.3 Completion status

Invalid connection set: indicates that the connection provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

6.2.4 [Port]

6.2.4.1 State diagram

Figure 5 represents the state diagram for the port class.

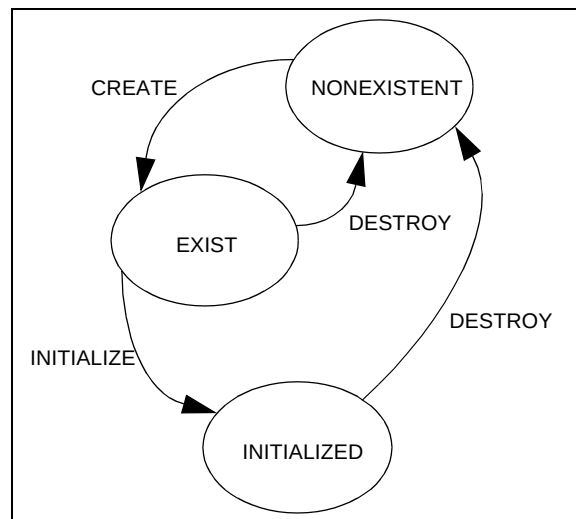


Figure 5—Port state diagram

6.2.4.2 [Port] create

6.2.4.2.1 Service interface

Service:	create	
Output:	port;	port_id

6.2.4.2.2 Description

This service shall create a port object and shall return a unique port object identifier. The output parameter, port, is an identifier that represents the actual port object and is unique to a port object. If the port cannot be created, a value that indicates this shall be returned. The object identifier is the only way to access a port object.

6.2.4.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.4.3 [Port] initialize

6.2.4.3.1 Service interface

Service:	<port_id>.initialize	
Input:	name;	string

6.2.4.3.2 Description

This service shall initialize a previously created port object. The input parameter, name, is a string that represents the physical name of the port.

6.2.4.3.3 Completion status

Invalid name:	indicates that an illegal name has been used.
Invalid port:	indicates that the port provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.2.4.4 [Port] destroy

6.2.4.4.1 Service interface

Service:	<port_id>.destroy
----------	-------------------

6.2.4.4.2 Description

This service shall destroy a port object by deleting its unique identifier.

6.2.4.4.3 Completion status

Invalid port:	indicates that the port provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.5 [Virtual Resource Group]

6.2.5.1 State diagram

Figure 6 represents the state diagram for the virtual resource group class.

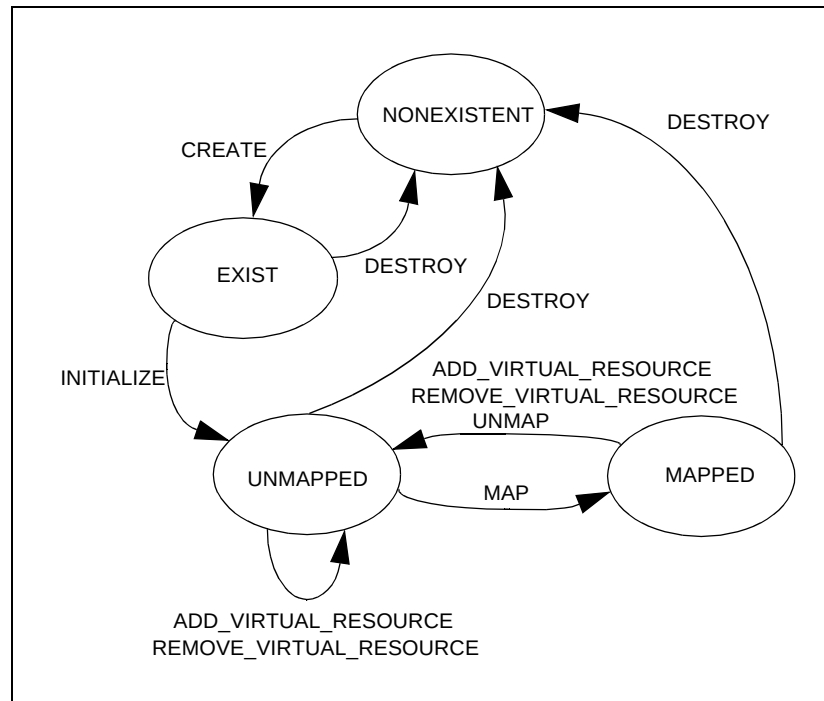


Figure 6—Virtual resource group state diagram

6.2.5.2 [Virtual Resource Group] create

6.2.5.2.1 Service interface

Service: create

Output: virtual resource group; virtual_resource_group_id

6.2.5.2.2 Description

This service shall create a virtual resource group that is a container structure that allows a grouping of virtual resources to occur. This service shall return a unique virtual resource group identifier that represents the created group. The virtual resource group identifier is the only way to access the group and its resources. If the virtual resource group cannot be created, a value indicating this shall be returned.

6.2.5.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.5.3 [Virtual Resource Group] initialize

6.2.5.3.1 Service interface

Service: <virtual_resource_group_id>.initialize

Input name; string

6.2.5.3.2 Description

This service shall initialize a previously created virtual resource group. The input parameter is a string that represents the virtual resource group name. Should the creation of this virtual resource group happen outside the domain of a test application, the test application shall use the name during initialization to retrieve the identifier later via a call to get_virtual_group_id.

6.2.5.3.3 Completion status

Invalid virtual resource group: indicates that the virtual resource group provided is invalid or not created.
Invalid name: indicates that an illegal name was used as a parameter.
Invalid state: indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation: indicates that the service has failed.

6.2.5.4 [Virtual Resource Group] destroy

6.2.5.4.1 Service interface

Service: <virtual_resource_group_id>.destroy

6.2.5.4.2 Description

This service shall destroy a virtual resource group object by deleting its unique identifier. No disabling of physical resources associated with any of the virtual resources within the group shall occur.

6.2.5.4.3 Completion status

Invalid virtual resource group: indicates that the virtual resource group provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

6.2.5.5 [Virtual Resource Group] map

6.2.5.5.1 Service interface

Service: <virtual_resource_group_id>.map

6.2.5.5.2 Description

This service shall map physical entities to all virtual resources that are associated with the virtual resource group that is identified by the virtual_resource_group_id. This could consist of mapping all virtual resources of the group or only those that were added since the last mapping. It is left to each implementation of the virtual resources to cause the behavior to occur.

6.2.5.5.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.5.6 [Virtual Resource Group] unmap**6.2.5.6.1 Service interface**

Service: <virtual_resource_group_id>.unmap

6.2.5.6.2 Description

This service shall unmap a group of virtual resources that belong to a virtual resource group. This service shall unmap all virtual resources associated with the virtual resource group that is specified by the virtual resource group identifier.

6.2.5.6.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.5.7 [Virtual Resource Group] add virtual resource**6.2.5.7.1 Service interface**

Service: <virtual_resource_group_id>.add_virtual_resource

Input: virtual resource; virtual_resource_id

6.2.5.7.2 Description

This service shall add a virtual resource to a virtual resource group. The virtual resource to be added is identified by its unique virtual resource object identifier.

6.2.5.7.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.