Xlib and X Protocol Test Suite
X Version 11 Release 6.1

User Guide for the X Test Suite

July 1992

Copyright © 1991, 1992 UniSoft Group Limited

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of UniSoft not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. UniSoft makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

User Guide for the X Test Suite

1. Introduction

This document is a guide to installing and running X Version 11 Release 6.1 of the X Test Suite. In order to do this, please work through all of the steps described in this guide in order. Information on the content, purpose and goals of the X Test Suite is found in a series of appendices, which follow the installation instructions.

Please read the "Release Notes for the X Test Suite", which describe particular features of this release.

Further information, which would be required by a programmer to modify or extend the X Test Suite, is contained in a separate document, "Programmers Guide for the X Test Suite".

Included in this release is a version of the "Test Environment Toolkit" (TET). This is required to build and execute the X Test Suite. The "Test Environment Toolkit" is a software tool developed by X/Open, UNIX International, and the Open Software Foundation. More details of the TET appear in appendix E.

The contents of this document cover the installation and use of the included version of the TET.

2. Preparation

This section of the User Guide describes how to check that the system on which you want to build the X Test Suite has the required utilities and sufficient disc space, how to check the version of the X Window System[1] you wish to test, and how to extract the software from the supplied distribution media.

2.1 Utilities

The X Test Suite assumes that the following utilities are available on your system.

2.1.1 Bourne shell

The configuration and building stages include example instructions 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.

2.1.2 make

The building stages assume the existence of `make`.

2.1.3 awk

The report writer `rpt` uses `awk`.

2.1.4 Compiler

A C compiler and link editor are required. The X Test Suite assumes that when these utilities execute successfully, they return a value of zero. The names of these utilities may be set in build configuration parameters.

2.1.5 Library archiver

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 build configuration parameters.

2.1.6 File utilities

The X Test Suite uses utilities to copy, move, remove and link files during the build stages. The names of these utilities may be set in build configuration parameters.

2.2 Checking your version of the X Window System

If your version of the X Window System supports the XTEST extension, you 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, based on a specification[2] produced by UniSoft. The extension provides access to the X server to enable testing of the following areas of the X Window System:

---

1. The X Window System is a trademark of the Massachusetts Institute of Technology.
   X Window System Version 11 Release 4 is abbreviated to X11R4 in this document.
   X Window System Version 11 Release 5 is abbreviated to X11R5 in this document.

2. Drake, K.J., “Some Proposals for a Minimal X11 Testing Extension.” *UniSoft Ltd. June 1991*

— Those which rely on the simulation of device events.

— Those requiring access to opaque client side data structures.

— Those requiring information on the cursor attribute of windows.

Before you configure the X Test Suite, you should determine whether your version of the X Window System includes the XTEST extension, and, if so, whether you wish to configure and build the X Test Suite to enable these features to be tested.

There are two things to check:

1. Check whether your X server supports the XTEST extension. This can be done by printing the list of extensions available in your 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.

2. Check whether you have 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 normally named `libX11.a`. If you want to access the XTEST extension, you will need two further libraries. These are the XTEST library (normally named `libXtst.a`) and the X extension library (normally named `libXext.a`).

2.3 Installing the X Test Suite

Change to the directory in which you wish to install the distribution. Set an environment variable TET_ROOT to the full path name of that directory.

Load the software from the media supplied into that directory. The precise commands you should use depend on the format of the media supplied to you, the utilities available on your system, the options supported by the utilities, and the names of the tape devices on your system. See the Release Notes for more information about installation.

3. *Configuring the X Test Suite*

This section contains instructions on all the procedures you should go through in order to configure the X Test Suite, before attempting to build it.

There is a description of the TET build tool in section 3.1, and the relationship between the TET build scheme and the Imake scheme in section 3.2.

Sections 3.3 and 3.4 contain details of build and clean configuration parameters, which you should edit to reflect the configuration of the target platform on which the X Test Suite is to be built.

Section 3.5 contains details of source files and include files which contain system dependent data which cannot be specified via the build configuration parameters. You should check these files before configuration and if necessary edit them to be suitable for your system.

3.1 *The TET build tool*

The TET provides a scheme to execute a build tool, which builds the tests in the X Test Suite. The execution of the build tool in the TET is controlled by a small number of TET configuration parameters, contained in a build configuration file. These are described in section 3.3.1.

A build tool has been developed and is provided as part of the X Test Suite. This is a shell script named `pmake`, which is supplied in the directory `$TET_ROOT/xtest/bin`. The shell script `pmake` is an interface to the `make` system command, and when invoked from the TET it builds a test using the rules provided in a `Makefile`.

Each `Makefile` in the X Test Suite is written portably, 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 `pmake` from additional parameters in the build configuration file which are described in sections 3.3.2-3.3.7.

The `pmake` utility may be invoked directly from the shell, as well as via the TET, to build individual parts of the X Test Suite. This is described further in subsequent sections of this guide.

There is also a clean tool `pclean` which is an interface to the system `make clean` system command. This uses parameters in a clean configuration file.

3.2 *Relationship between TET build scheme and Imake*

The TET is designed to provide a simple and self contained interface to configure and build tests. The X Test Suite can be configured and built with no specialised knowledge of the X Window System beyond that contained in the X Test Suite documentation, and using a limited set of commonly available system commands. The only information required to configure and build the X Test Suite is the location of the X Window System Xlib and include files.

