NIST Special Publication 500-181

Computer Systems Technology

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

PHIGS Validation Tests (Version 1.0): Design Issues

John Cugini
Mary T. Gunn
Lynne S. Rosenthal

NIST Special Publication 500-181

PHIGS Validation Tests (Version 1.0): Design Issues

John Cugini
Mary T. Gunn
Lynne S. Rosenthal

Information Systems Engineering Division
National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

July 1990

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director

Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) (formerly the National Bureau of Standards) has a unique responsibility for computer systems technology within the Federal government. NIST's National Computer Systems Laboratory (NCSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. NCSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. NCSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports NCSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

National Institute of Standards and Technology Special Publication 500-181
Natl. Inst. Stand. Technol. Spec. Publ. 500-181, 21 pages (July 1990)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1990

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

ABSTRACT

Conformance testing for the Programmer's Hierarchical Interactive Graphics System (PHIGS) standard presents certain novel difficulties, especially the indirect effect of many functions, and the inaccessibility to the program of visual effects. The model of logical inference offers a way to organize a system of the complexity needed to overcome these problems. This complexity makes the use of certain database concepts quite valuable in allowing users to comprehend the system. Special emphasis is placed on allowing the user to associate each test case with some specific requirement in the standard. Test output consists of a set of formatted messages that enable the user to assess test results rapidly and accurately.

KEYWORDS: conformance testing; graphics standards; PHIGS; testing of software; validation of software

CONTENTS

1. INTRODUCTION

PHIGS stands for Programmer's Hierarchical Interactive Graphics System. The PHIGS Validation Test (PVT) suite is a product of the National Computer Systems Laboratory (NCSL) of the National Institute of Standards and Technology (NIST). The function of this suite is to test whether implementations of PHIGS conform to the PHIGS standard [PHIGS88]. This document describes the major design issues that were encountered during development of version 1.0 of the PVT, the rationale for their resolution, and some tentative plans for the future evolution of the test suite. The PVT User's Guide [CUGI90] contains information on the installation and operation of the PVT suite, and on interpretation of its output.

1.1 The PHIGS Standard

The PHIGS standard defines a set of functions to be used by a programmer to manipulate and display 3-D graphical objects. For a full description of the features of PHIGS, see [PHIGS88]. The standard has been approved by the American National Standards Institute (ANSI) as ANSI X3.144-1988, by the International Organization for Standards (ISO) as ISO 9592:1988, and by the Federal Government as Federal Information Processing Standard (FIPS) 153.

1.2 Definitions

Conformance: the state of having satisfied the requirements of some specific standard(s) and/or specification(s), e.g., ISO, ANSI, IEEE.

Validation: the process of checking the conformity of an implementation of a standard to its standard specification through conformance testing and, when compliance is demonstrated, issuing a certificate.

The following terms are defined with respect to PHIGS and their use within this document.

Generic standard: defines the semantics of the PHIGS functions at an abstract level, independent of any programming language. In this document, "the standard" refers to the generic standard.

Language binding: supplementary standard, defining the concrete syntax by which PHIGS functions are invoked from the host programming language.

Host language: the language in which a graphics program is written, and from which PHIGS functions are invoked.

PHIGS implementation: an actual graphics system which generally adopts the PHIGS model of graphics.

2. DESIGN GOALS

This section discusses the global characteristics which were to be built into the system. These goals do not dictate a unique design, but serve as constraints on the final shape of the test suite.

2.1 Scope of Version 1.0

Because of the size of the PHIGS specification (319 functions in the generic standard), we needed a test suite design that would allow incremental development. We partitioned the features of the standard into several areas, for each of which tests can be generated separately:

a. State semantics: the manipulation and reporting of data structures specified in the standard, such as state lists, description tables, archive files, and the centralized structure store (CSS).

b. Traversal: the implicit process in PHIGS that renders a visible image on an output device.

c. Graphical input: accepting data from the operator at run-time.

d. Error system: the detection and processing of erroneous conditions, both by default and by the user.

e. Metafiles: the generation and interpretation of files containing graphical data.

Since interactive tests (those requiring active judgment on the part of a human operator) are both more difficult to write and more difficult to run, we decided to defer testing traversal and graphical input. The error system and metafiles are separable from the core semantics of PHIGS, and so their testing is also deferred. Because version 1.0 was limited to tests of state semantics, it could be developed in a reasonable amount of time, those responsible for development could gain experience with the simpler features of the standard before going on to the more advanced parts, and the tests themselves could be almost fully automated.

