X WINDOW SYSTEM™ VERIFICATION SUITE FOR X11R5 (VSW5)

USER’S GUIDE

Version 5.1.1

December 1999

The Open Group

REVISION HISTORY

RELEASE	DATE	AREA	MODIFICATIONS
5.1.1	December 1999		Maintenance Release
5.0.2	December 23, 1998		Maintenance Release
5.0.1	February 27, 1998		Maintenance Release
		Section 4.1.2	Added description of XT_COVERAGE variable.
		Section 5.2	Added description of Interface scenario.
		Chapter 8	Material on submitting SRs and IRs changed to point to the appropriate WWW URLs.
5.0.0	December 15, 1995		Initial Release

CONTENTS

1.0 INTRODUCTION

This document provides operational information for users of the X Window System Verification Suite for X11R5 (VSW5). Review of the entire document is suggested before installing or executing VSW5.

Chapters 3 and 4 of this Guide provide step-by-step instructions for installing and configuring VSW5. Chapter 5 provides instructions for executing VSW5. These procedures should be followed carefully to ensure VSW5 operates properly.

1.1 X WINDOW SYSTEM STANDARDS OVERVIEW

The X Window system is a network-transparent window system developed under the auspices of Project Athena at the Massachusetts Institute of Technology. The Open Group's specification for the X Window System is collectively known as the Window Management (X11R5) document set and is comprised of four documents:

- *X Window System Protocol*
- *Xlib - C Language Binding*
- *X Toolkit Intrinsics*
- *File Formats and Application Conventions*

These specifications are based on the MIT documentation for the X Window System Version 11, Release 5 (X11R5), with some additional Open Group requirements for Xlib (as defined in Chapter 18 of the *Xlib - C Language Binding* specification).

1.2 VSW5 OVERVIEW

VSW5 verifies conformance of implementations to the requirements of the Open Group Window Management (X11R5) document set. Tests are also provided for two X extensions: The Input Device Extension and the Shape Extension. Extension tests are not required for formal branding as these interfaces are not yet covered by the Open Group Window Management (X11R5) document set.

1.2.1 VSW5 Lite

VSW5 Lite is a subset of VSW5 intended for users needing to brand only the Open Group X Window System Display Component. VSW5 Lite is identical to VSW5 except that tests for the X Toolkit Intrinsics and SHAPE extension are not included. Installation, configuration and execution procedures are also the same as for VSW5, except that not all VSW5 scnearios are available to VSW5 Lite users (see section 5.2 on page 42) and some VSW5 build configuration variables are not used by VSW5 Lite (see section 3.5 on page 14).

1.2.2 Intended Users

Every effort has been made to make VSW5 easy to use and provide information helpful in troubleshooting implementation conformance issues. However, VSW5 is a complex product that performs an intricate and sophisticated task. It is intended for technically knowledgable users familiar with the details of the implementation under test.

In order to install and run VSW5 a user will need to have a thorough knowledge of:

- The configuration of the implementation under test.
- The commands and utilities available on the implementation, particularly the development environment.
- The operation of the APIs VSW5 verifies as provided by the implementation.

An experienced system administrator with substantial experience working with the implementation is the suggested profile for this sort of VSW5 user.

A second type of VSW5 user will be involved in analyzing and rectifying implementation conformance issues reported by VSW5. The reports produced by VSW5 are designed to be beneficial to a user with substantial operating system development experience. Evaluating this information and using it to understand and correct operating system conformance issues requires this sort of background.

1.3 DOCUMENT OVERVIEW

1.3.1 Audience

This Guide is intended for users of VSW5. Information is provided on installing, configuring, building and executing VSW5 as well as troubleshooting conformance issues it reports.

VSW5 is implemented under the Test Environment Toolkit (TET) test harness. Readers are assumed to be familiar with TET. VSW5 is able to run under TET3 as well as TET1.10.

Readers should also be familiar with the Open Group Window Management (X11R5) document set.

1.3.2 Document Structure

This Guide is divided into eight chapters and an appendix:

• Chapter 1	is this introduction.
• Chapter 2	provides an overview of VSW5.
• Chapter 3	provides procedures for installing VSW5.
• Chapter 4	provides procedures for configuring VSW5.
• Chapter 5	provides procedures for running VSW5.

• Chapter 6	describes the tools VSW5 provides for generating reports.
• Chapter 7	provides suggestions for troubleshooting problems.
• Chapter 8	provides procedures for the support services available to VSW5 users.
• Appendix A	provides a glossary of the terms used in this Guide.

Information which would be required by a programmer to modify or extend VSW5 is contained in a separate document: *X Window System Verification Suite for X11R5 (VSW5) Programmer’s Guide*.

1.3.3 Relationship to TET documentation

This Guide describes the relationship between VSW5 and TET but does not describe TET itself. Useful TET documents are listed below (see section 1.4 on page 3).

1.3.4 Relationship to the VSW5 Release Notes

Each release of VSW5 is accompanied by Release Notes. VSW5 Release Notes provide information on changes in VSW5 since the last release, known problems, and special procedures. The Release Notes for the release of VSW5 being used should be read before VSW5 is installed.

1.3.5 Relationship to VSW5

This Guide is updated periodically as needed, but not necessarily for each release of VSW5. As such the version number of the Guide may not correspond to the VSW5 release number. The Release Notes for each release of VSW5 define the version of this Guide that is current at that time. That version of the Guide is included in the release.

1.3.6 Relationship to the VSW5 Assertions

The coverage of VSW5 is defined by assertions - statements which specify the implementation behavior which is tested. The assertions themselves are included in the VSW5 release (see section 2.5 on page 8).

1.4 RELATED DOCUMENTS

This Guide is intended as a stand-alone document. It may, however, be beneficial to read it with reference to the following other documents:

1) Open Group Window Management (X11R5) document set:

   - *X/Open CAE Specification, Window Management (X11R5): X Window System Protocol* (ISBN: 1-85912-087-3, X/Open document number: C507). X/Open Company Limited, 1995.

   - *X/Open CAE Specification, Window Management (X11R5): Xlib - C Language Binding* (ISBN: 1-85912-088-3, X/Open document number: C508). X/Open Company Limited, 1995.

- *X/Open CAE Specification, Window Management (X11R5): X Toolkit Intrinsics* (ISBN: 1-85912-089-3, X/Open document number: C509). X/Open Company Limited, 1995.
- *X/Open CAE Specification, Window Management (X11R5): X Window File Formats and Application Conventions* (ISBN: 1-85912-090-3, X/Open document number: C510). X/Open Company Limited, 1995.

2) IEEE Std. 1003.3-1991 *Standard for Test Methods for Measuring Conformance to POSIX*. Institute of Electrical and Electronic Engineers, Inc., 1991.

For TET1.10

3) *Test Environment Toolkit: Architectural, Functional and Interface Specification*. X/Open, OSF, UNIX International, 1992.

4) *Test Environment Toolkit Programmer's Guide.* X/Open, OSF, UNIX International, 1991.

For TET3

5) *TETware Design Specification, Revision 1.0.* X/Open Company Limited, 1996.

6) *TETware User Guide, Revision 1.1.* X/Open Company Limited, 1997.

1.5 EDITORIAL CONVENTIONS

In this Guide different type styles and renditions are used to denote different implementation components:

- The names of files (including commands and devices) are shown in a fixed width font, for example: `/dev/console`.
- The names of functions are shown in italics, for example: *gethostbyname*.

This symbol is used when especially critical information is provided.

2.0 VSW5 OVERIEW

VSW5 is a source level system test suite. Its objective is to verify systems are compliant to the Open Group Window Management (X11R5) document set such that application programs in source form will be portable across these systems. In order to ensure application source portability VSW5 verifies the behavior of all required functions and macros and the ability to build and execute an application which accesses only required functions, macros, and variables.

VSW5 verifies the conformance of a system through building and running a series of many thousands of tests. Each of these tests verifies a specific requirement of the Open Group Window Management (X11R5) document set. To pass VSW5 a system must successfully build and execute each of the VSW5 tests. Having passed VSW5 a system can be counted on to build and execute an application which is compliant to the Open Group Window Management (X11R5) document set without requiring any change to the application.

The tests VSW5 performs are defined by *assertions*. VSW5 assertions are formal statements of the requirements of the Open Group Window Management (X11R5) document set in terms of what must be tested to ensure portability. Each VSW5 test has a corresponding assertion which in sum provide a formal specification of VSW5’s test coverage.

2.1 USE OF THE TEST ENVIRONMENT TOOLKIT (TET)

The Test Environment Toolkit (TET) is a widely used framework for implementing test suites. TET provides a common environment for test users and developers and allows tests from different sources to be easily integrated together.

TET consists of two major components: libraries of functions for test implementation and a utility known as the Test Case Controller (TCC). The TET libraries are linked with a test suite when it is built. The TET TCC is the mechanism by which a user executes tests which have been built with the TET libraries.

The VSW5 tests are built and executed using TET. TET coordinates running the VSW5 tests and gathering the results they generate. TET provides simple commands for running VSW5 and allows reports to be generated which summarize the results of testing.

The remainder of this section assumes familiarity with TET. TET must be installed on the implementation prior to the installation of VSW5. VSW5 does not include TET. VSW5 is able to run under TET3 as well as TET1.10.

VSW5 is implemented directly under TET:

- A Test Purpose is provided for every assertion.
- Test Purposes are implemented using the TET POSIX_C API.
- Test Purposes return TET result codes. Three VSW5-specific result codes (ABORT, FIP and WARNING) are defined (see section 6.1.3 on page 46).

- Configuration variables are maintained in TET configuration files.
- VSW5 is built and executed using the TET TCC.
- The output of VSW5 is a TET journal file.

TET provides a basic testing environment and test suites often layer additional levels of framework on top of TET. Information on the additional testing mechanisms used in VSW5 can be found in the *X Window System Verification Suite for X11R5 (VSW5) Programmer's Guide*.

2.1.1 Journal Files

The results of executing VSW5 are captured in a TET journal file. VSW5 places special information in the journal file that is used by the VSW5 report generators (see section 6.0 on page 45). This information gives the report generators access to the names and numbers of tests and allows invariant Test Purpose numbering regardless of build failures.

This information is produced through comments in the scenario file and *tet_infoline* calls in the tests themselves. Thus VSW5 journal files are TET-standard and can be processed with any utility that accepts TET journals. However only the VSW5 report generators will process the VSW5 special information and thus their use is required for producing formal conformance reports.

The VSW5 report generators rely on the journal file format produced by VSW5 and can only be used with journals created by VSW5.

2.1.2 The Test Case Controller

VSW5 is built and executed using the TET TCC.

Care should be exercised in using the more complex TCC options (e.g. -l) to ensure the additional information normally emitted by the VSW5 scenario file is written to the journal file.

2.2 STRUCTURE

The Test Cases in VSW5 are grouped into sections corresponding to areas of the Open Group Window Management (X11R5) document set and other areas of X:

Xlib3-17	tests for functions in Chapters 3 through 17 of the *Xlib - C Language Binding* specification.
Xopen	tests for Open Group-specific requirements in Chapter 18 of the *XLib - C Language Binding* specification.
Xt3-15, XtC, XtE	tests for requirements in Chapters 3 through 15 and Appendices C and E of the of the *X Toolkit Intrinsics* specification.
Xproto	tests for requirements in the *X Window System Protocol* specification.

XI	Tests for the X Input Device Extensions APIs.
XIproto	Tests for the X Input Device Extensions protocol.
SHAPE	Tests for the Shape Extension APIs.

TET scenarios are defined for each VSW5 Section, each VSW5 Test Case, and the full test suite.

2.3 LEVEL OF TESTING

Three levels of testing coverage are applied in VSW5 (the terminology is as per POSIX 1003.3):

EXHAUSTIVE	Full testing of all possible aspects of a requirement. For example testing every possible value for an argument to a function.
THOROUGH	Testing sufficient to validate that the implementation under test matches a requirement. For example, testing boundary conditions and other representative values for an argument to a function.
IDENTIFICATION	Testing only that the implementation under test provides the mechanism necessary to meet a requirement. For example, testing that a function exists without validating its processing of arguments.

VSW5 provides Exhaustive coverage where practical but in many areas this is not feasible due to the time and system resources required. Most VSW5 testing is performed at the Thorough level. This is also the case in test suites such as PCTS and VSU4. While this represents some compromise of completeness in the testing performed implementations are rarely sensitive to the difference in coverage.

Identification testing is employed in VSW5 only where no other requirements are defined, e.g. a requirement that a symbolic constant must be defined in a header file without a specification of its value.