The X Window System itself includes a configuration scheme which is known as `Imake`. This uses a utility `imake` supplied as part of the X Window System to create `Makefiles` from portable description files called `Imakefiles`.

If you are familiar with the `Imake` scheme, and have used it to configure and build the

X Window System on the platform being used to build the X Test Suite, you may be able to set a limited number of the TET build configuration variables described in section 3.3 to the same value you used for an `Imake` variable. Where this is possible, the name of the corresponding `Imake` variable is cross referenced.

3.3 Build configuration parameters

All build configuration parameters are contained in a configuration file that forms part of the TET. This file should be edited to reflect the configuration of the target machine. The file

```
$TET_ROOT/xtest/tetbuild.cfg
```

contains all the parameters that are needed to build the X Test Suite. The parameters are grouped in seven sections within the configuration file.

3.3.1 Configuration Parameters defined by the TET

None of these parameters require changing. They are already set to defaults which are correct for the X Test Suite.

TET_BUILD_TOOL

The name of the program that the TET will execute in build mode.

You should use the `pmake` command that is supplied in the directory `$TET_ROOT/xtest/bin`.

```
Eg: TET_BUILD_TOOL=pmake
```

TET_BUILD_FILE

Any flags required by the build tool. This parameter should be empty.

```
Eg: TET_BUILD_FILE=
```

TET_CLEAN_TOOL

The name of the program that the TET will execute in clean mode.

You should use the `pclean` command that is supplied in the directory `$TET_ROOT/xtest/bin`.

```
Eg: TET_CLEAN_TOOL=pclean
```

TET_CLEAN_FILE

Any flags required by the clean tool. This parameter should be empty.

```
Eg: TET_CLEAN_FILE=
```

TET_OUTPUT_CAPTURE

This flag is used by the TET to enable the output from the build tool to be saved and copied into the journal file. This line should not be altered.

3.3.2 Configuration for system commands

In this section the names of system commands are specified.

SHELL — The following line should cause the Bourne shell to be used by make.

```
Eg: SHELL=/bin/sh
```

CC — A command to invoke the C compiler.

```
Eg: CC=cc
```

Imake variable: CcCmd

RM — A command to remove a file without interactive help.

```
Eg: RM=rm -f
```

Imake variable: RmCmd

AR — A command to generate a library archive.

```
Eg: AR=ar crv
```

Imake variable: ArCmd

LD — A command to link object files.

```
Eg: LD=ld
```

Imake variable: LdCmd

LN — A command to make hard links.

```
Eg: 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, then set the parameter to that command. Otherwise it should be set to `echo` (or a command that does nothing).

```
Eg: 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 if 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 if RANLIB was set to a command which creates a random access library. otherwise set to `lorder`.

CP — A command to copy files.

```
Eg: CP=cp
```

Imake variable: CpCmd

CODEMAKER — A utility to produce C source files from dot-m files. The supplied utility `mc` should always be used. This line should not be altered.

```
Eg: CODEMAKER=mc
```

3.3.3 Configuration for the TET

This section contains the locations of various parts of the TET. Usually only the first four parameters will need changing, unless files have been moved from their default locations.

TET_ROOT — The directory that contains all the files in the X Test Suite. This should be set to the path to which `TET_ROOT` was set (see the section entitled "Installing the X Test Suite"). It must be written out as a full path without using any variable notation.

TETBASE — The directory that contains all the files in the TET system. This is used for convenience in defining the other directories. This should be set to `${TET_ROOT}/tet`.

PORTINC — An option that can be given to the C compiler that will cause it to search all directories that are required to allow portability to systems that do not support POSIX. Should be empty for POSIX systems. If compiling on a BSD system using the supplied compatibility library, then the following line should be used. (See section entitled "The portability library").

```
Eg: PORTINC=-I${TET_ROOT}/port/INC
```

PORTLIB — A library containing POSIX.1 and C library functions that are not supplied by the system. This should be empty for a POSIX system. If compiling on a BSD system using the supplied compatibility library, then the following line should be used. (See section entitled "The portability library").

```
Eg: PORTLIB=${TET_ROOT}/port/libport.a
```

TETINCDIR — The directory containing the TET headers.

```
Eg: TETINCDIR=${TETBASE}/inc/posix_c
```

TETLIB — The directory containing the TET library.

```
Eg: TETLIB=${TETBASE}/lib/posix_c
```

TCM — The Test Control Manager. This is part of the TET. It is an object file that is linked with each test.

```
Eg: TCM=${TETLIB}/tcm.o
```

TCMCHILD — The Test Control Manager. This is part of the TET. It is an object file that is linked with each program that is executed within a test by tet_exec().

```
Eg: TCMCHILD=${TETLIB}/tcmchild.o
```

APILIB — The TET API library.

```
Eg: APILIB=${TETLIB}/libapi.a
```

3.3.4 Configuration parameters for the X Test Suite

Only the first two of these parameters require changing unless directories have been moved from their default locations.