2.2 Comprehensible Conformance Tests

It should be recognized that a test suite for a software system can be oriented toward several different purposes, such as conformance, performance, quality, and debugging. While any test suite is likely to be somewhat useful in all these areas, there are significant differences of design among test suites emphasizing each of these purposes.

In particular, conformance tests tend to be less informative than debugging or quality tests. For conformance purposes, the only issue is whether or not an implementation performs as required - essentially a pass/fail test. The PVT suite therefore emphasizes error detection, not error diagnosis or correction.

However, conformance tests do need to be informative about the relationship of the test to the standard. It is hardly acceptable to a user to execute a large opaque body of code, only to be told "TEST FAILS" without further explanation. We took it as a central goal not merely that the tests be logically correct, but also that interested users be given a way to see for themselves that the expected outcome is correctly grounded in the specifications of the standard.

2.3 Portability

Obviously the tests must be portable, as long as it is understood that this means portability with respect to the relevant standards (the generic PHIGS standard, the language binding standard, and the host language standard), not necessarily with respect to all putative implementations. So, even though the results of the tests are theoretically guaranteed by the standard(s), the code is *not* written to be portable in the more practical sense of working on as wide a variety of actual systems as possible. Only PHIGS features, however, are to be used up to the limit of the standard; the host language features are to be used conservatively, since they are not the object of the test suite.

2.4 Host Language

Because there will be a need for PHIGS tests for several language bindings, version 1.0 of the PVT suite will be as language-independent as possible. Specifically, the semantic requirements and program design (see below) are formulated in terms of the generic standard. Only the source code itself depends on the language binding. Thus, the documentation is valid for use with all language bindings.

Full Fortran was chosen as the language of version 1.0 for reasons of standardization. Its language binding was the most stable, and Fortran itself has the benefit of an established and widely accepted standard [FORT78]. Because there are few, if any, implementations of the Fortran subset binding, there are no plans to generate a version of PVT for subset Fortran. We do intend, however, to convert the PVT to other languages, such as C.

2.5 Operator Interaction

In general, operator interaction is to be minimized, but cannot be totally eliminated. After all, the primary semantic requirement of the standard is to generate graphic output and accept graphic input. These so-called "real effects" (as opposed to the logical effects on the state of the computation) are not easily susceptible to automated testing, and for the foreseeable future, their testing must depend on a human operator.

As discussed above in section 2.1, version 1.0 postpones this problem because it does not contain traversal or input tests. The major design goals of future versions of the PVT system will be to simplify operator interaction as much as possible by:

a. displaying clear instructions at run-time,

b. minimizing operator discretion insofar as the operator must be relied upon to judge the correctness of graphic output, and

c. recording automatically the results of interactive tests (i.e., no manual checklists).

Apart from graphics-oriented operator interaction, many PVT programs must have access to implementation-specific information, such as workstation types and connection identifiers. We took it as an important goal that such information should be entered only once; the operator should not have to specify information repetitively each time a test program is executed.

3. LOGICAL MODEL

One helpful way of understanding the PVT system is within the framework of deductive logic. In a logical system, the basic operation is that of *inference*. From a finite number of given premises, one can infer (or prove or deduce) a set (possibly infinite) of conclusions. If the premises are true and the inference is valid, then the conclusions must also be true. Conversely, if the inference is valid and the conclusion is false, then one of the premises relied upon in the inference must be false. This latter mode of reasoning is therefore often called falsification testing; it is the one typically used for checking the conformance of a software system against functional specifications.

3.1 Standardization as an Ideal Logical System

The PHIGS standard can be understood as a set of premises for a logical system. The premises are of the form: "For all X, if X is a conforming implementation, then X must behave in the following way: ..."; i.e., the standard defines the behavior of a conforming implementation. When we test an alleged implementation, P, we rely on the additional premise: "P is a conforming implementation." From these premises we can then infer a great deal about the behavior of P. If we discover that the actual behavior of P contradicts this theoretical behavior, we can then conclude that either

a. one of the relevant premises of the standard is false,

b. the premise that P conforms is false, or

c. the inference about the behavior of P is invalid.

As a practical matter, we rule out the first possibility: the standard is never wrong. This leaves the last two possibilities, which mean that, respectively, the implementation is non-conforming or the test is invalid.