2.4 DIRECTORY STRUCTURE

VSW5 is installed in the $TET_ROOT directory. The structure of the VSW5 directory tree, in addition to the standard TET files, is:

bin	contains tools used in implementing VSW5.
doc	contains this Guide and the VSW5 Programmers Guide.
fonts	contains fonts used by VSW5.

lib	contains libraries used by VSW5.
man	contains man pages for the utilities contained in VSW5.
src	contains the source code for include files, tools, and libraries used in implementing VSW5.
tset	contains the Test Cases for VSW5. Test Cases for each of the VSW5 Sections are kept in their own sub-tree. Within each Section directory are directories for each element defined by that Section (header file, function, etc.) which contain the actual Test Case files.

2.5 ASSERTIONS

VSW5 assertions are defined and implemented in accordance with POSIX 1003.3. An assertion is defined for each requirement covered by VSW5. The general requirements applied in generating the VSW5 assertions are as follows:

- An assertion exists for each definitive statement in the standard that pertains to a system implementation.
- Each assertion corresponds specifically to language in the standard. The language used in the standard is used in the assertion where possible.
- An assertion is boolean, that is, either true or false for a specific implementation.
- An assertion describes the smallest piece of functionality possible.
- An assertion covers an area no smaller than a specific functional requirement, even if the test method used to validate that requirement is complex. For example, a requirement that a function return the cube root of a number might be tested by passing several values to the function and verifying the result for each of these values. This requirement would be captured with a single assertion, not one for each value used.
- An assertion defines functional requirements, it does not explain them. Commentary, explanation, etc. is not included in assertions. A statement in the specification that is a warning to programmers or is an implementation recommendation does not generate an assertion.

2.5.1 Assertion Classification

POSIX 1003.3 classifies an assertion in two ways:

- Whether the assertion must be tested (a base assertion) or can be left untested (an extended assertion).
- Whether the assertion must be true on all conforming systems (a required assertion) or must be true only if a conforming system provides functionality defined as optional by the standard (a conditional assertion).

Each assertion thus falls into one of four classes:

- Class A: Base required assertion
- Class B: Extended required assertion
- Class C: Base conditional assertion
- Class D: Extended conditional assertion

2.5.2 Generating Assertion Files

VSW5 assertions are contained with the source code of the VSW5 Test Cases. Each Test Case directory contains a source code file with a name ending in .m The command

```
ma -o outfile -h file.m
```

produces a file `outfile` containing the assertions from the file .m file `file.m.`

For example:

```
ma -o stclpmsk.a -h stclpmsk.m
```

outputs to the file `stclpmsk.a` a list of the assertions from the source code file `stclpmsk.m`.

The assertions are output in nroff format.

2.5.3 Printing Assertion Files

An assertion file generated as described above can be printed with a utility accepting nroff format files as input. No other macros are necessary beyond those contained in the assertion file. For example:

```
nroff stclpmsk.a
```

3.0 INSTALLING VSW5

This chapter provides instructions for installing VSW5. An installation checklist is provided at the end of the chapter. This checklist is an aid to ensuring the procedures in the body of the chapter are successfully completed.

3.1 REQUIRED RESOURCES

In order to execute VSW5 an implementation must provide the resources described below.

These resources are incremental to the requirements the governing specifications place on an implementation. While these resources must be present on the configuration used for running VSW5 to provide a viable environment for testing, this is not intended to expand the specification, nor does it imply that these resources must be present when conforming implementations are delivered to customers.

For procedures for configuring VSW5 to use these resources see section 4.0 on page 29.

3.1.1 TET

TET must be installed on the implementation before VSW5 is installed. If the implementation supports multiple object file formats (e.g. COFF and ELF) TET must be built in an object file format compatible with how VSW5 will be built. VSW5 is able to run under TET3 as well as TET1.10.

3.1.2 Disk Space

The amount of disk space needed for VSW5 varies from release to release and implementation to implementation. Disk space requirements for each release are provided in its Release Notes. Guidelines are as follows:

• VSW5 distribution (compressed tar)	6MB
• VSW5 source tree (uncompressed)	20MB

VSW5 should be run so as to remove executables as they are built (see section 5.3 on page 43), so the disk space allocated for the test suite does not need to accommodate the large (500MB+) amount of space the executables would consume. Note however that VSW5 journal files are large, the journal file from building and executing the “all” scenario for example is roughly 8MB. Disk space should be allocated to accommodate a reasonable number of journal files. The allocation of at least 100MB of disk space for VSW5 is recommended.

3.1.3 Commands and Utilities

VSW5 uses the following utilities provided by the implementation. These must be accessible through the PATH environment variable when VSW5 is built and executed.

<u>Bourne shell</u>

Configuring and building VSW5 includes operations which have only been tested using the Bourne shell.

The build configuration file sets the SHELL variable so that the Bourne shell will be used by `make`. No other settings for this variable have been tested.

<u>make</u>

Building VSW5 requires the `make` command.

<u>C Compiler</u>

A C compiler and link editor are required. VSW5 assumes that when these utilities execute successfully they return a value of zero. The names of these utilities may be set in the build configuration file.

<u>Library archiver</u>

A library archiver and a means of ordering the libraries are required. The ordering software may be part of the library archiver, the `ranlib` utility, or the utilities `lorder` and `tsort`. The names of these utilities may be set in the build configuration file.

<u>File Utilities</u>

VSW5 uses utilities to copy, move, remove and link files. The names of these utilities may be set in the build configuration file.

In addition to these utilities the `tar` and `uncompress` commands are required to install VSW5.

3.1.4 Semaphores

VSW5 requires a set of semaphores. At least 8 semaphores must be configured in this set. Note these semaphores are not necessary if using VSW5 Lite.

3.2 INSTALLING THE VSW5 FILES

VSW5 must be installed in the $TET_ROOT directory of a TET1.10 or TET3 directory tree.

TET must be installed before VSW5 can be installed

The VSW5 files do not need to be owned by a special user or group, they can be installed by any user. However the owner of the VSW5 files MUST NOT be privileged (e.g. cannot be root). When VSW5 is executed the user running it must be in the same group as the user that installed it. This should be taken into account in selecting the user to own the VSW5 files.

VSW5 is distributed as a compressed tar file. To install the VSW5 distribution:

1) Copy the distribution file (`VSW5nnn.tar.Z` or `VSW5LTnnn.tar.Z` where nnn is the release number) into the $TET_ROOT directory.

2) Unpack the distribution:

```
zcat VSW5nnn.tar.Z |tar xof -
```

or

```
zcat VSW5LTnnn.tar.Z |tar xof -
```

This creates a directory `vsw5` in the $TET_ROOT directory which is the root of a directory tree containing the VSW5 files.

3.3 SURPRESSING EXTRANEOUS COMPILER MESSAGES

If the C compiler on the implementation outputs messages during compilation other than warnings and errors (e.g. copyright messages or other identification strings) these messages need to be suppressed for use with VSW5. Extraneous messages will result in FIP results. If these sorts of messages are being produced replacing the compiler with a script which will filter them out after invoking it is recommended.

3.4 CHECKING SUPPORT FOR THE XTEST EXTENSION

If the implementation under test supports the XTEST extension VSW5 will be able to perform tests for some assertions which are otherwise untestable. The XTEST extension has been produced by MIT since the initial release of X11R5. It provides access to the X server to enable testing of the following areas of the X Window System:

- Those which rely on the simulation of device events.
- Those requiring access to opaque client side data structures.

In order for VSW5 to use the XTEST extension two conditions must be true:

- The X server supports the XTEST extension. This can be verified by printing the list of extensions available in the X server using the X utility `xdpyinfo`. Note - the name of the extension should be printed exactly as in this User Guide - there are other testing extensions for X which are not compatible with the XTEST extension.

- The implementation has the required libraries to link the test suite clients so as to access the XTEST extension. All test suite clients must be linked with Xlib, which is often named `libX11.a`. To access the XTEST extension two further libraries are needed. These are the XTEST library (often named `libXtst.a`) and the X extension library (often named `libXext.a`).

If both these conditions are true for the implementation VSW5 should be configured to use it through the SYSLIBS (see section 3.5.6 on page 18) and DEFINES (see section 3.5.7 on page 20) configuration variables.

3.5 SETTING BUILD CONFIGURATION PARAMETERS

TET builds tests using a *build tool*. The build tool for VSW5 is a utility named `wbuild` which is supplied in the directory `$TET_ROOT/vsw5/bin`. The `wbuild` utility is an interface to the `make` system command which builds a test using the rules provided in a Makefile. The `wbuild` utility may be invoked directly from the shell, as well as via TET, to build individual parts of VSW5.

Each Makefile in VSW5 is written using symbolic names to describe commands and parameters which may vary from system to system. The values of these symbolic names are all obtained by `wbuild` from the build configuration parameters in the TET build configuration file for VSW5: `$TET_ROOT/vsw5/tetbuild.cfg`. Before proceeding with the installation of VSW5 edit this file as described below to reflect the requirements of the implementation under test.

Some paramaters are marked as not used by VSW5 Lite. Note these paramaters must still be defined in `tetbuild.cfg` though they may be set to any value.

3.5.1 Configuration parameters defined by TET

None of these parameters may be changed. They are already set to values which are correct for VSW5.

TET_BUILD_TOOL	The name of the program that the TET will execute in build mode. This must be the `wbuild` utility that is supplied with VSW5.
TET_BUILD_FILE	Flags required by the build tool. This parameter must be empty.
TET_CLEAN_TOOL	The name of the program that TET will execute in clean mode. This must be the `wclean` utility that is supplied with VSW5.
TET_CLEAN_FILE	Flags required by the clean tool. This parameter must be empty.

TET_OUTPUT_CAPTURE	Used by the TET to determine if the output from the build tool is to be saved and copied into the journal file. This parameter must be set to TRUE.

3.5.2 X11 Release Number

XT_X_RELEASE	The release number of X11 supported by the implementation, e.g. 5 for X11R5, 4 for X11R4, and 6 for X11R6).

X11R4 implementations can be tested with a subset of the X11R5 requirements by setting XT_X_RELEASE to a value of 4. This will cause X11R5-specific tests to return results of UNSUPPORTED.

Though VSW5 does not contain tests for X Window System functionality beyond that in X11R5, later implementations (e.g. X11R6) can be tested for conformance to the X11R5 requirements following the same procedures described herein for X11R5. To avoid the tests for XtSpecificationRelease failing on X11R5+ implementations, set XT_X_RELEASE to the value of the release being tested (6 or above).

3.5.3 Configuration for system commands

In this section the names of system commands are specified.

SHELL	The full pathname of the Bourne shell on the implementation. Example: SHELL=/bin/sh
CC	A command to invoke the C compiler. Example: CC=cc *Imake variable: CcCmd*
RM	A command to remove a file without interactive help. Example: RM=rm -f *Imake variable: RmCmd*
AR	A command to generate a library archive. Example: AR=ar crv *Imake variable: ArCmd*
LD	A command to link object files. Example: LD=ld *Imake variable: LdCmd*

LN	A command to make hard links. Example: LN=ln *NB: This does not correspond to the Imake variable: LnCmd*
RANLIB	If the system supports a command to order library archives into random access libraries set the parameter to that command. Otherwise it should be set to `true` (or a command that does nothing). Example: RANLIB=ranlib *Imake variable: RanlibCmd*
TSORT	Set to `cat` if AR was set to a command which inserts a symbol table in the library archive or RANLIB was set to a command which creates a random access library. Otherwise set to `tsort`.
LORDER	Set to `echo` if AR was set to a command which inserts a symbol table in the library archive or RANLIB was set to a command which creates a random access library. Otherwise set to `lorder`.
CP	A command to copy files. Example: CP=cp *Imake variable: CpCmd*
CODEMAKER	A utility to produce C source files from dot-m files. This line must not be altered.

3.5.4 Configuration for TET

This section contains the locations of various parts of TET. Unless TET files have been moved from their default locations only the first parameter need to be changed.

TET_ROOT	The directory that contains the TET files. i.e. where VSW5 was installed. It must be written out as a full path without using any variable notation.
TETBASE	A synonym for TET_ROOT. This value must not be changed.
TETINCDIR	The directory containing the TET headers. Example: TETINCDIR=${TETBASE}/inc/posix_c

TETLIB	The directory containing the TET library. Example: TETLIB=${TETBASE}/lib/posix_c
TCM	The Test Control Manager. This is part of TET. It is an object file that is linked with each test. Example: TCM=${TETLIB}/tcm.o
TCMCHILD	The Test Control Manager. This is part of TET. It is an object file that is linked with each program that is executed within a test by *tet_exec*. Example: TCMCHILD=${TETLIB}/tcmchild.o
APILIB	TET's API library. Example: APILIB=${TETLIB}/libapi.a

3.5.5 Configuration parameters for VSW5

Unless the VSW5 directories have been moved from their standard locations only the first three of these parameters need to be changed.