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 you have set on your host, or may be set 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.

```
Eg. XTESTHOST=`hostname`
Eg. XTESTHOST=`uname -n`
Eg. XTESTHOST=triton
```

XTESTFONTDIR The directory in which to install the test fonts.

```
Eg: XTESTFONTDIR=/usr/lib/X11/fonts/xtest
```

XTESTROOT The directory that is the root of the X Test Suite.

```
Eg: XTESTROOT=${TET_ROOT}/xtest
```

XTESTLIBDIR The directory containing libraries for the X Test Suite.

```
Eg: XTESTLIBDIR=${XTESTROOT}/lib
```

XTESTLIB The xtest library. This library contains subroutines that are common to many tests in the X Test Suite.

```
Eg: XTESTLIB=${XTESTLIBDIR}/libxtest.a
```

XSTLIB The X Protocol test library. This library contains subroutines that are common to many tests in the X Protocol section of the X Test Suite.

```
Eg: XSTLIB=${XTESTLIBDIR}/libXst.a
```

XTESTFONTLIB The fonts library. This library contains font descriptions that are common to many tests in the X Test Suite.

```
Eg: XTESTFONTLIB=${XTESTLIBDIR}/libfont.a
```

XTESTINCDIR The xtest header file directory. This directory contains headers that are local to the X Test Suite.

```
Eg: XTESTINCDIR=${XTESTROOT}/include
```

XTESTBIN The xtest binary file directory. This directory contains utility programs that are used by X Test Suite.

```
Eg: XTESTBIN=${XTESTROOT}/bin
```

3.3.5 System Parameters

Location of system libraries and include files.

SYSLIBS — Options to cause the C compiler to search any system libraries that are required for the X Test Suite that are not searched by default. This will probably include Xlib.

```
Eg: SYSLIBS=-lX11
```

If you wish to build the X Test Suite to make use of the XTEST extension, you will need to include the XTEST library and the X extension library (in that order).

```
Eg: SYSLIBS=-lXtst -lXext -lX11
```

Imake variables: ExtraLibraries

XP_SYSLIBS — Any system libraries that are needed, to link the X Protocol tests. This will include Xlib, since libXst.a (which is part of the test suite) will include at least one call to XOpenDisplay.

```
Eg: XP_SYSLIBS=-lX11
```

Imake variables: ExtraLibraries

SYSINC — Any commands that should be given to the C compiler to cause all relevant system include files to be included. This will probably include /usr/include/X11.

```
Eg: SYSINC=-I/usr/include/X11
```

3.3.6 C Compiler Directives

Directives to the C compiler. Usually only the first four parameters will need changing. The remainder are internally used parameters, which are an amalgam of previously set parameters.

COPTS — Options to the C compiler.

```
Eg: COPTS=-O
```

Imake variables: DefaultCDebugFlags and DefaultCCOptions

DEFINES — Options required by the C compiler to set up any required defines. For example in strict ANSI Standard-C systems you will need to define _POSIX_SOURCE. Additionally on an X/Open conformant system it may be necessary to define _XOPEN_SOURCE.

```
Eg: DEFINES=-D_POSIX_SOURCE
```

If there is no symbol `NSIG` defined in the system header file `signal.h`, then this has to be supplied for use by the TET API. It should be the number of signal types on the system.

```
Eg: DEFINES=-D_POSIX_SOURCE -DNSIG=32
```

If you wish to build the X Test Suite to make use of the XTEST

extension, you will need to define XTESTEXTENSION.
XTESTEXTENSION is only used when building the X Test Suite library.

```
Eg: DEFINES=-D_POSIX_SOURCE -DNSIG=32 -DXTESTEXTENSION
```

Imake variables: StandardDefines

XP_DEFINES — C compiler defines specific to the X Protocol tests.
This can be set as DEFINES, but you can build support for additional connection methods beyond TCP/IP, using the following defines, if XP_OPEN_DIS is XlibNoXtst.c (R4/R5 XOpenDisplay emulation):

```
-DDNETCONN - Connections can also use DECnet[3].
-DUNIXCONN - Connections can also use UNIX[4] domain sockets.
```

Refer to your documentation for building and installing Xlib on your platform.
If XP_OPEN_DIS is one of XlibXtst.c or XlibOpaque.c then none of the defines listed above will be required.

```
Eg: XP_DEFINES=-D_POSIX_SOURCE -DUNIXCONN
```

Imake variables: StandardDefines

LINKOBJOPTS — Options to give to the LD program to link object files together into one object file that can be further linked.

```
Eg: LINKOBJOPTS=-r
```

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

```
INCLUDES=-I. ${PORTINC} -I${TETINCDIR} -I${XTESTINCDIR} ${SYSINC}
```

CFLAGS — Flags for the C compiler. This should not need changing as it is just an amalgam 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)

```
CFLAGS=$(CFLOCAL) $(COPTS) $(INCLUDES) $(DEFINES)
```

XP_CFLAGS — Flags for the C compiler. This parameter is used by the X Protocol tests in the X Test Suite. This should not need changing as it is just an amalgam of other parameters.

```
XP_CFLAGS=$(CFLOCAL) $(COPTS) $(INCLUDES) $(XP_DEFINES)
```

---

4. *DEC and DECnet are registered trademarks of Digital Equipment Corporation.*

4. *UNIX is a registered trademark of UNIX System Laboratories, Inc. in the U.S. and other countries.*

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.

```
Eg. LDFLAGS=-ZP
```

LIBS — List of libraries. This should not need changing as it is just an amalgam of other parameters.

```
LIBS=${XTESTLIB} ${XTESTFONTLIB} ${APILIB} ${PORTLIB}
```

XP_LIBS — List of libraries. This parameter is used by the X Protocol tests in the X Test Suite. This should not need changing as it is just an amalgam of other parameters.

```
XP_LIBS=${XSTLIB} ${XTESTLIB} ${XTESTFONTLIB} ${APILIB} ${PORTLIB}
```

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 your Xlib includes post R5 enhancements to _XConnectDisplay ensuring maximum portable protocol test coverage. These enhancements include arguments to _XConnectDisplay to return authorisation details on connection. If you use this option when your Xlib does not have these enhancements to _XConnectDisplay, the results of running the X Protocol tests will be **undefined**.