3.2 Implementation of Logical Concepts in PVT

Although this outline is neat in theory, in practice there are a number of difficulties. First and foremost, the standard is not cast as a series of logical premises, but as a document composed of English prose. Second, in order to infer a seemingly simple conclusion, one often needs a large number of premises from the standard including several which are not of direct interest to the test at hand. Third, the proof involved in deriving the conclusion from the premises may be complex. Finally, it is difficult to relate all this logical machinery to the actual code.

Despite the impracticality of carrying out the full realization of the pure logical model, it serves as an ideal case which can suggest ways of structuring the actual system. In particular, the PVT uses the notion of semantic requirements (SRs), which play the role of the premises of the standard, and test case (TC) results, which play the role of conclusions.

3.2.1 Semantic Requirements

The PVT system consists of modules (see below), each of which contains a set of semantic requirements (SRs). These SRs represent a partial axiomatization of a given topical area of the standard: the premises, perhaps gleaned indirectly, for that section of the standard. Like logical premises, well-designed SRs should be:

a. Independent – We do not want to have one SR which implies another; if both are valid, we simply keep the stronger of the two, since the other is redundant.

b. Complete – The SRs should require everything that the standard requires.

c. Specific – Each SR must have some testable consequence, perhaps in conjunction with other SRs. Broad generalities are to be avoided in favor of lower-level concrete assertions.

d. Consistent – The SRs should not contradict each other.

e. Simple – An SR should not be a long list of requirements; to a reasonable extent, each SR should state an atomic rule about conforming implementations.

Even given these guidelines, there is no uniquely best way of formulating the SRs for a given module. Nor is this process, given the style in which the standard is written, automatable; some human judgment is required. Once a set of SRs has been generated, however, we do have a clear statement of the requirements to which an implementation will be held. It is understood, of course, that the SRs are expressing requirements for a conforming implementation; they are not used to state restrictions on PHIGS programs. Nor is it stated explicitly in each SR that "In order to conform, an implementation must..." – this is implicit.

3.2.2 Test Cases

From a single SR, it is usually impossible to infer any conclusion about the behavior of an implementation. Thus, it is generally not the case that an SR will be testable in isolation. The typical situation, rather, is that from a set of SRs some conclusion can be drawn which is directly testable. Such a conclusion is the basis for a test case (TC) within the module. Each module contains, not only a set of SRs, but a set of programs each of which contains in turn a set of TCs. These TCs are the executable core of the PVT system. Each TC consists of an explicit conclusion about the behavior of a conforming implementation, worded so as to state what "should" happen.

The important thing is to relate these TCs back to the SRs. Each TC comes with a list of SRs upon which its validity is based. There is, in general, a many to many relationship between SRs and TCs: one TC will typically depend on several SRs, and each SR will typically be used by several TCs.

Harking back to the logical model, each TC is associated with a set of SRs in order to suggest how a proof might be constructed that all conforming implementations succeed in the TC; and conversely, that failure in the TC implies failure to conform to the standard. These proofs are not, in fact, made explicit; the hope is that the inference is straightforward enough that users can "fill in" the derivation steps themselves between the premises (SRs) and the conclusion (TC).

As is often pointed out in the literature on validation, passing the TCs is no guarantee that an implementation really does conform; it might well violate the standard in untested areas. But failure in a valid TC does strictly imply failure to conform.

3.3 Background Assumptions

It would be very useful for debugging purposes if we could also conclude, not only that the implementation fails to conform, but that the reason it does so is that it violates at least one of the SRs of the TC (although without determining which one). Unfortunately, this isn't normally a guaranteed result. The problem is that the SRs explicitly named in the TC are usually not strictly sufficient to imply the validity of the TC. More often, there are a series of background assumptions, or implicit premises, to which we must appeal. This distinction between explicit and implicit testing complicates the use of the tests for debugging purposes. Below we shall discuss the various reasons that failure might be reported, other than that the implementation fails to conform to a valid, explicit SR.

3.3.1 PVT Assumptions

The first set of assumptions, about the PVT, amounts to saying that the tests are in fact valid tests of conformance. If these assumptions are incorrect, then, of course, one may not conclude that a failed test implies non-conformance on the part of the implementation.