XTESTHOST	The name of the host on which test suite clients are to be executed. This may be set to the value returned by a command which can be executed using the PATH environment variable or to a specific name. This is used to produce a resource file named `.Xdefaults-$(XTESTHOST)` in the test execution directory. The resource file is created when building the test for *XGetDefault*. This parameter is only used in the Makefile of the test for *XGetDefault*. Example: XTESTHOST=‘hostname‘ Example: XTESTHOST=anchovy
XTESTFONTDIR	The directory in which to install the VSW5 fonts. Example: XTESTFONTDIR=/usr/lib/X11/fonts/vsw5
XTTESTLIB	Not used by VSW5 Lite. The VSW5 X Toolkit Intrinsics test libraries. The XtTest library contains subroutines that are common to many tests in the X Toolkit Intrinsics section of VSW5. This value generally does not need to be changed. Example: XTTESTLIB=${XTESTLIBDIR}/libXtTest.a
XTESTROOT	The directory that is the root of VSW5. Example: XTESTROOT=${TET_ROOT}/vsw5

XTESTLIBDIR	The directory containing libraries for VSW5. This value generally does not need to be changed. Example: XTESTLIBDIR=${XTESTROOT}/lib
XTESTLIB	The VSW5 library. This library contains subroutines that are common to many tests in VSW5. This value generally does not need to be changed. Example: XTESTLIB=${XTESTLIBDIR}/libxtest.a
XSTLIB	The VSW5 X Protocol test library. This library contains subroutines that are common to many tests in the X Protocol section of VSW5. This value generally does not need to be changed. Example: XSTLIB=${XTESTLIBDIR}/libXst.a
XTESTFONTLIB	The VSW5 fonts library. This library contains font descriptions that are common to many tests in VSW5. This value generally does not need to be changed. Example: XTESTFONTLIB=${XTESTLIBDIR}/libfont.a
XTESTINCDIR	The VSW5 header file directory. This directory contains header files that are local to VSW5. This value generally does not need to be changed. Example: XTESTINCDIR=${XTESTROOT}/include
XTESTBIN	The VSW5 binary file directory. This directory contains utility programs that are used by VSW5. This value generally does not need to be changed. Example: XTESTBIN=${XTESTROOT}/bin

3.5.6 System Parameters

Locations of system libraries and include files.

SYSLIBS	Options to allow the C compiler to search for any system libraries that are required for VSW5 that are not searched for by default, e.g. Xlib. Example: SYSLIBS=-lX11 To build VSW5 to make use of the XTEST extension include the XTEST library and the X Window System extension library (in that order). Example: SYSLIBS=-lXtst -lXext -lX11 To build VSW5 to test the Input Device Extension include the libXi library and the X Window System extension

library (in that order). The XTEST library may also be included.
Example: SYSLIBS=-lXi -lXext -lX11

Imake variables: ExtraLibraries

XP_SYSLIBS

Any system libraries that are needed, to link the X Protocol tests. This will include Xlib.
Example: XP_SYSLIBS=-lX11

To build VSW5 to test the Input Device Extension include the libXi library and the X Window System extension library (in that order).

Example: SYSLIBS=-lXi -lXext -lX11

Imake variables: ExtraLibraries

XT_SYSLIBS

Not used by VSW5 Lite. Any system libraries needed to link the X Toolkit Intrinsics tests. This will include Xlib and Xt Do not include the Athena widget libraries (see XT_ATHENA below).
Example: XT_SYSLIBS=-lXt -lX11

To build VSW5 to make use of the XTEST extension include the XTEST library and the X Window System extension library (in that order).
Example: SYSLIBS=-lXtst -lXext -lXt -lX11

To build VSW5 to test the Shape Extension include the X Window System extension library.
Example: SYSLIBS= -lXext -lXt -lX11

XT_ATHENA

Not used by VSW5 Lite. Any system libraries needed to link the X Toolkit Intrinsics tests with the Athena widgets.

If Athena widgets are provided by the implementation this is generally libXmu and libXaw.
Example: XT_ATHENA=-lXaw -lXmu

If Athena widgets are not provided by the implementation the Xaw and Xmu libraries provided by the test suite will need to be specified (see section 3.8.1 on page 26).
Example: XT_ATHENA=${XTESTLIBDIR}/libXtaw.a ${XTESTLIBDIR}/libXtmu.a

SYSINC	Any commands that should be given to the C compiler to cause all relevant system include files to be included. This will generally include both /usr/include/ and /usr/include/X11. Example: SYSINC= -I/usr/include/ -I/usr/include/X11 Note that if /usr/include is not specified explicitly Athena widget header files provided by VSW5 will be used rather than those provided by the system. Thus /usr/include, or the equivalent root of the include tree, should always be explicitly specified.

3.5.7 C Compiler Directives

Directives to the C compiler. Usually only the first few parameters will need changing. The remainder are just combinations of other parameters.

COPTS	Options to the C compiler. Example: COPTS=-O *Imake variables: DefaultCDebugFlags and DefaultCCOptions*
DEFINES	Options required by the C compiler to set up any required defines. For example on strict ANSI Standard-C systems it is generally necessary to define _POSIX_SOURCE while on Open Group conformant systems it is generally necessary to define _XOPEN_SOURCE. Example: DEFINES=-D_XOPEN_SOURCE To build VSW5 to make use of the XTEST extension, define XTESTEXTENSION. XTESTEXTENSION is only used when building VSW5 library. Example: DEFINES= -DXTESTEXTENSION -D_XOPEN_SOURCE To build VSW5 to test the Input Device Extension, define INPUTEXTENSION. Example: DEFINES= -DINPUTEXTENSION -DXTESTEXTENSION -D_XOPEN_SOURCE *Imake variables: StandardDefines*
XP_DEFINES	C compiler defines needed for the X Protocol tests. This can

be set as DEFINES but support for additional connection methods beyond TCP/IP can be built using the following defines if XP_OPEN_DIS is XlibNoXtst.c (R4/R5 XOpenDisplay emulation):

-DDNETCONN -	Connections can also use DECnet.
-DUNIXCONN -	Connections can also use UNIX domain sockets.

Refer to the implementation's documentation for building and installing Xlib on the platform to determine if either of these additional defines are necessary. If XP_OPEN_DIS is one of `XlibXtst.c` or `XlibOpaque.c` neither of the defines listed above will be required.

Example: XP_DEFINES=-D_XOPEN_SOURCE -DUNIXCONN

To build VSW5 to test the Input Device Extension, define INPUTEXTENSION.

Example: XP_DEFINES=-DINPUTEXTENSION -D_XOPEN_SOURCE -DUNIXCONN
Imake variables: StandardDefines

XT_DEFINES

Not used by VSW5 Lite. C compiler defines needed for the X Toolkit Intrinsics tests. On must implementations -DNeedFunctionPrototypes needs to be included to avoid compiler warnings about the definition of XtPointer.

To build VSW5 to make use of the XTEST extension in Xt test, define XTESTEXTENSION.

Example: XT_DEFINES=-D_XOPEN_SOURCE -DNeedFunctionPrototypes -DXTESTEXTENSION

LINKOBJOPTS

Options to give to the LD program to link object files together into one object file that can be further linked.
Example: LINKOBJOPTS=-r

LDFLAGS

Flags used by the loader. This is needed on some systems to specify options used when object files are linked to produce an executable.
Example: LDFLAGS=-ZP

XP_OPEN_DIS

A choice of which code to build in the X Protocol library to make an X server connection. This must be set to one of three possible values:

`XlibXtst.c`

Use this option only if Xlib includes post R5 enhancements to *_XConnectDisplay* ensuring maximum portable protocol test coverage. These enhancements include arguments to *_XConnectDisplay* to return authorization details on connection. Using this option when Xlib does not have these enhancements to *_XConnectDisplay* will yield undefined VSW5 results.

`XlibOpaque.c`

The implementation has a normal R4 Xlib or early R5 Xlib which cannot be patched to include the enhancements to _XConnectDisplay, and these cannot be emulated by building `XlibNoXtst.c`, so only client-native testing can be done portably, and no failure testing of *XOpenDisplay* can be done. This option uses *XOpenDisplay* to make the connection, from which the file descriptor is recovered for our own use. *XCloseDisplay* shuts down the connection.

`XlibNoXtst.c`

As for `XlibOpaque.c` but the implementation can use the R4/R5 connection emulation supplied. (Note: R4/R5 independent) This will ensure maximum protocol test coverage but may not be portable to all platforms.
Reasons for not being able to build `XlibNoXtst.c` might include:

i) different interfaces to connection setup and connection read/write;
ii) different access control mechanisms.

Refer to the implementation’s Xlib documentation for further details.

Example: XP_OPEN_DIS=XlibOpaque.c

INCLUDES Options to cause C compiler to search the correct directories for headers. This should not need changing as it is just a combination of other parameters.

CFLAGS Flags for the C compiler. This should not need changing as

	it is just a combination of other parameters. Note that CFLOCAL is not defined in the configuration file; it is available for use in makefiles to define parameters that only apply to a particular case. (It intentionally uses parentheses rather than braces)
XP_CFLAGS	Flags for the C compiler. This parameter is used by the X Protocol tests in VSW5. This should not need changing as it is just a combination of other parameters.
XT_CFLAGS	Not used by VSW5 Lite. Flags for the C compiler. This parameter is used by the X Toolkit Intrinsics tests in VSW5. This should not need changing as it is just a combination of other parameters.
LIBS	List of libraries. This should not need changing as it is just a combination of other parameters.
XP_LIBS	List of libraries. This parameter is used by the X Protocol tests in VSW5. This should not need changing as it is just a combination of other parameters.
XT_LIBS	Not used by VSW5 Lite. List of libraries. This parameter is used by the X Toolkit Intrinsics tests in VSW5. This should not need changing as it is just a combination of other parameters.

3.5.8 Pixel validation section

This section defines a number of parameters that are used only when generating known good image files. These are not intended to be modified and need not be used when running the test suite. They are only used in the VSW5 development environment when generating known good image files.

3.6 SETTING CLEAN CONFIGURATION PARAMETERS

When cleaning up after running a test TET employs a clean tool analogous to the build tool discussed above. The clean tool for VSW5 is a utility named `wclean` which is supplied in the directory `$TET_ROOT/vsw5/bin`. The `wclean` utility is an interface to the `make` system command which cleans-up after a test using the rules provided in a Makefile. The `wclean` utility may be invoked directly from the shell, as well as via TET, to cleanup individual parts of VSW5.

Each Makefile in VSW5 is written using symbolic names to describe commands and parameters which may vary from system to system. The values of these symbolic names are all obtained by `wclean` from the configuration parameters in the TET clean configuration file for VSW5:

$TET_ROOT/vsw5/tetclean.cfg. tetclean.cfg is actually identical to tetbuild.cfg. After tetbuild.cfg is configured for the implementation, copy it into the clean configuration file:

```
cd $TET_ROOT/vsw5
cp tetbuild.cfg tetclean.cfg
```

3.7 SYSTEM DEPENDENT SOURCE FILES

This section describes files provided in VSW5 which may need to be edited to reflect the system under test.

3.7.1 Host address structures

The file xthost.c in the directory $TET_ROOT/vsw5/src/lib contains three items which may need to be edited to reflect the system under test. These are all related to the mechanisms provided by the X server under test to add, get, and remove hosts from the access control list. These are only used in the tests for those Xlib functions which use or modify the access control list.

The host access control functions use the XHostAddress structure. Refer to the Xlib documentation for the implementation under test to determine the allowed formats for host addresses in an XHostAddress structure. It may be helpful to refer to the X Window System Protocol documentation supplied with the X server under test. The section describing the ChangeHosts protocol request gives examples of host address formats supported by many X servers. The symbols FamilyInternet, FamilyDECnet, FamilyChaos and FamilyUname are defined on many systems in the include files X.h and Xstreams.h. The X server under test is not guaranteed to support these families, and may support families not listed here. Determine which families are supported for the X server under test, by examining the header files supplied with the system, and consulting the documentation supplied with the X server.
Some default declarations are contained in the file but there is no guarantee that they will work correctly on a given system.

To create a version of $TET_ROOT/vsw5/src/lib/xthost.c which will operate with the implementation under test:

- Ensure that there is a declaration for an array xthosts[] of at least 5 XHostAddress structures containing valid family, length, address triples.

- Ensure that there is a declaration for an array xtbadhosts[] of at least 5 XHostAddress structures containing invalid family, length, address triples. If the supplied examples cannot be used, the simplest way to do this is to use an invalid family, which is not supported by the X server under test, in each structure of the array.

- Ensure that there is a declaration for a function *samehost* that compares two XHostAddress structures and returns True if they are equivalent. (It is unlikely that the sample function will need modification - no systems requiring modification have yet been identified).