XlibOpaque.c
: You have a normal R4 Xlib or early R5 Xlib which you cannot patch to include the enhancements to _XConnectDisplay, and you cannot emulate these 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 you 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 your Xlib documentation for further details.

```
Eg. XP_OPEN_DIS=XlibOpaque.c
```

3.3.7 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 development environment at UniSoft when generating known good image files.

3.4 Clean configuration parameters

The TET provides a scheme to execute a clean tool, which removes previously built tests and object files.

All clean configuration parameters are contained in a configuration file that forms part of the TET. The file

```
$TET_ROOT/xtest/tetclean.cfg
```

contains all the parameters that are needed to clean the X Test Suite.

To save configuration effort, we have arranged that the build and clean configuration files may contain identical parameter settings. Both files are needed, since the TET requires both a default build and clean configuration file.

Copy the build configuration file into the clean configuration file:

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

3.5 System dependent source files

This section describes source files and include files provided in the X Test Suite which contain data, which you may need to edit to reflect the system under test.

3.5.1 Host address structures

The file `xthost.c` in the directory `$TET_ROOT/xtest/src/lib` contains three items, which you may need to edit to reflect the system under test. These are all related to the mechanisms provided by the X server under test to add, get or 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. You should refer to the Xlib documentation for your system, to determine the allowed formats for host addresses in an XHostAddress structure. You may also find it 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. You should find out which families are supported for the X server under test, by examining the header files supplied with your 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 your system.

The three items are as follows:

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

2. You should ensure that there is a declaration for an array xtbadhosts[] of at least 5 XHostAddress structures containing invalid family,length,address triples. You should ensure that there is a declaration for an array xtbadhosts[] of at least 5 XHostAddress structures containing invalid family,length,address triples. If you cannot use the supplied examples, 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.

3. You should 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).

4. Building the TET

The X Test Suite runs under the Test Environment Toolkit (TET).

This section of the User Guide tells you how to build and install the supplied version of the TET.

The following instructions assume the use of a Bourne shell.

The PATH variable should have the directory `$TET_ROOT/xtest/bin` prepended to it.

```
PATH=$TET_ROOT/xtest/bin:$PATH
export PATH
```

4.1 The portability library

The current version of the TET used by the X Test Suite is designed to run on a POSIX.1 system.

Since many systems running the X Window System currently run on BSD based systems a portability library that emulates the required routines using BSD facilities has been provided.

This library is not part of the TET itself.

The portability library source is kept in `$TET_ROOT/port`.

The portability library may be useful in porting the X Test Suite to other environments as described below. Please refer to the "Release Notes for the X Test Suite" for details of the target execution environments, and a list of systems to which it has already been ported.

4.1.1 Porting to a POSIX.1 system

If your system conforms to POSIX.1 and has an ANSI Standard-C compiler then this library should not be built and so this section can be skipped. The exception is that `putenv()` may not exist on a POSIX.1[5] system, however it is in the SVID[6] and the XPG[7], so in practice most non-BSD machines will have this function.

4.1.2 Porting to a BSD system

If the system is a standard BSD one, then the portability library can be used as it is; build it as follows.

```
cd $TET_ROOT/port
pmake
```

4.1.3 Porting to other systems

The portability library may be useful as a base for porting the TET to other non-POSIX systems, however the portability library is designed to run on a BSD system, and will not necessarily build without change on other systems.

---

5. *IEEE Std 1003.1-1990, Portable Operating System Interface for Computer Environments*

6. System V Interface Definition, Issue 1, AT&T, Spring 1985.

7. *X/Open Portability Guide Issue 3, Volume 2: XSI System Interface and Headers*

The following routines are emulated for use under a BSD system, and these may be needed on other systems:

```
getcwd() getopt() putenv() sigaction()
sigaddset() sigdelset() sigemptyset() sigfillset()
sigismember() sigpending() sigprocmask() sigsuspend()
strchr() strcspn() strftime() strpbrk() strrchr()
strspn() strtok() toupper() upcase()
vsprintf() waitpid()
```

Only the features that are used by the X Test Suite are emulated. They are not meant to mimic completely the standard behaviour.

There is also an include directory `$TET_ROOT/port/INC` that contains header files that are required that are not found on a BSD system. These files contain only items that are needed for the X Test Suite, they are not designed to replace completely the standard ones.

To adapt the portability library to other systems, the following hints may be found useful:

- Examine the directory `$TET_ROOT/port/INC`. If the system already provides a standard conforming header file of the same name as one in the INC directory, then remove the version from the INC directory.
- The header files contain the bare minimum required to compile the X Test Suite, and use BSD features. It may be necessary to alter them to suit the local system. This applies particularly to `signal.h`.
- It may be necessary to add other header files.
- In the library `Makefile` remove any function that is already provided by the system in a standard conforming form.
- Examine the code of the remaining functions to make sure that they will work on the target system.