3.3.1.1 Validity of SRs — An SR might be incorrect; that is, it might require behavior that is not actually mandated by the standard. We have, of course, tried to base the SRs on the standard, but there are cases in which its intent is questionable. The hope is that even if an SR is incorrect, at least its meaning is clear. This serves to sharpen any interpretation question which may emerge. These cases serve as feedback to the standardization process so that inconsistent or incomplete specifications in the standard may be corrected.

3.3.1.2 Validity of TCs — Even if a set of SRs is correct, a given TC might not be derivable from them (together with other background assumptions). The resolution would be either to fix the TC, or perhaps formulate a stronger SR to support the TC.

3.3.1.3 Validity of Code — Even if a TC is correct, the actual code which supposedly embodies it might not in fact really reflect the condition under test.

3.3.2 Implementation Assumptions

The second set of assumptions concerns aspects of the implementation not under explicit test. If these assumptions are incorrect, then one cannot conclude that the implementation fails to support one of the explicit SRs. However, the implementation still would not conform; it just fails to do so for unanticipated reasons, not those which were the purpose of the test.

3.3.2.1 Auxiliary PHIGS Functions — Very often, the tests rely on the establishment of a reasonable state of system. The tests and SRs do not refer to the need for "environment-setting" functions, such as <open phigs>, <open workstation>, and <open structure> when these are not the ones being tested explicitly. Note that we assume the proper operating state context for execution of functions in the wording of the SRs. For example, we say "<inquire ...> does X", not "If workstation is open, then <inquire ...> does X".

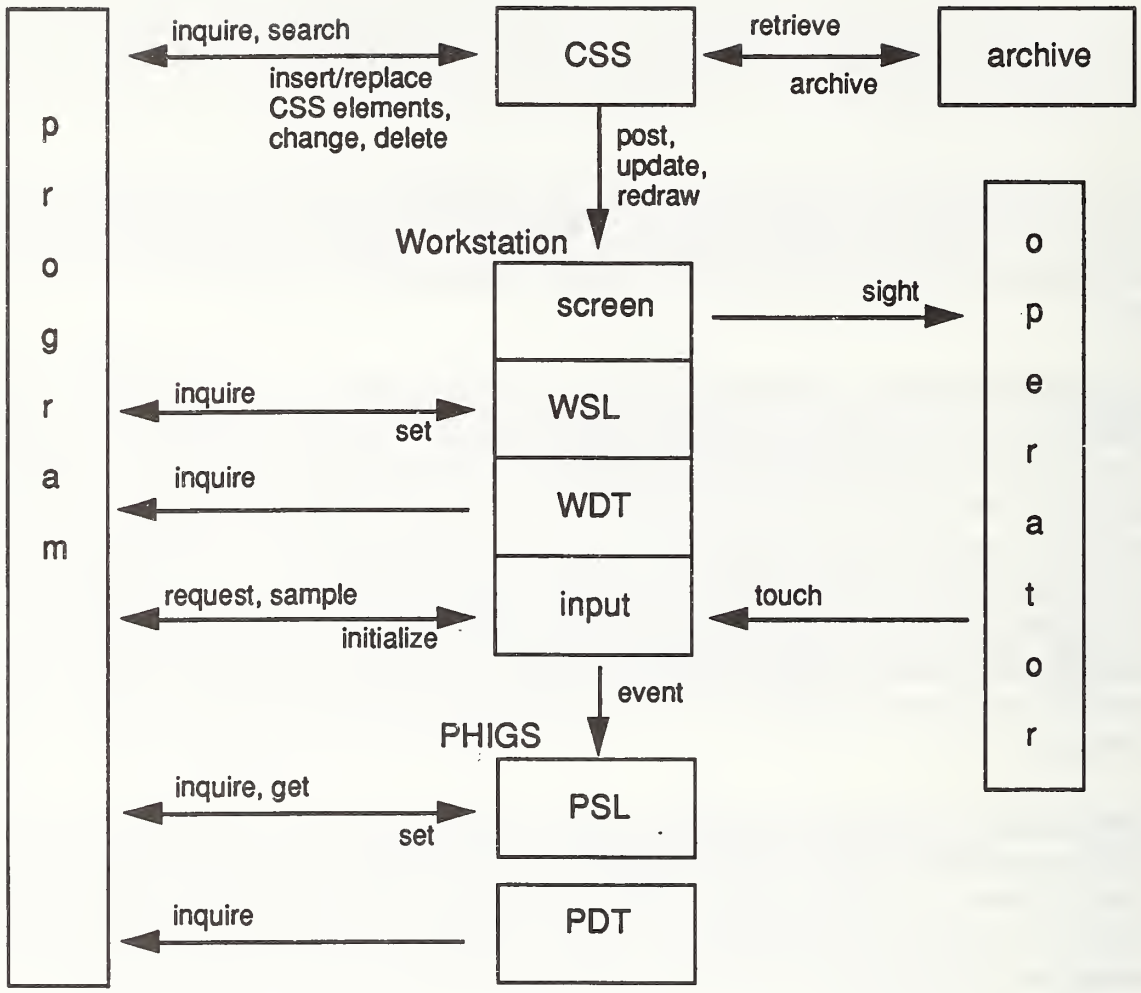
One must distinguish, then, between incidental and purposeful uses of PHIGS functions within the test suite. Most PHIGS functions cannot do anything in isolation, but only in a context established by other functions. When a routine invokes a function in order to test directly whether it works, it is a purposeful use. When a function is invoked simply to set up the environment for another function, it is an incidental use.