3.8 INSTALLING THE VSW5 LIBRARIES AND TOOLS

Before proceeding with the installation process, ensure:

1) The PATH environment variable contains `$TET_ROOT/bin`, `$TET_ROOT/vsw5/bin` and the current working directory ".".

2) The user who will be performing the remainder of the installation process has read, write and search access to the VSW5 directories AND IS THE OWNER OF THE VSW5 FILES.

3) The utilities needed by VSW5 are accessible through the PATH environment variable.

These conditions are required for the remaining steps in the installation process.

Once the VSW5 distribution has been loaded and the steps defined earlier in this chapter have been completed the libraries and tools used by VSW5 must be built. Note some libraries and tools are not rebuilt automatically when tests are built. If the implementation header files are modified after VSW5 is installed this step of the installation process should be performed again to generate versions of the libraries and tools which reflect these changes.

To build the libraries and tools change the current working directory to `$TET_ROOT/vsw5` and execute the command:

```
make
```

This command causes the VSW5 libraries and tools to be built and installed in the proper locations in the VSW5 directory tree and sets test suite directory permissions to 775. The progress of this process is indicated by messages printed at each major step.

The command

```
make build
```

performs the same tasks but without setting test suite directory permissions. This is useful when rebuilding the VSW5 libraries and tools after initial installation.

Messages produced during the installation process are recorded in the file `$TET_ROOT/vsw5/results/install.log`. After the installation process is complete the contents of this file should be examined for reported problems and warnings.

The code in the VSW5 libraries and tools is intended to be highly portable. Warnings may occur if the implementation's header files are not yet conformant. If warnings occur with conforming header files they should be reported as potential VSW5 problems.

Any fatal error during the build of the VSW5 libraries and tools is a serious problem that will prevent execution of at least a portion of VSW5. For recommendations on troubleshooting such problems see section 7.5 on page 58.

3.8.1 VSW-Provided libXaw and libXmu

Note: this section does not apply to VSW5 Lite.

The X Toolkit Intrinsics tests utilize the Athena widget set. These widgets are include in the X Window System sample implementation and will generally be available on the implementation under test. For implementations which do not provide the Athena widgets VSW5 includes versions of the appropriate libraries and headers. This software is a subset of the complete Athena widget set including just those features needed by VSW5 and is unlikely to be suitable for purposes other than use in VSW5.

As it is preferable to use the Athena widgets provided by the implementation, if available, the VSW5-provided Athena widgets are not built during the standard VSW5 libraries and tools installation process. To build the VSW5 Athena Widgets change the current working directory to `$TET_ROOT/vsw5` and execute the command:

```
make athena
```

This causes the VSW5 Athena widget implementation to be built and installed in a location within the VSW5 directory tree.

3.9 COMPILING AND INSTALLING THE TEST FONTS

VSW5 contains a series of test fonts which are used to test the correctness of the information returned by the graphics functions in the X Window System. This is done by comparing the information returned by those functions with expected font characteristics which are compiled into the tests.

There are nine test fonts whose descriptions are contained in the files

```
xtfont0.bdf     xtfont1.bdf      xtfont2.bdf
xtfont3.bdf     xtfont4.bdf      xtfont5.bdf
xtfont6.bdf     xtfont7.bdf      xtfont8.bdf
```

These files are located in the directory `$TET_ROOT/vsw5/fonts`.

The manner in which fonts should be compiled and installed for any particular X server is system dependent. Some sample instructions are given here which may be useful on many systems. These may not be appropriate for a given system, or they may need adaptation to work properly and are provided only as a guide.

Move to the directory `$TET_ROOT/vsw5/fonts.`

```
cd $TET_ROOT/vsw5/fonts
```

Compile the seven bdf files into snf, pcf, or fb format, as appropriate for the implementation.

```
wbuild comp_snf
```

or

```
wbuild comp_pcf
```

or

```
wbuild comp_dxpcf
```

or

```
wbuild comp_fb
```

Copy the compiled fonts into the server font directory (the XTESTFONTDIR configuration parameter).

```
wbuild install_snf
```

or

```
wbuild install_pcf
```

or

```
wbuild install_dxpcf
```

or

```
wbuild install_fb
```

X WINDOW SYSTEM VERIFICATION SUITE

INSTALLATION CHECKLIST

Steps should be completed in the order given. Numbers in parentheses below are references to sections of this Chapter.

- ❐ Install and build TET on the system if it is not already in place (3.1.1)

- ❐ Verify the system provides the resources needed to install VSW5
  - ❐ Disk Space (3.1.2)
  - ❐ Commands and Utilities (3.1.3)

- ❐ Verify the system provides the resource needed to run VSW5*
  - ❐ Semaphores (not required if using VSW5 Lite) (3.1.4)

- ❐ Load the VSW5 Files (3.2)
  - ❐ Change the current working directory to the $TET_ROOT directory
  - ❐ Copy the VSW5 distribution to the current working directory
  - ❐ Uncompress the distribution
  - ❐ Un-tar the distribution

- ❐ Configure the VSW5 environment
  - ❐ Suppress extraneous compiler messages (3.3)
  - ❐ Modify `tetbuild.cfg` (3.5)
  - ❐ Copy `tetbuild.cfg` to `tetclean.cfg` (3.6)
  - ❐ Make modifications to system dependent files if needed (3.7)

- ❐ Install the VSW5 Libraries and Tools (3.8)
  - ❐ Verify the PATH environment variable contains `$TET_ROOT/bin` and `$TET_ROOT/vsw5/bin`
  - ❐ Verify the user has read, write and search access to the VSW5 directories and is the owner of the VSW5 files
  - ❐ Change the current working directory to the $TET_ROOT/vsw5 directory
  - ❐ Execute the command `make`
  - ❐ Examine the file `$TET_ROOT/vsw5/results/install.log` for reports of errors during the installation
  - ❐ Build the VSW5 version of the Athena widgets libraries if these are not provided by the implementation under test (does not apply to VSW5 Lite) (3.8.1)

- ❐ Install the VSW5 Test Fonts (3.9)

* These resources do not need to be available to install VSW5 but are needed to configure and execute it.

4.0 CONFIGURING VSW5

This chapter provides instructions for configuring VSW5 for use on an implementation. A configuration checklist is provided at the end of the chapter. This checklist is an aid to ensuring the procedures in the body of the chapter are successfully completed.

Configuration of VSW5 entails:

- modifying the `tetexec.cfg` configuration file.

4.1 SETTING EXECUTION CONFIGURATION PARAMETERS

The file `$TET_ROOT/vsw5/tetexec.cfg` contains the configuration variables used in executingVSW5. Each variable name specific to VSW5 is prefixed with “XT_”. As distributed this file contains sample values. These are not default values. The XT_ variables must be modified to match the behavior of the implementation in order to ensure correct VSW5 operation. The requirements for setting each variable are documented in the `tetexec.cfg` file itself and covered in the remainder of this section.

Note: Numeric values may be specified in the `tetexec.cfg` file in decimal, octal, or hexadecimal. Octal values must be a sequence of octal digits preceded by 0. Hexadecimal values must be a sequence of hexadecimal digits preceded by 0x or 0X.

4.1.1 Configuration parameters defined by TET

TET_EXEC_IN_PLACE	Setting this variable to False indicates that files will be executed in a temporary execution directory. Use of a temporary execution directory for each test enables parallel execution of the test suite against multiple servers. Setting this variable to True provides improved performance if not attempting parallel execution of the test suite against multiple servers. Example: TET_EXEC_IN_PLACE=False
TET_SAVE_FILES	Which files generated during execution of tests are to be saved for later examination. This line must not be altered.

4.1.2 Configuration parameters for VSW5

The following parameters are used in many places in VSW5. These should be set to match the X server to be tested and the underlying operating system on which the X Window System is implemented.

XT_COVERAGE	There are three different levels of coverage that can be tested. They are largely identical but effect a few tests which

are in a transitional state in terms of the consistency of the specification, test suite, and sample code. The value of this variable determines whether these tests are run, or return results of UNTESTED:

0 All tests are run. This level is intended for use by developers of the sample code and the test suite as it enables execution of tests which are under investigation.

1 All tests which are expected to pass are run. At present this is as of R6.4gamma of the sample code. This value must be used with implementations derived from the sample code at this revision or above. Otherwise see value 2. This value causes a value of UNTESTED for the following tests:

Xt11/XtSetValues	13
Xt11/XtVaSetValues	13
Xt9/XtAppPeekEvent	3
XtC/XtPeekEvent	2

2 This value may be used with implementations which are derived from revisions of the sample code earlier than R6.4gamma. It adds the following to the set of tests which give UNTESTED results:

Xt8/XtMakeResizeRequest	9
Xt9/XtGrabButton	3
Xt11/XtCvtStringToAcceleratorTable	3
Xt11/XtCvtStringToFloat	3
Xt11/XtCvtStringToFloat	4
Xt11/XtCvtStringToTranslationTable	4

This value is intended to reduce failures due to issues with alignment of the sample code, test suite, and specification for releases prior to R6.4gamma. Please note however that failures of this sort remain possible for implementations derived from versions of the sample code prior to R6.4gamma. In such cases if it not possible to upgrade to a newer version of the sample code then Interpretation Requests for PINs or TINs should be filed for each such failure (the VSW5 IR database should be checked first, as PINs or TINs covering these failures may already have been granted). Please do not submit IRs or SRs requesting the list of tests returning UNTESTED for these coverage values be expanded. It is expected that once consistent versions of the specification, test suite, and sample code are in wide use these coverage variables will be removed from VSW5, and thus requests to expand them in the interim will not be approved.

XT_DISPLAY

A display string that can be passed to *XOpenDisplay* to access the display under test. It must include a screen; all testing is done for a particular screen.
Example: XT_DISPLAY=:0.0

XT_ALT_SCREEN

If the display supports more than one screen, the number of a screen that is different from that incorporated in the XT_DISPLAY variable. Set to the string UNSUPPORTED if only one screen is available. Note that this should be a screen number, not a display string that can be passed to *XOpenDisplay.*
Example: XT_ALT_SCREEN=1

XT_FONTPATH

A comma separated list that is a valid font path for the X server. It must include at least the components of the default font path for the X server, enabling the cursor font to be accessed. One of the components must be the directory in which the test fonts were installed (see see section 3.8 on page 25).

This parameter will be used to set the font path for specific test purposes which access the test fonts. The font path is restored on completion of the specific test purposes.
Example: XT_FONTPATH=/usr/lib/X11/fonts/vsw5/,/usr/lib/X11/fonts/misc/

XT_SPEEDFACTOR

A speedfactor indicating the relative delay in response of the underlying operating system and X server combined.

Co-operating processes within VSW5 allow a time delay in proportion to this speedfactor to account for scheduling delays in the underlying operating system and X server. Set to a number greater than or equal to one. There should be no need to change the default unless the round trip time to the X server can be very long (>15 seconds); in this case set this parameter to a value larger than the maximum round trip time divided by 3.
Example: XT_SPEEDFACTOR=5

XT_RESET_DELAY	A delay time in seconds. Set to a time which is greater than or equal to the maximum time required by the X server to reset when the last client is closed. The test suite pauses for this time whenever a connection is about to be opened since the server may be resetting as a result of closing the last connection in the previous test case. Example: XT_RESET_DELAY=1
XT_EXTENSIONS	Whether to test the extended assertions which require the XTEST extension. Set to Yes if the XTEST extension is available on the implementation (see section 3.4 on page 13) and VSW5 has been configured to build in the XTEST extension libraries (see section 3.5.7 on page 20), otherwise set to No. Example: XT_EXTENSIONS=No

4.1.3 Configuration parameters for specific tests

The following parameters are used to control one or more specific test purposes in VSW5. They must be set to appropriate values for the X server to be tested.