4.2 Building libraries and utilities

There is a top level Makefile which can be used to automatically perform a number of the following steps. You should still check through the User Guide and perform the steps which need to be done manually. In particular you need to build and install the test fonts as described in the section entitled "Compiling and installing the test fonts".

The top level Makefile enables the following steps to be performed:

Building the TET:

1. The Test Case Controller (TCC)
2. The API library

Building the X test suite libraries and utilities:

1. Building the X test suite library
2. Building the X Protocol library

3. Building the X test fonts library
4. Building the mc utility
5. Building the blowup utility

To use the top level Makefile, move to the top level directory:

```
cd $TET_ROOT
```

Make the utilities and libraries with the command:

```
make
```

4.3 The Test Case Controller (TCC)

Move to the directory containing the TCC source.

```
cd $TET_ROOT/tet/src/posix_c/tools
```

Make the TCC with the command:

```
pmake install
```

Note: the supplied version of the TCC assumes that the *cp* utility on your system supports recursive copy using the option *-r*. There are two occurrences of *cp* in the file `exec.c` which use this option.

In the X Test Suite, recursive copying is not required.

If your system does not support this option, you can remove the use of this option in the source code before building the TCC. If you do this you may not be able to use the supplied TCC with other test suites.

Alternatively, you can provide a shell script in the directory `$TET_ROOT/xtest/bin` which copies files using *cp* but ignores any option *-r*.

4.4 The API library

Move to the API library source directory.

```
cd $TET_ROOT/tet/src/posix_c/api
```

Run the command

```
pmake install
```

which should produce the files `libapi.a` and the Test Case Manager files `tcm.o` and `tcmchild.o`.

5. Building the X test suite libraries and utilities

5.1 The X test suite library

A library of common subroutines for the X Test Suite has source in `$TET_ROOT/xtest/src/lib`. This is built automatically when building tests in the X Test Suite. Should it be required to build it separately for any reason run the command.

```
cd $TET_ROOT/xtest/src/lib
pmake install
```

The list of source files in this library is described in the "Programmers Guide".

5.2 The X Protocol library

A library of common subroutines for the X Protocol tests in the X Test Suite has source in `$TET_ROOT/xtest/src/libproto`. This is built automatically when building tests in the X Test Suite. Should it be required to build it separately for any reason run the command.

```
cd $TET_ROOT/xtest/src/libproto
pmake install
```

The list of source files in this library is described in the "Programmers Guide".

5.3 The X test fonts library

A library of common subroutines defining the characteristics of the test fonts for the X Test Suite has source in `$TET_ROOT/xtest/fonts`. This is built automatically when building tests in the X Test Suite. Should it be required to build it separately for any reason run the command.

```
cd $TET_ROOT/xtest/fonts
pmake install
```

The list of source files in this library is described in the "Programmers Guide".

Note that the directory `$TET_ROOT/xtest/fonts` also contains the test fonts themselves in bdf format, which must be compiled and installed. Instructions for performing these steps are included in the next section entitled "Compiling and installing the test fonts".

5.4 Compiling and installing the test fonts

The X Test Suite 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 the expected font characteristics which are compiled into the tests via the X test fonts library. The X test fonts library is described in an earlier section of this document.

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

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

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

The manner in which fonts should be compiled and installed for any particular X server is system dependent, and you should refer to the instructions supplied with your release of the X Window System for details of how to do this.

Some sample instructions are given here which may be useful on many systems. These may not be appropriate for your system, or they may need adaptation to work properly on your system and so are provided only as a guide.

1. Move to the directory `$TET_ROOT/xtest/fonts`.

```
cd $TET_ROOT/xtest/fonts
```

2. Compile the seven `bdf` files into `snf`, `pcf`, or `fb` format, as appropriate for your system.

```
pmake comp_snf
```

or

```
pmake comp_pcf
```

or

```
pmake comp_dxpcf
```

or

```
pmake comp_fb
```

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

```
pmake install_snf
```

or

```
pmake install_pcf
```

or

```
pmake install_dxpcf
```

or

```
pmake install_fb
```

5.5 Building the `mc` *utility*

The `mc` utility is used to generate test set source files and Makefiles from a template file, known as a dot-m file. The file naming scheme is described further in appendix B. The file formats are described further in the "Programmers Guide".

The Makefiles and test set source files will be created using `mc` whenever test sets are built, if the dot-m file is found to be newer than the source file or Makefile, or if these files do not exist.

Build `mc` and install in the xtest bin directory as follows.

```
cd $TET_ROOT/xtest/src/bin/mc
pmake install
cd $TET_ROOT/xtest/src/bin/mc/tmpl
pmake install
```

5.6 Building the blowup utility

The blowup utility is required for examining any incorrect image files generated by the X server during a test run. Instructions for running the blowup program are given in the section entitled "Examining image files".