For example, the basic semantic requirement for <archive structure> is that it copy a structure from the CSS to an archive file. But in order to test this, other PHIGS functions must be used to set up the CSS in the first place, to retrieve the structure from the file, and to inspect the resulting CSS. Figure 1 illustrates the way PHIGS functions interact with the program, the various internal data structures, and the operator.

When a TC is failed, its SRs identify which feature or combination of features is being purposefully tested and how so. This is the probable cause of failure, but of course other functions being used incidentally cannot be ruled out. Nonetheless, even if the cause of failure cannot be certain, the fact of failure is.

3.3.2.2 System Resources — Virtually every implementation will depend on the proper working of system software, such as language compilers, linkers, loaders, and the like. Similarly, an actual system normally has a working terminal and other physical realizations of such PHIGS abstractions as workstations. Even though these are not used only by PHIGS, they are nonetheless part of the complete PHIGS implementation. To put it succinctly, an implementation is not just a subroutine library. It is, rather, any system capable of accepting a PHIGS program and producing the mandated behavior, including the internal results which may be checked directly by the program, or the external results which can be checked only by human inspection. The components of such a system are not relevant to conformance; only the implementation as a whole conforms, or fails to conform, to the standard.

As an extreme example, note that the PHIGS standard mandates that an implementation have at least one OUTIN workstation. If there is only one terminal which realizes that workstation and it malfunctions, at that moment the implementation ceases to conform. Any test which relies on the existence of an OUTIN workstation will correctly signal failure, even if the reason for failure may not be found among the explicit SRs.



Edges are labeled with operation that causes data movement. Except for *sight* and *touch*, these are all PHIGS functions.

CSS: Centralized Structure Store
PSL: PHIGS State List
PDT: PHIGS Description Table
WSL: Workstation State List
WDT: Workstation Description Table

Figure 1. Data flow within PHIGS.

4. PVT ARCHITECTURE

Given the design goals and the logical model set forth, the question becomes how to realize these in an actual system of software and documentation. The representation of the abstract entities should be easily accessible by both automatic processes and the human user.

4.1 Modularity

The set of all SRs for the entire PHIGS standard is far too large to be handled as a single entity. Rather, they are divided into topically coherent subsets, which we call modules. Ideally there would be no interaction between modules and strong logical interdependence within each module.

Recalling the many-to-many relationship between SRs and TCs, we strive to keep each TC dependent mainly on the SRs of its own module (these are the SRs being explicitly tested). Thus, the rationale for grouping a set of SRs and TCs into one module is that their functions and data structures must be tested together, i.e., their behavior is strongly interrelated. Hence, the module's set of SRs and TCs form a "natural cluster."

Even though all the SRs of a module are closely related, it will often happen that the TCs are numerous or diverse enough that they need to be further subdivided into several programs. How should this subdivision take place? At one extreme we could put one TC per program, since a program with several TCs may terminate abnormally before finishing, thus leaving some conditions untested. Such an approach would incur a great deal of overhead, however, especially given the fact that often a complex computational state is set up and then several TCs are executed based on it.