XT_VISUAL_CLASSES	A space separated list of the visual classes that are supported for the screen given by XT_DISPLAY. Each visual class is followed by a list of depths at which the class is supported (enclosed by brackets and separated by commas with no spaces). Visual classes and depths that are supported only by other screens must not be included. Note that this parameter is only used to check the correctness of the values returned by *XMatchVisualInfo* and *XGetVisualInfo*. Other tests which loop over visuals obtain the values by calling these functions. Example: XT_VISUAL_CLASSES=StaticGray(8) GrayScale(8) StaticColor(8) PseudoColor(8) TrueColor(8) DirectColor(8)
XT_FONTCURSOR_GOOD	The number of a glyph in the default cursor font known to exist. XT_FONTCURSOR_GOOD+2 must also be a glyph in the default cursor font. Neither of these may be the same as the X server’s default cursor. Example: XT_FONTCURSOR_GOOD=2
XT_FONTCURSOR_BAD	The number of a glyph in the default cursor font known not to exist. If no such value exists set to UNSUPPORTED. Example: XT_FONTCURSOR_BAD=9999

XT_FONTPATH_GOOD	A comma separated list that is a valid font path for the X server. It must be different from XT_FONTPATH. It need not contain the test fonts. Example: XT_FONTPATH_GOOD=/usr/lib/X11/fonts/100dpi/,/usr/lib/X11/fonts/75dpi/
XT_FONTPATH_BAD	A comma separated list that is an invalid font path for the X server. If a suitable value cannot be determined set to UNSUPPORTED. There is no default value - by default, tests which use this parameter will be reported as UNSUPPORTED. Example: XT_FONTPATH_BAD=/jfkdsjfksl
XT_BAD_FONT_NAME	A non-existent font name. Example: XT_BAD_FONT_NAME=non-existent-font-name
XT_GOOD_COLORNAME	The name of a color which exists in the color database for the X server. Example: XT_GOOD_COLORNAME=red
XT_BAD_COLORNAME	The name of a color which does not exist in the color database for the X server. Example: XT_BAD_COLORNAME=nosuchcolour
XT_DISPLAYMOTIONBUFFERSIZE	A non-zero value (the value returned by *XDisplayMotionBufferSize*) if the X server supports a more complete history of pointer motion than that provided by event notification, or zero otherwise. The more complete history is made available via the Xlib functions *XDisplayMotionBufferSize* and *XGetMotionEvents*. Example: XT_DISPLAYMOTIONBUFFERSIZE=256

4.1.4 Configuration parameters for display function tests

The following parameters are used to control one or more test purposes for Xlib Display functions. They should be set to match the display specified in the XT_DISPLAY parameter.

Some of these parameters are specific to the particular screen of the display under test. This is also specified in the XT_DISPLAY parameter.

Suitable values for most of these parameters can be obtained from the output of the X11 utility `xdpyinfo`.

XT_SCREEN_COUNT	The number of screens available on the display as returned by *XScreenCount*. Example: XT_SCREEN_COUNT=2
XT_PIXMAP_DEPTHS	A space separated list of depths supported by the specified screen of the display that can be used for pixmaps. Example: XT_PIXMAP_DEPTHS=1 8
XT_BLACK_PIXEL	The black pixel value of the specified screen of the display. Example: XT_BLACK_PIXEL=1
XT_WHITE_PIXEL	The white pixel value of the specified screen of the display. Example: XT_WHITE_PIXEL=0
XT_HEIGHT_MM	The height in millimeters of the specified screen of the display. Example: XT_HEIGHT_MM=254
XT_WIDTH_MM	The width in millimeters of the specified screen of the display. Example: XT_WIDTH_MM=325
XT_PROTOCOL_VERSION	The major version number (11) of the X protocol as returned by XProtocolVersion. Example: XT_PROTOCOL_VERSION=11
XT_PROTOCOL_REVISION	The minor protocol revision number as returned by XProtocolRevision. Example: XT_PROTOCOL_REVISION=0
XT_SERVER_VENDOR	The X server vendor string as returned by XServerVendor. Example: XT_SERVER_VENDOR=MIT X Consortium
XT_VENDOR_RELEASE	The X server vendor’s release number as returned by *XVendorRelease*. Example: XT_VENDOR_RELEASE=5000
XT_DOES_SAVE_UNDERS	Set to Yes if the specified screen of the display supports save unders (indicated by *XDoesSaveUnders* returning True) otherwise set to No. Example: XT_DOES_SAVE_UNDERS=Yes
XT_DOES_BACKING_STORE	Set to one of the following: 0 - the specified screen supports backing store NotUseful 1 - the specified screen supports backing store WhenMapped 2 - the specified screen supports backing store Always

	The way the specified screen supports backing store is indicated by the return value of *XDoesBackingStore*. Example: XT_DOES_BACKING_STORE=2

4.1.5 Configuration parameters for connection function tests

The following parameters are used to control one or more test purposes for *XOpenDisplay*, *XCloseDisplay* and *XConnectionNumber.*

They must be set to match the display specified in the XT_DISPLAY parameter and the characteristics of the underlying operating system.

XT_POSIX_SYSTEM	Set to Yes to indicate that the underlying operating system is a POSIX system. If this parameter is set to Yes, some extended assertions which describe implementation dependent functionality will be tested assuming POSIX concepts. Example: XT_POSIX_SYSTEM=Yes
XT_DECNET	Set this to Yes if clients can connect to the X server under test using DECnet. This will be used (on a POSIX system) in the tests for *XOpenDisplay.* Example: XT_DECNET=No
XT_TCP	Set to Yes if clients can connect to the X server under test using TCP streams. This will be used (on a POSIX system) in the tests for *XOpenDisplay.* Example: XT_TCP=Yes
XT_DISPLAYHOST	The hostname of the machine on which the display is physically attached. This will be used instead of XT_DISPLAY (on a POSIX system) in the tests for *XOpenDisplay* which specifically test the hostname component of the display name. Note that this may not be the same as the machine on which the test suite clients execute (XTESTHOST). Example: XT_DISPLAYHOST=xdisplay.lcs.mit.edu
XT_LOCAL	Set to Yes if clients can connect to a local X server without passing a hostname to *XOpenDisplay.* This will be used (on a POSIX system) in the tests for *XOpenDisplay.* This is usually the case when the X server under test is running on the same platform as VSW5. When a hostname is omitted, the Xlib implementation of *XOpenDisplay* can use the fastest available transport mechanism to make local

	connections. Example: XT_LOCAL=No

4.1.6 Configuration parameters which do not affect test results

There are a number of execution configuration parameters which can be used to reduce the size of the journal file, or dump out more information from the test suite. They will not alter the behavior of the tests or the test results.

XT_SAVE_SERVER_IMAGE	When set to Yes the image produced by the server that is compared with the known good image is dumped to a file with suffix “.sav”. Example: XT_SAVE_SERVER_IMAGE=Yes
XT_OPTION_NO_CHECK	This may be set to Yes to suppress the generation of journal file messages containing CHECK keywords. These messages are used to record passing checkpoints in a test. They contain the checkpoint number within the test and the line number within the source code. Surpressing these messages can reduce the size of the journal file considerably. Example: XT_OPTION_NO_CHECK=Yes
XT_OPTION_NO_TRACE	This may be set to Yes to suppress the generation of journal file messages containing TRACE keywords. These messages are used to provide information about the interaction between a test and the implementation. Surpressing these messages can reduce the size of the journal file considerably. Example: XT_OPTION_NO_TRACE=Yes

4.1.7 Configuration parameters for debugging tests

There are a number of execution configuration parameters which should not be set when performing verification test runs. These are intended for debugging purposes. These parameters may affect the behavior of some test purposes if they are set to assist debugging.

XT_DEBUG	This may be set to a debugging level. A higher level produces more debugging output. Output is only produced by the test suite at levels 1, 2 and 3. Setting this variable to 0 produces no debug output, and 3 gives everything possible (setting this variable to 3 can give an enormous volume of output so it should not be used when running large numbers of test sets). Example: XT_DEBUG=0

XT_DEBUG_OVERRIDE_REDIRECT
When set to Yes windows are created with override_redirect set. This enables tests to be run more easily with a window manager running on the same screen This may not be set to Yes for formal verification testing.
Example: XT_DEBUG_OVERRIDE_REDIRECT=No

XT_DEBUG_PAUSE_AFTER When set to Yes the test pauses after each call to the Xlib function being tested, until Carriage Return is entered. This is useful to enable the results of graphics operations to be observed. This may not be set to Yes for formal verification testing.
Example: XT_DEBUG_PAUSE_AFTER=No

XT_DEBUG_PIXMAP_ONLY When set to Yes tests which would normally loop over both windows and pixmaps are restricted to loop over just pixmaps. This is useful for speeding up the execution of the test set. This may not be set to Yes for formal verification testing.

If XT_DEBUG_WINDOW_ONLY is also set to Yes, some tests will report UNRESOLVED due to the fact that nothing has been tested.
Example: XT_DEBUG_PIXMAP_ONLY=No

XT_DEBUG_WINDOW_ONLY
When set to Yes tests which would normally loop over both windows and pixmaps are restricted to loop over just windows. This is useful for speeding up the execution of the test suite. This should not be set to Yes for formal verification testing.

If XT_DEBUG_PIXMAP_ONLY is also set to Yes some tests will report UNRESOLVED due to the fact that nothing has been tested.
Example: XT_DEBUG_WINDOW_ONLY=No

XT_DEBUG_DEFAULT_DEPTHS
When set to Yes tests which would normally loop over multiple depths are restricted to test just the first visual returned by XGetVisualInfo and/or the first pixmap depth returned by *XListDepths* (depending on whether XT_DEBUG_PIXMAP_ONLY or XT_DEBUG_WINDOW_ONLY is also set). This is useful for speeding up the execution of the test suite. This may not be set to Yes for formal verification testing.

Note that the first visual returned by XGetVisualInfo may not be the default visual for the screen.
Example: XT_DEBUG_DEFAULT_DEPTHS=No

XT_DEBUG_VISUAL_IDS

When set to a non-empty string, tests which would normally loop over multiple depths are restricted to test just the visuals IDs listed. Note that visual IDs for visuals on more than one screen may be entered, but those used will depend on whether the test being executed uses visuals on the default screen or alternate screen. The visuals IDs should be entered in decimal, octal or hexadecimal and separated with commas and with no intervening spaces. This may not be set to a non-empty string for formal verification testing.
Example: XT_DEBUG_VISUAL_IDS=0x22,0x24,0x27

XT_DEBUG_NO_PIXCHECK

When set to Yes tests which would normally perform pixmap verification omit this (all other processing is performed in those tests as normal). Pixmap verification is a scheme which compares the image produced by the X server with a known good image file which is part of VSW5 (see section 7.1 on page 55). This may not be set to Yes for formal verification testing.
Example: XT_DEBUG_NO_PIXCHECK=No

XT_DEBUG_BYTE_SEX

When set to NATIVE, REVERSE, MSB, or LSB the X Protocol tests will only be executed with the specified byte sex. When set to BOTH the X Protocol tests make connections to the X server using both the native and reversed byte sex.

Note: The parameter must be set to NATIVE when the build configuration parameter XP_OPEN_DIS is set to XlibOpaque.c
Example: XT_DEBUG_BYTE_SEX=NATIVE

XT_DEBUG_VISUAL_CHECK

When set to a non-zero value the X Protocol tests will pause for the specified time interval (in seconds) to enable a visual check to be performed on the displayed screen contents.
Example: XT_DEBUG_VISUAL_CHECK=5

4.1.8 Invariant configuration parameters

These parameters are defined for use during test development and should not be modified.

XT_LOCALE	The locale in which input method tests are run.
XT_FONTSET	The base font name list used to select fonts when font sets are generated.
XT_LOCALE_MODIFIERS	Used to verify that XSetLocaleModifiers works properly.
XT_SAVE_IM	Used for developing and debugging input method tests.

4.1.9 Configuration parameters used only during test development

This section defines parameters that are used only when generating known good image files. These are not intended to be modified and need not be used when running the test suite. They are only used in the development environment when generating known good image files.

XT_FONTDIR	The directory in which the VSW5 fonts are located (before being installed). This must be set such that appending a string gives a valid file name. This is normally set to `$TET_ROOT/vsw5/fonts/`. Example: XT_FONTDIR=/usr/mit/testsuite/vsw5/fonts/

VSW5 TEST SUITE

CONFIGURATION CHECKLIST

Steps should be completed in the order given. Numbers in parentheses below are references to sections of this Chapter.

❒ Modify each of the values in the `tetexec.cfg` file to match the implementation to be tested (4.1)

5.0 RUNNING VSW5

Before running VSW5 ensure:

1) The PATH environment variable contains `$TET_ROOT/bin`, `$TET_ROOT/vsw5/bin` and the current working directory “.”.

2) Read, write and, search access is available to the VSW5 directories.

3) The user running VSW5 is not specially privileged (i.e. is not root).

4) The utilities needed by VSW5 are accessible through the PATH environment variable (see section 3.1.3 on page 11).

5.1 X Server Configuration

A number of the tests within VSW5 can only give reliable results if there is no window manager and no other clients making connections to the X server. Thus there should be no window manager and no other clients connected to the X server when executing VSW5.

It is recommended to close down and restart the X server to be tested before each test run in order to ensure that results produced are repeatable and are not affected by earlier tests, although this is not strictly necessary.

The screen saver should be switched off if possible when running VSW5. This is because some X servers implement the screen saver in a way which interferes with windows created by test suite clients, which may cause misleading results. If the screen saver cannot be switched off, the time interval should be set so large as to prevent interference with the tests.

Access control must be disabled for the server under test so the test suite can make connections to the server. Also (if the X server allows this) ensure that clients on the host system (as specified in the build configuration parameter XTESTHOST can modify the access control list. Some X servers support the -ac option which disables host-based access control mechanisms. If this option is supported it should be used when starting the X server for VSW5 testing.