Build `blowup` and install in the xtest bin directory as follows.

```
cd $TET_ROOT/xtest/src/pixval/blowup
pmake install
```

6. *Building the tests*

6.1 *Building tests using the TET*

The entire X Test Suite can be built by using the build mode of the TCC. In this mode, the build configuration parameters in the file `$TET_ROOT/xtest/tetbuild.cfg` are used to build each test set in the X Test Suite separately.

```
cd $TET_ROOT/xtest
tcc -b [ -s scenario_file ]  [ -j journal_file ] [ -y string ] xtest all
```

`-b`
: This invokes the TCC in build mode. (If you have just finished building the TCC from the `csh`, you will probably have to `rehash` to get `tcc` in your path.)

`-s scenario_file`
: This option builds the test sets in the named scenario file. The default is a file named `tet_scen` in the directory `$TET_ROOT/xtest`. For more details refer to the section entitled "Building modified scenarios using the TET".

`-j journal_file`
: This option sends the output of the build to the named journal file. The default is a file named `journal` in a newly created sub-directory of `$TET_ROOT/xtest/results`. Sub-directories are created with sequential four digit numbers, with the TCC flags (in this case "b") appended. The TCC will exit if the specified journal file already exists, thus the journal file should be renamed or removed before attempting to execute the TCC.

`-y string`
: This option only builds tests which include the specified string in the scenario file line. This may be used to build specific sections or individual test sets.

`xtest`
: This is the name of the test suite. It determines the directory under $TET_ROOT where the test suite is to be found.

`all`
: This is the scenario name in the default scenario file `$TET_ROOT/xtest/tet_scen`. For more details refer to the section entitled "Building modified scenarios using the TET"

This will execute the TET build tool in the TET configuration variable TET_BUILD_TOOL (which is normally pmake), in each test set directory of the X Test Suite.

The journal file should be examined to verify that the build process succeeded. The report writer `rpt` cannot interpret the contents of a journal file produced during the build process.

Note: If the TCC terminates due to receipt of a signal which cannot be caught, the TCC may leave lock files in the test source directories. Subsequent attempts to restart the TCC may give error messages saying that a lock file was encountered. At this point TCC may suspend the build. It may be necessary to find and remove files or directories named `tet_lock` before continuing.

6.1.1 Signal handling in the TET

An interrupt signal (caused for example by typing the system interrupt character on the controlling terminal) will cause the TCC to abort the currently executing test case. The journal file output records the fact that the test case was interrupted.

Any other signal which can be caught by the TCC causes it to terminate. By default, the system suspend character will also cause the TCC to terminate. If you wish to be able to suspend the TCC, you can add the relevant signals to the parameter SIG_LEAVE in the Makefile for the TCC. Signals in this list will not be caught, but will cause their default action. This is explained further in the Test Environment Tookit Release Notes.

6.2 Building, executing and cleaning tests using the TET

Each test in the X Test Suite may be built, executed and cleaned before the next test set in the scenario. This mode of use has the advantage that the entire X Test Suite may be executed, without necessarily building all the test sets in advance. This mode of use has the disadvantage that you will need to rebuild a test set before rerunning, which will take considerably longer than when it is built in advance.

To do this, skip to the section entitled "Executing the X Test Suite", and refer to the instructions in the sub-section entitled "Building, executing and cleaning tests using the TET"

6.3 Building modified scenarios using the TET

6.3.1 Format of the scenario file

The TET uses a scenario file to determine which test sets to build. The file `$TET_ROOT/xtest/tet_scen` is the default scenario file. The format is basically a scenario name starting in column one, followed by list of test sets to be built (each starting beyond column one). Only one scenario named "all" is provided in the default scenario file.

The names of the test sets are given relative to the directory $TET_ROOT/xtest, and must commence with a leading slash.

6.3.2 Modifying the scenario file

The file `$TET_ROOT/xtest/tet_scen` may be modified by removing lines corresponding to test sets which are not wanted. These will then simply not be built by the TCC. Alternatively, unwanted lines may be commented out by placing # in column one of a line.

It is recommended that the supplied scenario file should be saved if it is modified.

6.3.3 Creating new scenario files

A new scenario file may be created in the directory `$TET_ROOT/xtest`. The TCC will use this scenario file instead of the file `$TET_ROOT/xtest/tet_scen` if it is passed via the `-s` option. For example

```
cd $TET_ROOT/xtest
tcc -b -s scenario_file [ -j journal_file ] [ -y string ] xtest all
```

6.4 Building tests without using the TET

See section 11, entitled "Building, executing and reporting tests without using the TET".

6.5 Building tests in space-saving format

It is possible to build the tests in the X Test Suite such that all the executable files in one section are links to a single executable file. This normally allows a considerable reduction in the disc space requirements for the X Test Suite when fully built.

Note that the names of the files built in space-saving format are different to the names of the separate executable files built using the instructions in previous sections. There is nothing to prevent both sets of executables being built (although there is no value in this, and unnecessary disc space will be consumed).

6.5.1 Building tests in space-saving format using the TET

Before reading this section, read the section entitled "Building the tests using the TET". This gives an explanation of the build mode of the TET, and the structure of scenario files.

A scenario named `linkbuild` is provided in a scenario file named `link_scen` in the directory $TET_ROOT/xtest. This enables the TCC to build the space-saving executable files and create all the required links for each test set in each section of the X Test Suite. The -y option allows a particular space-saving executable for a single section to be built.