At the other extreme, if we wish to minimize the operator's effort, we could put all the TCs of a module, or even of the entire system, into one program. This, too, seems impractical, given the resource limitations of many systems, and the resulting difficulty for human readers. We tried to strike a reasonable balance, grouping closely related TCs within single programs, yet keeping the programs fairly small and comprehensible.

Thus, the PVT suite is organized as a set of logically independent modules, each of which contains a set of SRs and a set of TCs, with the TCs distributed among several programs. Concretely, each module will consist of a single documentation file containing the statement of the SRs and the design of the programs, and several source code files, one per program.

4.2 Tree Organization

The standard is not just an undifferentiated set of requirements, but has clearly delineated functional areas. We considered using the sequence of function definitions in section 5 of the standard [PHIGS88] or the data structures of section 6 as an organizing principle, but it seemed finally that the more conceptual organization of section 4 provided the best model, especially given the goal that each module deal with all the strongly related requirements of a topic. For instance, a set and inquire function often must be tested together and obviously belong in the same module, even though in section 5 all the inquires are treated as a separate group.

The standard, like many documents, has a hierarchical structure. Also, most computer systems provide a hierarchy for their file systems. It therefore seemed reasonable to take advantage of this built-in order and organize the modules into a topical hierarchy, or tree, to be realized using the tree structure of the file system.

4.3 Order

Although it has some advantages for debugging purposes, we have *not* adopted a test-before-use regime. First, it is unclear, given the interdependence of the PHIGS functions, whether such a program could really be carried out. Second, test-before-use does not, as is sometimes claimed, infallibly isolate the cause of an error; follow-up testing is typically needed to determine exactly what went wrong. Third, by dispensing with the test-before-use rule we can make the test structure line up coherently with the standard's own conceptual structure. Given our emphasis on conformance rather than debugging, we believe that it is more important to maintain a simple relationship between the test system and the standard than to try to isolate the cause of failures more precisely than is otherwise done.

4.4 Database Model of PVT

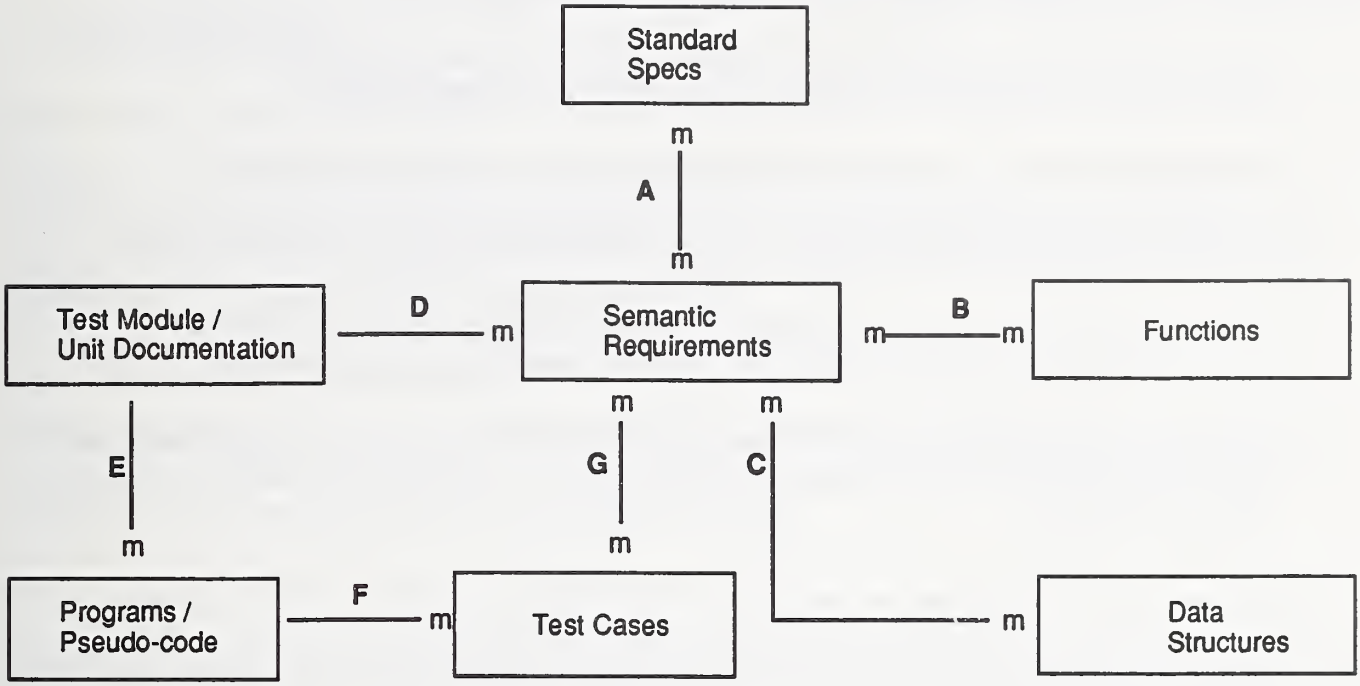
Given that the PVT structure is not tied directly to the order of functions in the standard, or to that of the data structures, it seemed useful to try to provide a cross-reference index into the system for the functions and data structures. Likewise, even though the PVT structure resembles that of one section of the standard, it is still useful to have an explicit detailed cross-reference between the SRs and the text of the entire standard.