5.1.1 Informal testing and debugging

Although no guarantee can be made that the tests within VSW5 will give correct results if there are window managers and other clients connected to the X server, it is still possible to run many tests satisfactorily.

This section gives some guidelines which may be helpful in running tests with a window manager present, and still deriving correct results. The guidelines have been derived from the experience gained during the development of the tests.

1) Set XT_DEBUG_OVERRIDE_REDIRECT=Yes in the execution configuration file. This is described in more detail in the next section.

2) Do not raise any windows on top of those created by running tests.

3) Avoid having any windows at position (0,0). Note that some window managers such as tvtwm create their own “root” window at position (0,0). This mainly affects tests for the Xlib10 section.

4) Be prepared to lose the input focus when tests are running and don’t forcibly restore it. This mainly affects tests for the Xlib10 section.

5.2 SCENARIOS

The scenario file `$TET_ROOT/vsw5/tet_scen` contains the TET scenarios for VSW5.

The *all*[1] scenario executes all of the VSW5 tests for requirements defined in the Open Group Window Management (X11R5) document set. This scenario should be used when registeringan imlpementation as conforming to both the X Window System Application Interface and X Window System Display Product Standards. It cannot be used with VSW5 Lite as some of the tests executed are not included in the Lite subset of VSW5.

The *Interface*[1] scenario executes just the VSW5 tests applicable to the X Window System Application Interface Product Standard (including V2) and should be used when registering an implementation as conforming to just this Product Standard. It cannot be used with VSW5 Lite as some of the tests executed are not included in the Lite subset of VSW5.

The *Display* scenario executes just the VSW5 tests applicable to the X Window System Display Product Standard and should be used when registering an implementation as conforming to just this Product Standard (e.g. when using VSW5 Lite).

The following additional scenarios are defined for use in development and debugging:

- Xlib: executes the VSW5 tests for Xlib requirements
- Xt[1]: executes the VSW5 tests for X Toolkit Intrinsics requirements

- VSW5 Sections:

  - Xlib*nn*[2] tests for a portion of Xlib (e.g. Xlib3, Xlib15).

1. These scenarios are not usable with VSW5 Lite.

• Xopen	tests for the Open Group specific requirements for Xlib.
• Xproto	tests for the X protocol
• Xt*nn*[1,2]	tests for a portion of the X Toolkit Intrinsics (e.g. Xt3, Xt15).

- Extensions tests - these are not required for formal branding and thus are not included in the *all* and *Display* scenarios:

    - InputDeviceExt: Executes all the VSW5 tests for the Input Device Extension
        - XI: Executes the VSW5 tests for Input Device Extension APIs
        - XIproto: Executes the VSW5 test for the Input Device Extension protocol
    - ShapeExt: executes all the VSW5 tests for the Shape Extension[3]

- Each of the VSW5 Test Cases. The names of these scenarios are the full names of the functions and headers files involved, e.g. XOpenDisplay. One exception is the scenario for XtDisplayStringConversionWarning which needed a shorter name due to the TETware limit of 31 characters on scenario names and is thus called XtDisplayStringConversionWarn.

    Scenario names for X Server tests are prefixed with a p, e.g. pAllocColor.

    Scenario names for Open Group-specific Xlib tests are prefixed with an x, e.g. xXStringToKeysym.

5.3 BUILD, CLEAN AND EXECUTE

The standard TET commands are used to clean, build, and execute VSW5 scenarios. For example:

- To clean all of VSW5 the command is: `tcc -c vsw5 all`
- To build all of VSW5 the command is: `tcc -b vsw5 all`
- To execute all of VSW5 the command is: `tcc -e vsw5 all`

Any of the other VSW5 scenarios can be used in place of ‘all’ in the above examples.

The VSW5 report generators (see section 6.0 on page 45) generate meaningful reports only for journal files generated with a tcc command containing the -e option. Journal files containing the output of running the tcc command with only the -b and/or -c options should not be used with the report generators.

For this reason, and because a build of VSW5 is necessary to generate FIP results, VSW5 scenarios should always be run with the options -be. In order to ensure that all files are rebuilt this should be preceded by running the same scenario with the -c option. Using the -c option to clean up files as the scenario is run is also recommended since the amount of disk space used by a full

2. These number correspond to chapter numbers in the governing specifications. Please note that these numbers are one greater than the corresponding chapter numbers in the X Consortium documentation as the Open Group specifications contain an additional introductory chapter.
3. This scenario is not usable with VSW5 Lite.

set of VSW5 executables can be quite large. Thus the recommended commands for executing the ‘all’ scenario, for example, are:

```
tcc -c vsw5 all; tcc -bec vsw5 all
```

5.3.1 Running a Single Test Purpose

The utility `run_assert` allows simple execution of individual VSW5 Test Purposes.

The command synopsis for `run_assert` is:

```
run_assert [ -Truss ] [ -NoClean ] TestCase AssertionNumber...
```

`TestCase` is the full name of a VSW5 Test Case scenario (i.e. XOpenDisplay, XtError, etc.). If the specified scenario is not found an error will be printed and `run_assert` will abort.

`AssertionNumber...` is a space separated list of Test Purposes to execute. If a nonexistent Test Purpose is specified an error will be printed. The program will continue and run all valid Test Purposes specified. Test Purposes are run in the order specified.

Other options for `run_assert` are documented in a manual page contained in the file `$TET_ROOT/vsw5/man/run_assert.man`.

This utility is not a supported part of VSW5 and is not required for implementation conformance.

6.0 VSW5 REPORTS

VSW5 includes three utilities that produce formatted reports from TET journal files produced by VSW5:

- `vswrpt` produces a report from a journal for a single run of VSW5
- `vswrptm` produces a report comparing the results of several VSW5 journals
- `vswrptx` produces a formal Open Group conformance report from a VSW5 journal

This chapter describes the contents of VSW5’s journal files and provides an overview of the VSW5 report generators.

6.1 JOURNAL FILES

VSW5 creates a TET journal file when it is run. For each Test Purpose executed several pieces of information are placed in the journal file:

- The associated assertion number.
- The assertion text.
- A description of the test method used.
- The result of the test.
- A description of any errors observed.

A sample journal file entry, with TET journal formatting information removed, is shown in Figure 1.

```
SECTION: Xt3

TEST CASE: XtIsComposite

TEST PURPOSE #1
Assertion XtIsComposite-1.(A)
A call to Boolean XtIsComposite(w) shall return True if the
class of the widget w is equal to or a subclass of the
Composite widget class.
PREP: Initialize toolkit, Open display
PREP: Create windows for widgets and map them
TEST: Returns true for subclass of Composite
ERROR: Expected Return value of 1, Received 8
1 FAIL
```

FIGURE 1 - SAMPLE JOURNAL FILE INFORMATION

6.1.1 Assertion Text

Unless the test did not build the assertion for the test is output to the journal file.

6.1.2 Test Method Description

Test Purposes output informational messages to the journal file through *tet_infoline*. In the X Toolkit Intrinsics tests this occurs for each call made to a system function, and other significant pieces of code. This provides a high level “trace” of test execution. These messages start with “PREP:” for code that establishes the preconditions necessary to verify the assertion, “TEST:” for code that directly verifies the assertion and/or calls the function under test, and “CLEANUP:” for code that undoes the effects of the test. Significant interim results are output in messages starting with “INFO:”.

In tests other than those for the X Toolkit Intrinsics the test method is output immediately following the assertion in messages prefixed with “METH” and messages prefixed with “TRACE”, “CHECK” and “REPORT” are emitted during test execution.

6.1.3 Test Results

Each Test Purpose executed has a result indicated in the journal file. The result indicates whether the test had an acceptable or unacceptable result, or that further analysis of the journal file information is needed to determine the outcome.

ACCEPTABLE RESULTS

RESULT	REASON
NOTINUSE	The assertion number is reserved - no test is provided.
PASS	The test verified the implementation under test performed as required.
UNSUPPORTED	The test is conditional and the behavior is not provided by the implementation under test.
UNTESTED	The assertion is an extended assertion - no test is provided.

UNACCEPTABLE RESULTS

RESULT	REASON
FAIL	The implementation under test’s behavior was not as required by the assertion.
ABORT	A fatal error occurred during test execution and execution

	was halted.
UNINITIATED	The test could not be built. UNINITIATED results are produced by the VSW5 report generators (see section 6.5 on page 53) when failures are encountered in building tests.
UNRESOLVED	The preconditions for the test could not be established.

FUTHER ANALYSIS REQUIRED

RESULT	REASON
FIP	Further Information Provided. Manual inspection of the test output is necessary to determine the test result. FIP results are produced by the VSW5 report generators (see section 6.5 on page 53) when compiler messages are produced in building a test.
WARNING	The implementation under test passed the test but something unexpected was observed, or, the expected behavior was not observed but it is defined in the specification only as behavior that “may” occur.

6.1.4 Error Descriptions

Conditions that cause an unacceptable result are documented in the journal file with messages starting with “ERROR:”, “WARNING:” or “FIP:” for X Toolkit Intrinsics tests and “REPORT:” for other tests. In the case of erroneous behavior by the implementation the actual behavior and, unless completely redundant with the assertion, the expected behavior are reported. Where a system function fails and sets `errno` the value of `errno` after the failure is included in the error message.

Generally a condition that causes a result other than PASS terminates a test immediately. In cases where the information that would result is considered of value the test is allowed to proceed.

6.2 vswrpt

The `vswrpt` utility produces a report from a single VSW5 journal file. VSW5 users should employ vswrpt to determine the results of testing, rather then examining raw journal files, since vswrpt provides additional information and more readable formatting.

`vswrpt` is documented in a manual page contained in the file `$TET_ROOT/vsw5/man/vswrpt.man`.

The journal file given to `vswrpt` must be the result of a tcc command that includes the -e option.

The -b and/or -c options may also have been used. `vswrpt` cannot be used on journal files that do not contain the additional information provided by VSW5. Some modes of the TET tcc (e.g. the -r mode) thus cannot be used with `vswrpt`.

`vswrpt` always produces a summary report of the results of the tests run in the input journal. It can also provide detailed information about the execution of Test Cases and Test Purposes. Three options control what additional information is reported.

-d detail	Specifies the level of detail to be provided. 0 - All information output by the test run. 1 - Reasonable detail (the default). 2 - Nothing - no detail information is generated. 3 - Results only. 4 - Assertions and results only.
-s scope	Specifies the type of Test Purposes for which the requested detailed information will be reported. 0 - All Test Purposes. 1 - Tests Purposes with result codes of FAIL, UNRESOLVED, UNINITIATED, ABORT, NORESULT, WARNING, or FIP (the default). 2 - Test Purposes with result codes of UNINITIATED. 3 - Test Purposes with result codes of UNSUPPPORTED. 4 - Test Purposes with result codes of WARNING or FIP. 5 - Test Purposes with result codes of NOTINUSE. 6 - Test Purposes with result codes of UNTESTED. 7 - Test Purposes with result codes of NORESULT.
-t test_case	Report additional information only for the specified Test Case. The default is to report information for all Test Cases.

By default `vswrpt` uses the most recent journal file under the directory `$TET_ROOT/vsw5/results` as input. Three options allow specification of alternative files for use as the input journal:

-f file_name	Use the file `file_name` as the input journal.
-j dir_number	Use the file `journal` in the directory under `$TET_ROOT/vsw5/results` with the specified number as the input journal. Leading zeros and additional portions of the directory name need not be specified, i.e. -j 30 matches 0030bec/journal, 0030b/journal, etc.
-u	Use the most recent journal created by the user running `vswrpt` as the input journal.

These three options cannot be used in combination.

Additional options are provided - these are described in the `vswrpt` manual page.