Execute the command:

```
cd $TET_ROOT/xtest
tcc -b -s link_scen [ -j journal_file ] [ -y string ] xtest linkbuild
```

This command will execute the TET build tool in the TET configuration variable TET_BUILD_TOOL (which is normally pmake), in the top level directory of each section of the test suite.

6.5.2 Building tests in space-saving format without using the TET

This section describes how to build the space-saving executable files for a particular section of the X Test Suite directly without using the TET.

This can be simply done by calling pmake in the required directory. For example, to build all the space-saving executable files for section 5 of the X Test Suite, execute the command:

```
cd $TET_ROOT/xtest/tset/CH05
pmake
```

7. *Executing the X Test Suite*

Once you have built the X Test Suite as described in the previous sections, work through the following sections to execute the tests.

7.1 *Setting up your X server*

The first step is to ensure that the X server to be tested is correctly set up.

7.1.1 *Formal verification testing*

A number of the tests within the X Test Suite can only give reliable results if there is no window manager and no other clients making connections to the X server. Thus, when conducting formal verification tests, there should be no window manager and no other clients connected to the X server.

It is recommended that you close down and restart your X server before a formal verification test run, in order to ensure that results produced are repeatable and are not affected by earlier tests, although this is not strictly necessary.

You should switch off the screen saver if possible before starting formal verification tests. 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.

You should also ensure that access control is disabled for the server under test, so that the test suite can make connections to the server. Also (if the X server allows this) you should 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, you should use it.

7.1.2 *Informal testing and debugging*

Although no guarantee can be made that the tests within the X Test Suite 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.

Using these guidelines in connection with the instructions in section 11, entitled "Building, executing and reporting tests without using the TET", gives a rapid means to investigate the results of particular tests in detail.

1. Set XT_DEBUG_OVERRIDE_REDIRECT=Yes in your 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 section 8 of the X11R4 Xlib specifications.

4. Be prepared to lose the input focus when tests are running and don't forcibly restore it. This mainly affects tests for section 8 of the X11R4 Xlib specifications.

7.2 Execute configuration parameters

The next step is to set up the execution configuration file.

All execution configuration parameters are contained in a configuration file that forms part of the TET. This file should be edited to reflect the configuration of the X server to be tested and the underlying operating system on which Xlib is implemented. The file

```
$TET_ROOT/xtest/tetexec.cfg
```

contains all the parameters that are needed to execute the X Test Suite. The parameters are grouped in eight sections within the configuration file.

Numeric execution parameters may be specified 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.

7.2.1 Configuration parameters defined by the 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` will give you improved performance if you are not attempting parallel execution of the test suite against multiple servers.

```
Eg: TET_EXEC_IN_PLACE=False
```

TET_SAVE_FILES

This indicates which files generated during execution of tests are to be saved for later examination. This line should not be altered.

```
Eg. TET_SAVE_FILES=Err*.err,*.sav
```

7.2.2 Configuration Parameters for the X Test Suite

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

XT_DISPLAY

This should be set to 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.

```
Eg: XT_DISPLAY=:0.0
```

XT_ALT_SCREEN

If the display supports more than one screen, this parameter should be set to 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.

```
Eg: XT_ALT_SCREEN=1
```

XT_FONTPATH

This should be set to a comma separated list that is a valid font path for the X server. It should 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 the section entitled "Compiling and installing the test fonts").

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.

```
Eg: XT_FONTPATH=/usr/lib/X11/fonts/xtest/,/usr/lib/X11/fonts/misc/
```

XT_SPEEDFACTOR

This is a speedfactor which should be set to reflect the relative delay in response of the underlying operating system and X server combined. Co-operating processes which must synchronize allow a time delay in proportion to this speedfactor, to account for scheduling delays in the underlying operating system and X server. This should be 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.

```
Eg: XT_SPEEDFACTOR=5
```

XT_RESET_DELAY

Specifies a delay time in seconds. Set this to be a time which is greater than or equal to the maximum time required by your server to reset when the last client is closed. The test suite pauses for this time whenever a connection is about to be opened and the server may be resetting. The server may be resetting when the test case is entered (in startup()) as a result of closing the last connection in the previous test case. The server also resets in a few places in the test for XCloseDisplay().

```
Eg. XT_RESET_DELAY=1
```

XT_EXTENSIONS

Specifies whether you wish to test the extended assertions which require the XTEST extension. Set this to Yes if the XTEST extension is available on your system, and you have configured the test suite to use the XTEST extension, and you want to execute these tests, otherwise set to No.

```
Eg. XT_EXTENSIONS=No
```

7.2.3 Configuration parameters for specific tests

The following parameters are used to control one or more specific test purposes in the X Test Suite. These should be set to appropriate values for the X server to be tested.

These parameters may cause temporary changes in the settings of the X server under test (such as the font path). Settings are restored on completion of the specific test purposes.

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 should 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.

```
Eg. XT_VISUAL_CLASSES=StaticGray(8) GrayScale(8) StaticColor(8)
                      PseudoColor(8) TrueColor(8) DirectColor(8)
```

(This must be typed as one line.)