The SRs anchor the system; they specify the precise behavior of PHIGS functions and data. Therefore, the SRs also serve as the reference points for related entities. Specifically, each SR is annotated with a list of related functions, data structures, and text from the standard. By adopting a canonical numbering of the functions and data structures and documenting the references according to a well-defined format, we allow the cross-reference tables to be built automatically once the original annotation is done. Besides enabling users to navigate within the system, an equally important goal of this approach is that it gives us a good coverage metric: we can see which functions and data structures have been probed by the total PVT system.

It is useful to think of the PVT system as embodying a database. Figure 2 contains a schematic diagram exhibiting the main features of this database, namely its entities and the relationships among them. Note the central role of the SRs.

4.5 Message System

If a test program is to be a practical tool for conformance measurement, it must communicate its findings to the user in a simple and convenient way. We settled on the notion of a *message* as the basic unit of output of a PVT program. Besides the English content of each message, the two dimensions of interest are type and destination.



A lowercase "m" next to an entry indicates that there may be many instances of the entity. Thus, the D relationship between Test Module and Semantic Requirements is one-to-many; the B relationship between Semantic Requirements and Functions is many-to-many.

Relationships:			Captured by:
A SR	is derived from	Standard Specs	#S in SR
B SR	depends on	Function	#F in SR
C SR	depends on	Data Structure	#D in SR
D Module	tests	SR	SR in DOC.TXT
E Module	contains	Program	design in DOC.TXT
F Program	contains	TCs	SETMSG in code
G TC	directly tests	SR	SRs in SETMSG (also #T in SR)

Figure 2. The PVT as a database.

While the core purpose of a test program is to notify the user of any failed test cases, there are several other kinds of information that need to be conveyed. For example, the user might also want to know which program is running, what test cases were passed, unanticipated circumstances, and general information about the implementation. To accommodate these possibilities each message is categorized as one of six types, each type denoted by a two-character code:

1. System (SY) messages note the beginning and completion of execution of each PVT program and are used to summarize the number of TCs executed and errors detected.

2. OK messages indicate a passed test case. Messages of this type can be suppressed.

3. Error (FA) messages indicate a failed test case, resulting from violation of explicit testing.

4. Unanticipated (UN) messages are generated when the program detects some anomalous condition that prevents further processing, but does not imply non-conformance.

5. Unanticipated non-conformance (NC) messages are generated when the program detects the failure of some PHIGS function being implicitly relied on (not explicitly tested) that prevents further processing.

6. Informational (IN) messages are used for all other communication.

The second dimension is that of destination; where are the messages to be sent? We envisage several plausible possibilities:

1. The operator may want to see the messages at run-time.

2. There could be an individual message file for each test program execution.

3. There could be a global message file which is used to accumulate the results of all the programs.

The decision on message destination seemed to be one best left to the operator. Accordingly, we provide a means during system initialization for the operator to decide which combination of these three will receive messages. Each time a message is generated it is broadcast to the appropriate files. Every enabled destination receives exactly the same set of messages.

5. DEVELOPMENT TOOLS

Given the highly integrated nature of the PVT system, we could not hope to maintain internal consistency based only on human review. As development progressed, we implemented and relied heavily on a set of tools to generate and check various parts of the system. Brief descriptions of the more important of these follow.

5.1 Code Outline Generator

The generator produces a plausible source code outline from the program design of the documentation file. All the formatted information, such as program title, and the wording of test cases is generated in correct Fortran. Other parts of the design are rendered as comment lines, allowing the program author to follow closely the documented logic.