THE OPEN GROUP
VSW5 SUMMARY RESULTS REPORT

Test suite version: 5.0.0b1
Specification version: X/Open Window Management (X11R5) document set
Test run by: andy
System: UNIX_SV taffy 4.2MP 2.0 i386
Test run started: Saturday July 01, 1995 02:56:05 PM
Test run ended: Saturday July 01, 1995 05:52:49 PM
Journal file: /home/andy/tet/vsw5/results/0145bec/journal
TCC command line: tcc -bec vsw5 all
Report type: -d 2 -s 1

	CASES	TESTS	PASS	UNSUP	UNTST	NOTIU	WARN	FIP	FAIL	UNRES	UNIN	ABORT
Xproto	122	389	373	0	1	0	0	0	15	0	0	0
Xlib3	109	161	123	4	27	1	0	0	6	0	0	0
Xlib4	29	324	271	17	27	5	0	0	4	0	0	0
Xlib5	15	84	73	2	5	0	0	0	1	3	0	0
Xlib6	8	50	20	0	30	0	0	0	0	0	0	0
Xlib7	58	172	154	0	13	0	0	0	1	4	0	0
Xlib8	29	165	133	10	22	0	0	0	0	0	0	0
Xlib9	46	1472	1079	46	36	201	19	0	91	0	0	0
Xlib10	23	95	58	2	35	0	0	0	0	0	0	0
Xlib11	33	195	83	22	43	43	0	0	4	0	0	0
Xlib12	27	138	105	2	15	12	0	0	2	2	0	0
Xlib13	32	269	154	2	102	3	0	0	8	0	0	0
Xlib14	45	60	44	0	0	0	0	0	15	0	1	0
Xlib15	45	159	126	0	33	0	0	0	0	0	0	0
Xlib16	30	75	64	0	8	0	0	0	3	0	0	0
Xlib17	55	131	109	0	22	0	0	0	0	0	0	0
Xopen	8	127	121	0	0	0	0	0	6	0	0	0
Xt3	21	75	73	0	2	0	0	0	0	0	0	0
Xt4	33	203	94	0	19	89	0	0	1	0	0	0
Xt5	10	73	25	0	19	26	0	0	3	0	0	0
Xt6	7	71	71	0	0	0	0	0	0	0	0	0
Xt7	11	110	90	0	7	6	0	0	7	0	0	0
Xt8	7	50	35	0	11	0	0	0	4	0	0	0
Xt9	33	203	120	14	19	49	0	0	1	0	0	0
Xt10	8	25	16	0	9	0	0	0	0	0	0	0
Xt11	58	308	228	0	37	18	3	0	19	3	0	0
Xt12	22	72	52	1	6	10	0	3	0	0	0	0
Xt13	39	191	129	0	60	0	0	0	2	0	0	0
Xt14	2	18	18	0	0	0	0	0	0	0	0	0
Xt15	1	2	1	1	0	0	0	0	0	0	0	0
XtC	29	148	90	1	6	50	0	0	1	0	0	0
XtE	1	1	1	0	0	0	0	0	0	0	0	0
TOTAL	996	5616	4133	124	614	513	22	3	194	12	1	0

FIGURE 2 - SAMPLE SUMMARY REPORT

```
TEST PURPOSE #76
Assertion XDrawArcs-76.(A)
When a line has coincident endpoints (x1=x2, y1=y2), and
the cap_style is applied to both endpoints and the
line_width is not equal to zero and the cap_style is
CapRound, then the closed path is a circle, centered at the
endpoint, and with the diameter equal to the line-width.
METH: Draw zero length line with CapRound.
METH: Pixmap verify.
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0048.err for results
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0049.err for results
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0050.err for results
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0051.err for results
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0052.err for results
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0053.err for results
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0054.err for results
REPORT: A total of 3 out of 9000 pixels were bad
REPORT: Pixel check failed. See file Err0055.err for results
76 FAIL
```

FIGURE 3 - SAMPLE DETAILED REPORT INFORMATION

6.3 vswrptm

The `vswrptm` utility produces a report comparing the Test Purpose results from two to six journal files produced from runs of VSW5.

`vswrptm` is documented in a manual page contained in the file `$TET_ROOT/vsw5/man/vswrptm.man`.

The journal files given to `vswrptm` must be the result of tcc commands that include the -e option. The -b and/or -c options may also have been used. `vswrptm` cannot be used on journal files that do not contain the additional information provided by VSW5.

`vswrptm` always produces a summary report of the results of the test runs in the input journals. By default it also reports the results for all Test Purposes for which the results are not the same in each journal. Four options provide the ability to alter this later information.

-a	Show results for all Test Purposes, not just variations.

-f	Show results for Test Purposes where a failure is present as well as variations.
-u	Show results for Test Purposes where UNINITIATED results are present as well as variations.
-s section	Report results only for the specified VSW5 Section, e.g. Xlib3, Xt4, XProto.
-t test_case	Report results only for the specified Test Case.

Additional options are provided - these are described in the `vswrptm` manual page.

Sample `vswrptm` output is shown in Figure 4.

```
                         THE OPEN GROUP
               VSW5 JOURNAL FILE COMPARISON REPORT
                  VERSION 5.0.0 Snapshot 1


Report generated: 07/06/95 17:11:31

FILE 1 = 0045bec/journal
FILE 2 = 0145bec/journal

Variance for all Sections and Test Cases

TEST CASE           PURPOSE  FILE 1   FILE 2

XcmsAddFunctionSet     1     UNRES    pass
XcmsConversionProc     5     FIP      pass
XcmsConversionProc     6     FIP      pass
XcmsConversionProc     7     FIP      pass
XcmsConversionProc     8     FIP      pass
XcmsConversionProc     9     FIP      pass
XcmsStoreColor         1     FIP      pass
XcmsStoreColor         2     FIP      pass
XcmsStoreColor         3     FIP      pass
XcmsStoreColor         4     FIP      pass
XcmsStoreColors        1     FIP      pass
XcmsStoreColors        2     FIP      pass

12 variances found

TEST CASES                   714      714
TEST PURPOSES                4057     4057
GOOD RESULTS
 PASS                        3094     3106
 UNSUPPORTED                 102      102
 UNTESTED                    541      541
 NOTINUSE                    265      265
ANALYSIS NEEDED
 WARNING                     0        0
 FIP                         18       7
ERROR RESULTS
 FAIL                        18       18
 ABORT                       0        0
 UNRESOLVED                  19       18
 UNINITIATED                 0        0
 NORESULT                    0        0
```

FIGURE 4 - SAMPLE JOURNAL COMPARISON REPORT

6.4 vswrptx

`vswrptx`produces a document comprised of a `vswrpt` summary and other information required for formal Open Group conformance reporting.

By default `vswrptx` uses the most recent journal file under the directory `$TET_ROOT/vsw5/results` as input. Three options allow specification of alternative files for use as the input journal:

-f file_name	Use the file `file_name` as the input journal.
-j dir_number	Use the file `journal` in the directory under `$TET_ROOT/vsw5/results` with the specified number as the input journal. Leading zeros and additional portions of the directory name need not be specified, i.e. -j 30 matches `0030bec/journal, 0030b/journal`, etc.
-u	Use the most recent journal created by the user running `vswrptx` as the input journal.

These options cannot be used in combination.

Note that an Open Group conformance report must be based on a single journal containing the result of building and running the VSW5 "all" scenario.

`vswrptx` places a footer at the bottom of each page of the document showing the date, the page number, the system tested and the Test Centre which generated the report. Three options allow control of this footer:

-l lines	Break pages after the specified number of lines of report data. By default this number is 54.
-o organization	Specifies the Test Centre name. The string provided can be at most 50 characters.
-s system	Specifies the implementation tested. The string provided can be at most 55 characters.

The -o and -s options must be used when generating a report for submission to The Open Group.

`vswrptx` must be run on the system where the input journal was generated.

6.5 GENERATED RESULTS

In addition to reporting results produced by VSW5 Test Purposes the report generators are also responsible for generating two types of results:

- The report generators produce a result of UNINITIATED for any Test Purpose that is part of the requested scenario but is not executed. This is indicative of a build failure or a problem on the part of the TCC in running a test.

- The report generators produce a result of FIP when compiler messages are output during the build of a file containing Test Purposes and the result of a Test Purpose would otherwise be PASS. This indicates a potential problem with the definition of a function prototype, structure or other variable in an implementation header file. It is not possible to determine in a portable fashion which Test Purpose is referenced in a compiler message. Therefore if a compiler message is generated during the build of a file containing multiple Test Purposes, FIP results are produced for all of them.

7.0 TROUBLESHOOTING

This chapter provides recommendations and hints for troubleshooting problems that may be encountered in working with VSW5. Information on the structure of the VSW5 source code which may be of use in debugging problems is also provided.

7.1 GENERATING PIXMAP ERROR FILES

During VSW5 execution discrepancies may be encountered between the image displayed by the server and the expected image. The expected image may have been obtained from a known good image file supplied with the release, or it may have been determined analytically.

Should a discrepancy be encountered the test purpose will give a result code of FAIL. The failure reason message will name a pixmap error file in which is contained both the known good image and the server image.

A debug option has been provided, which skips any verification of the image produced by the server with known good image files. This is done by setting the execution configuration parameter XT_DEBUG_NO_PIXCHECK to Yes.

7.2 PIXMAP ERROR FILE NAMING SCHEME

Each invocation of the `tcc` utility creates a sub-directory in `$TET_ROOT/vsw5/results`. Sub-directories are created with sequential four digit numbers, with the tcc flags (e.g. “bec”) appended. The default TET journal is a file named `journal` created in this directory.

Pixmap error files are stored in a directory tree created within the newly created results sub-directory. So, for example, when the line

```
/tset/Xlib9/drwln
```

is executed in a scenario file, pixmap error files might be produced in a directory named `$TET_ROOT/vsw5/results/0001bec/tset/Xlib9/drwln`.

The creation of a new results directory tree for each execution of the `tcc` utility enables results to be obtained in parallel against multiple X servers.

Pixmap error files are named Err*nnnn*.err,where *nnnn* is a four digit number. This number does not correspond to the number of the test purpose which caused the error.

Note - if tests are executed without using the `tcc` utility the error files are produced in the current directory.

7.3 KNOWN GOOD IMAGE FILE NAMING SCHEME

All the required known good image files for the test programs in VSW5 (as supplied) have been created in advance. The known good image files for each test program are supplied in VSW5 in the test set directory in which the dot-m file is supplied. They are named a a*nnn*.dat, where *nnn* is the number of the test purpose for which the known good image file was generated.

7.4 USING BLOWUP TO VIEW IMAGE FILES

The contents of the two images in a pixmap error file may be compared by using the blowup utility.

Also, a known good image file may be viewed directly.

The file formats of the error file and the known good image file are the same. The blowup utility detects which file type is being viewed by means of file name. For this reason the pixmap error and known good image files should not be renamed.

7.4.1 Blowup command

The `blowup` utility may be used to view one or more pixmap error files or known good image files as follows:

blowup [-z zoom_factor] [-f font] [-d display] [-colour] file(s)

-z zoom_factor
This option sets the magnification factor in all the blowup windows.

-f font
This option ensures that `font` is used rather than the default font. The default font is 6x10.

-d display
This option uses the display named `display` for the display windows.

-colour
On a color display, this option will display different pixel values in different colors corresponding to that server’s color table. No attempt is made to preserve colors between different servers.

7.4.2 Blowup windows

Two windows are created. The first is called Comparison, and the second is called Blowup. The Blowup window shows a magnified version of a portion of the Comparison window, which is indicated in the Comparison window by a rectangle. A user interface menu is shown in the

Blowup window.

The title of the Comparison window will change to “Server Data”, then to “Pixval Data” and then back to “Comparison” when the “B/G/Both” option on the menu is used.

7.4.3 Selection of a viewing region

This may be done in one of three ways:

1) Click in the Comparison window.

2) Click in a square in the Blowup window. This becomes the new centre square in that window

3) Choose the “next error” option on the menu. The next pixel at which there is a discrepancy will become the new centre square in the Blowup window.

7.4.4 Information displayed

The value stored in the center pixel and its coordinates are shown as the top items in the menu. Under some circumstances, the expected pixel value will be shown to the right of the actual value.

7.4.5 Display of errors

When the “B/G/Both” option is set to Both, and the title of the Comparison window is Comparison, errors are displayed in two ways: one for each window.

- In the Comparison window pixels set to non-zero in the “good image” but set incorrectly in the “server data” are shown as a cross (X).

  In the blowup window these are shown as a white square with a cross (X) through it.

- In the Comparison window pixels set to zero in the “good image” but set incorrectly in the “server data” are shown as shaded squares.

  In the blowup window these are shown as a black square with a white cross through it.

7.4.6 Commands (via menu in the Blowup window)

All of the commands are invoked by clicking the left mouse button when the corresponding menu item is highlighted (inverted). The available commands are, from top to bottom:

B/G/Both

Show Bad (Server Data), Good (Pixval Data) or Both (Comparison). Clicking in this advances around the cycle

$$\begin{tikzcd}\text{Bad} \arrow[r] & \text{Good} \arrow[r] & \text{Both} \arrow[ll, bend left]\end{tikzcd}$$

The Comparison window's name changes to reflect the current state.

If a known good image file is being displayed then only the Good option is available. A pixmap error file is required for this command to be useful.

color/mono
: Use color/monochrome in the Blowup window.

next error
: Advance centre pixel point to be next pixel at which there is a discrepancy.

sub-zoom +
: Zoom in (make bigger by zoomfactor) on the Blowup window

sub-zoom -
: Zoom out (make smaller by zoomfactor) on the Blowup window

quit
: Quit from the blowup utility.

big-zoom +
: Zoom in (make bigger by zoomfactor) on the Comparison window

big-zoom -
: Zoom out (make smaller by zoomfactor) on the Comparison window

next
: View next file in the list. The Blowup window will be removed and a new one created for each file. The size, and zoom factor, of the Comparison window will be preserved across files

7.5 LIBRARIES AND TOOLS INSTALLATION

The libraries and tools for VSW5 must be built successfully before it can be run. Problems encountered in building them should be addressed before testing begins.