5.2 Module and Program Checker

The module checker runs a variety of compatibility and format checks on the documentation file and program files within a single module. The program checker's main job is to perform data type checking on the parameters in the calls to PHIGS functions.

5.3 Cross-Reference Tools

The module preparation program scans the document file and builds cross-reference files for each of the various entities to be linked to the SRs. The module post-processing program alters the documentation file to include some of the cross-reference information. In addition, global cross-reference files are built from the local files for each module and distributed as part of the PVT documentation.

5.4 Re-number SRs

As a module is designed, it is common to insert, delete, and re-order the set of SRs within it. Since these are referred to by the TCs, this can become a cumbersome process, because the references must be updated to agree with the new SR numbers. The re-numbering tool performs this operation automatically.

6. SUMMARY AND INTENTIONS

When developing a validation test suite, the designer(s) must create a test system that "fits" the standard while also possessing some plausible internal structure. The PHIGS Validation Test suite was set up as a semantic hierarchy, into which the features of PHIGS are organized. This document has presented the rationale for this design, our goals, and the tradeoffs we made. Moreover, we described the PVT system itself, including the logical model on which it is based and the tools we used in its creation.

This first version of the PVT system is concerned with state semantics and requires minimal operator interaction. Future versions would encompass interactive tests, focusing on traversal and graphical input. To complete the PVT system, tests for the error system and metafiles would also need to be developed.

Although the PVT suite is designed in a language-independent manner, it is coded in FORTRAN. We plan to convert the PVT to other languages, with C being the most probable choice as the next available host language.

Finally, we intend to use this PVT suite to validate PHIGS implementations which have been submitted for testing and to make it available to other accredited testing laboratories for the same purpose. Additionally, the suite will be available to other organizations for use in developing and testing PHIGS implementations.

7. REFERENCES

[CUGI90] John Cugini, Mary T. Gunn, Lynne S. Rosenthal, *User's Guide for the PHIGS Validation Tests*, NISTIR-4349, National Institute of Standards and Technology, Gaithersburg, MD, 1990.

[FED88] *Federal Information Processing Standards: Conformance Testing Policy and Procedures*, Federal Register, Vol. 53, No. 149, August 3, 1988.

[FORT78] *Programming Language FORTRAN*, ANSI X3.9-1978, American National Standards Institute, New York, NY, 1978.

[GKS85] *Computer Graphics - Graphical Kernel System (GKS) Functional Description*, ANSI X3.124-1985, American National Standards Institute, New York, NY, 1985.

[GKST89] *GKS Validation Test Suite, Version 2.1*, National Institute of Standards and Technology, Gaithersburg, MD, 1989.

[PHIGS88] *Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) Functional Description, Archive File Format, Clear-Text Encoding of Archive File*, ANSI X3.144-1988, American National Standards Institute, New York, NY, 1988.

NIST-114A
(REV. 3-89)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NIST/SP-500/181

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
July 1990

4. TITLE AND SUBTITLE

PHIGS Validation Tests (Version 1.0): Design Issues

5. AUTHOR(S)

John Cugini, Mary T. Gunn, and Lynne S. Rosenthal

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED
Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

Same as item 6.

10. SUPPLEMENTARY NOTES

[ ] DOCUMENT DESCRIBES A COMPUTER PROGRAM; SF-185, FIPS SOFTWARE SUMMARY, IS ATTACHED.

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

Conformance testing for the Programmer's Hierarchical Interactive Graphics System (PHIGS) standard presents certain novel difficulties, especially the indirect effect of many functions, and the inaccessibility to the program of visual effects. The model of logical inference offers a way to organize a system of the complexity needed to overcome these problems. This complexity makes the use of certain database concepts quite valuable in allowing users to comprehend the system. Special emphasis is placed on allowing the user to associate each test case with some specific requirement in the standard. Test output consists of a set of formatted messages that enable the user to assess test results rapidly and accurately.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

conformance testing; graphics standards; PHIGS; testing of software; validation of software

13. AVAILABILITY

[X] UNLIMITED

[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

[X] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

[X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
21

15. PRICE

ELECTRONIC FORM

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NIST Technical Publications

Periodical

Journal of Research of the National Institute of Standards and Technology—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW., Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NIST research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NIST publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NIST Interagency Reports (NISTIR)—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, MD 20899