This code is designed to be very portable and problems in building it are likely to be due to conformance issues in the implementation's header files. The recommended troubleshooting procedure for build problems in these areas is to look first for header file issues.

Note that `wbuild` can be executed in any of the source directories for the tools and libraries to cause them to build without recourse to the master makefile. This facility should only be employed for troubleshooting purposes.

7.6 TEST RESULTS

A Test Purpose that generates a result of PASS, UNSUPPORTED, NOTINUSE, or UNTESTED is successful. A PASS result indicates the Test Purpose was run and the implementation performed as expected. An UNSUPPORTED result indicates the Test Purpose verifies optional functionality that is not supported by the implementation. NOTINUSE and UNTESTED results indicate the assertion number is reserved and no test is provided for it. Other test results indicate a lack of success to some degree.

FAIL, ABORT, and UNRESOLVED results indicate the Test Purpose was executed but the implementation did not perform as expected. For a FAIL result the problem was detected in the functionality being verified, for an UNRESOLVED or ABORT result the problem was detected with functionality used to set up the conditions necessary to perform the test.

An UNINITIATED result indicates the Test Purpose failed to build and so was not executed.

A NORESULT result indicates the Test Purpose was executed but failed to complete. This is an indication of a serious problem which prevented VSW5 from operating correctly.

FAIL, ABORT, NORESULT, UNRESOLVED, and UNINITIATED results need to be corrected.

FIP and WARNING results indicate a potential problem requiring investigation. FIP results need to be examined and either corrected or a rationale for there being no problem documented. WARNING results should be examined to ensure the behavior observed is that intended.

7.6.1 FAIL Results

A FAIL result is due to non-conforming behavior of the implementation.

FAIL results for tests that use configuration variables may be due to a configuration problem. Tests which used configuration variables place messages in the journal file indicating which variable they are using. For failures in these tests it is advisable to check the VSW5 configuration.

Check the detailed information in the report to determine the nature of the problem; correct it; and re-run the test.

7.6.2 FIP Results

FIPs are generated by the VSW5 report generators due to the production of compiler messages during the build of a file.. The compiler messages output during the build are captured in the journal file. By default `creport` displays the first 10 lines of compiler output per test. To view all these messages use the -d 0 option to `creport`.

It is not possible to determine in a portable fashion which Test Purpose is referenced in a compiler message. Therefor if a compiler message is generated during the build of a file containing multiple Test Purposes, FIP results are produced for all of them.

These problems are most often due to a non-conforming function prototype or structure definition in a header file. Check the compiler messages to determine where the problem is located; correct it; and re-run the test. Also see section 3.3 on page 13 for a discussion of the need to avoid spurious compiler messages that generate FIP results.

7.6.3 NORESULT Results

A NORESULT result is due to a Test Purpose having completed without indicating a result to TET. A NORESULT result is an indication of a serious problem which has prevented VSW5 from being able to verify the behavior of the implementation.

Check the detailed information in the report to determine at what point the Test Purpose stopped executing; correct the problem; and re-run the test.

7.6.4 UNINITIATED Results

When a file containing Test Purposes fails to build the Test Purposes have a result of UNINITIATED in the test report. The compiler messages output during the build are captured in the journal file. By default `creport` displays the first 10 lines of compiler output per test. To view all these messages use the -d 0 option to `creport`.

These failures are most often due to a missing library function or header file definition. Check the compiler messages to determine where the problem is located; correct it; and re-run the test.

7.6.5 UNRESOLVED and ABORT Results

An UNRESOLVED or ABORT result is due to non-conforming behavior of the implementation detected by the Test Case in setting up preconditions necessary to verify an assertion.

UNRESOLVED and ABORT results for tests that use configuration variables may be due to a configuration problem, particularly a missing or malformed configuration variable value. Tests that use configuration variables place messages in the journal file indicating which variable they are using. For failures in these tests it is advisable to check the VSW5 configuration

In some tests, reliance is made on the successful behaviour of another area of VSW5. Where this reliance is made, and that area of VSW5 does not behave as expected, a result code UNRESOLVED may occur. The test information messages should indicate the area of the underlying problem. It may be necessary to look at the test results for that area first and investigate and resolve the underlying problem before re-running the UNRESOLVED tests.

Tests which give a result code of UNRESOLVED and the message “Path check error” normally contain programming errors. The test reached the point at which a PASS result would be assigned, but the number of check points passed in executing the test code differs from the expected number.

Tests which give a result code of UNRESOLVED and the message "No CHECK marks encountered" may be due to programming errors. The test reached the point at which a PASS result would be assigned, but no check points had been passed. This can also occur when tests are executed using debug options. For example, the message occurs when tests are executed which normally loop over windows and pixmaps and

```
XT_DEBUG_PIXMAP_ONLY=Yes
        or
XT_DEBUG_WINDOW_ONLY=Yes
        or
XT_DEBUG_DEFAULT_DEPTHS=Yes
        or
XT_DEBUG_VISUAL_IDS=Yes .
```

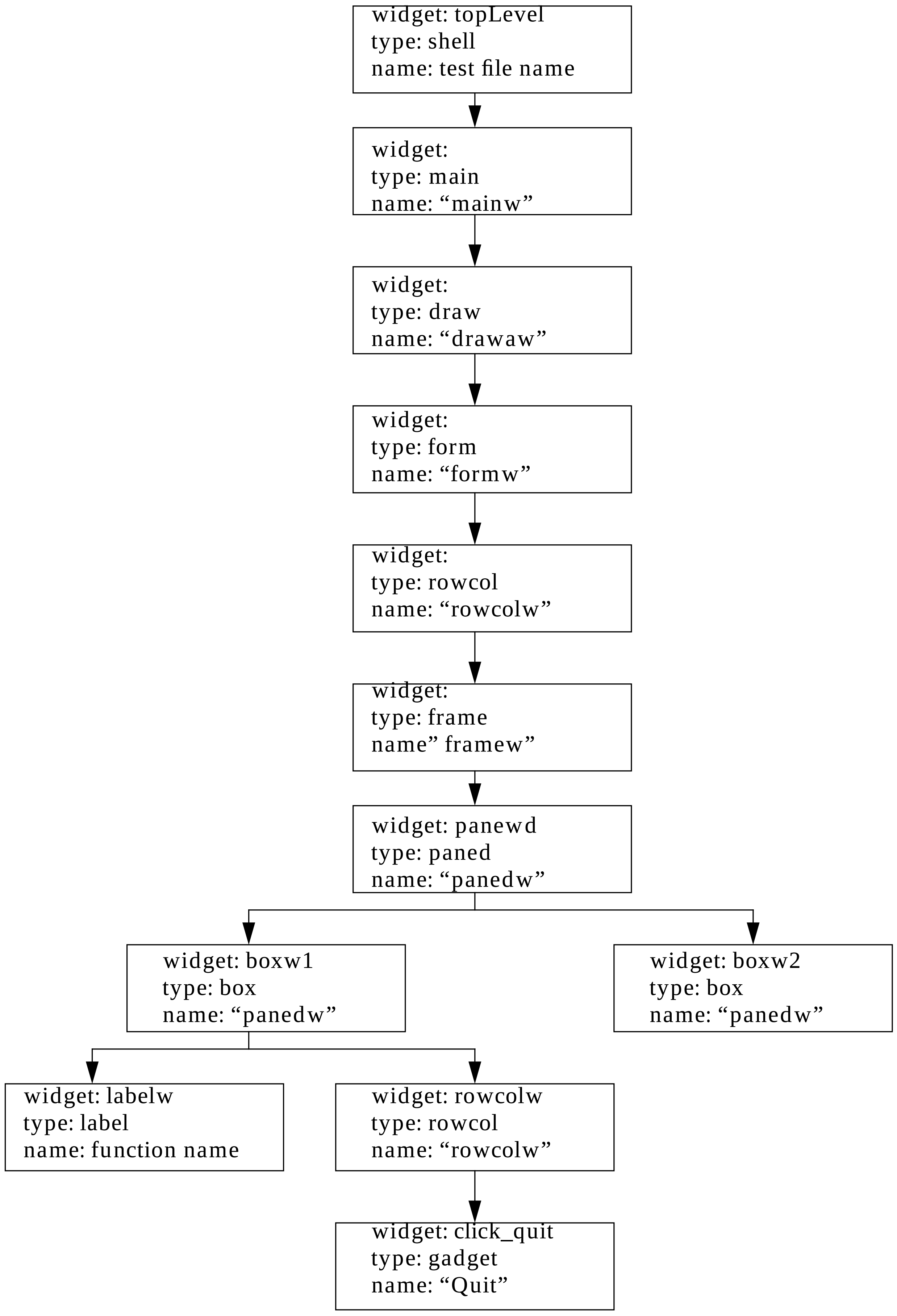
Check the detailed information in the report to determine the nature of the problem; correct it; and re-run the test.

7.6.6 WARNING Results

If a WARNING is encountered the behavior of the implementation should be evaluated to determine if it conforms to the Open Group Window Management document set or if the WARNING indicates an implementation problem. If an implementation problem is found, correct it and re-run the test. If no problem is found the WARNING result can be ignored.

7.7 XT TEST WIDGET HIERARCHY

The following figure shows the widget hierarchy used in most VSW5 Xt tests. Tests are performed on these widgets and test-specific widgets created as their children.


widget: topLevel
type: shell
name: test file name
widget:
type: main
name: “mainw”
widget:
type: draw
name: “drawaw”
widget:
type: form
name: “formw”
widget:
type: rowcol
name: “rowcolw”
widget:
type: frame
name” framew”
widget: panewd
type: paned
name: “panedw”
widget: boxw1
type: box
name: “panedw”
widget: boxw2
type: box
name: “panedw”
widget: labelw
type: label
name: function name
widget: rowcolw
type: rowcol
name: “rowcolw”
widget: click_quit
type: gadget
name: “Quit”

8.0 SUPPORT AND INTERPRETATION PROCEDURES

8.1 SUPPORT REQUESTS

A Support Request (SR) is the mechanism for licensees of VSW to:

- report suspected VSW problems
- suggest VSW enhancements
- request general information about VSW.

Procedures and forms for submitting SRs are available at The Open Group's World Wide Web site using the URL http://www.opengroup.org/testing/support/.

8.2 INTERPRETATION REQUESTS

An Interpretation Request (IR) is the mechanism for licensees of VSW to:

- request a specification interpretation
- request a formal branding waiver due to a test suite deficiency
- request a formal branding waiver due to a minor system conformance issue

Procedures and forms for submitting IRs are available at The Open Group's World Wide Web site using the URL http://www.opengroup.org/interpretations/interpform.html.

A database of IRs to date for VSW is available on the World Wide Web using the URL http://www.opengroup.org/interps.

8.3 CSQS

Conformance Statement Questionnaires need completing when making a formal application for UNIX product registration. These can be completed electronically, see The Open Group's World Wide Web site for more information: URL http://www.opengroup.org/public/csq.

APPENDIX A

A.1 GLOSSARY

Assertion	A statement of functionality that is true for a conforming system. Assertions are used to define the scope of the tests contained in VSW5. The practice of using assertions for test suite specification was codified in POSIX.3.
Base Assertion	An assertion that is required to be tested.
Extended Assertion	An assertion that is not required to be tested.
Journal File	The file into which test results and tracking data are deposited by the TET TCC.
Result Code	The determination made by a TET Test Case about a Test Purpose.
POSIX.3	The IEEE P1003.3 specification.
Scenario	A sequence of one or more Invocable Components associated with a single user-exposed name. There may be multiple scenarios per scenario file. Scenarios are used by the TCC to translate user requests into actions.
Test Case	The software that conducts tests. The scope of the term "test" is broad. It may range from a single Test Purpose for a single function being tested, all the way to a complete suite of conformance tests for a specification.
Test Purpose	The software which tests that statement that is the smallest level of granularity of a test specification. A Test Purpose always leads to a single result. In a POSIX.3 conforming test suite Test Purposes correspond to assertions.
TCC	The TET tool that provides structure and control for Test Cases. The TCC builds Test Cases when invoked in build mode, removes them in clean mode, and executes the Invocable Components within Test Cases when in execute mode. The tool handles functions such as sequencing of Test Cases, parameter passing, and journal file construction.
TET	The Test Environment Toolkit, a user interface and programming environment for test suites developed by X/Open, the Open Software Foundation, and UNIX International.

INDEX