XT_FONTCURSOR_GOOD

This specifies the number of a glyph in the default cursor font known to exist. XT_FONTCURSOR_GOOD+2 should also be a glyph in the default cursor font. Neither of these should be the same as the X server's default cursor.

```
Eg: XT_FONTCURSOR_GOOD=2
```

XT_FONTCURSOR_BAD

This specifies the number of a glyph in the default cursor font known not to exist. If no such value exists, set to UNSUPPORTED.

```
Eg: XT_FONTCURSOR_BAD=9999
```

XT_FONTPATH_GOOD

This should be set to a comma separated list that is a valid font path for the X server. It should be different from XT_FONTPATH. It need not contain the test fonts.

```
Eg: XT_FONTPATH_GOOD=/usr/lib/X11/fonts/100dpi/,/usr/lib/X11/fonts/75dpi/
```

XT_FONTPATH_BAD

This should be set to a comma separated list that is an invalid font path for the X server. If you cannot determine a suitable value, set to UNSUPPORTED. There is no default value - by default, tests which use this parameter will be reported as UNSUPPORTED.

```
Eg: XT_FONTPATH_BAD=/jfkdsjfksl
```

XT_BAD_FONT_NAME

This should be set to a non-existent font name.

```
XT_BAD_FONT_NAME=non-existent-font-name
```

XT_GOOD_COLORNAME

This should be set to the name of a colour which exists in the colour database for the X server.

```
Eg: XT_GOOD_COLORNAME=red
```

XT_BAD_COLORNAME

This should be set to the name of a colour which does not exist in the colour database for the X server.

```
Eg: XT_BAD_COLORNAME=nosuchcolour
```

XT_DISPLAYMOTIONBUFFERSIZE

This should be set to 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.

```
Eg: XT_DISPLAYMOTIONBUFFERSIZE=256
```

7.2.4 Configuration parameters for Display functions

The following parameters are used to control one or more test purposes for Xlib Display functions which are in section 2 of the X11R4 Xlib specifications. These 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.

Settings to these parameters will not cause any change in the settings of the X server under test.

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

XT_SCREEN_COUNT

This parameter should be set to the number of screens available on the display as returned by XScreenCount.

```
Eg: 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.

```
Eg: XT_PIXMAP_DEPTHS=1 8
```

XT_BLACK_PIXEL

This parameter should be set to the black pixel value of the specified screen of the display.

```
Eg: XT_BLACK_PIXEL=1
```

XT_WHITE_PIXEL

This parameter should be set to the white pixel value of the specified screen of the display.

```
Eg: XT_WHITE_PIXEL=0
```

XT_HEIGHT_MM

This parameter should be set to the height in millimeters of the specified screen of the display.

```
Eg: XT_HEIGHT_MM=254
```

XT_WIDTH_MM

This parameter should be set to the width in millimeters of the specified screen of the display.

```
Eg: XT_WIDTH_MM=325
```

XT_PROTOCOL_VERSION

This should be set to the major version number (11) of the X protocol as returned by XProtocolVersion.

```
Eg. XT_PROTOCOL_VERSION=11
```

XT_PROTOCOL_REVISION

This should be set to the minor protocol revision number as returned by XProtocolRevision.

```
Eg. XT_PROTOCOL_REVISION=0
```

XT_SERVER_VENDOR

This should be set to the X server vendor string as returned by XServerVendor.

```
Eg: XT_SERVER_VENDOR=MIT X Consortium
```

XT_VENDOR_RELEASE

This should be set to the X server vendor's release number as returned by XVendorRelease.

```
Eg. XT_VENDOR_RELEASE=5000
```

XT_DOES_SAVE_UNDERS

Set this to Yes if the specified screen of the display supports save unders (indicated by XDoesSaveUnders returning True) otherwise set to No.

```
Eg. XT_DOES_SAVE_UNDERS=Yes
```

XT_DOES_BACKING_STORE

Set this to the following value:
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.

```
Eg. XT_DOES_BACKING_STORE=2
```

7.2.5 Configuration parameters for connection tests

The following parameters are used to control one or more test purposes for XOpenDisplay, XCloseDisplay and XConnectionNumber. These should be set to match the display specified in the XT_DISPLAY parameter and the characteristics of the underlying operating system.

Settings to these parameters will not cause any change in the settings of the X server under test.

These parameters are not used when making connections to the X server in other tests.

XT_POSIX_SYSTEM
: This may be 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.

```
Eg. 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.

```
Eg. XT_DECNET=No
```

XT_TCP
: Set this 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.

```
Eg. XT_TCP=Yes
```

XT_DISPLAYHOST
: Set this to 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).

```
Eg. XT_DISPLAYHOST=xdisplay.lcs.mit.edu
```

XT_LOCAL
: Set this 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 the X Test Suite. When a hostname is omitted, the Xlib implementation of XOpenDisplay can use the fastest available transport mechanism to make local connections.

```
Eg. XT_LOCAL=No
```

7.2.6 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 behaviour 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" .

```
Eg: XT_SAVE_SERVER_IMAGE=Yes
```

XT_OPTION_NO_CHECK

This may be set to Yes to suppress the journal file records containing CHECK keywords. Refer to appendix D for information on the contents of these messages.

```
Eg: XT_OPTION_NO_CHECK=Yes
```

XT_OPTION_NO_TRACE

This may be set to Yes to suppress the journal file records containing TRACE keywords. Refer to appendix D for information on the contents of these messages.

```
Eg: XT_OPTION_NO_TRACE=Yes
```

7.2.7 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 behaviour 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 you should not do this when running large numbers of test sets).

```
Eg: 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 should not be set to `Yes` for verification tests.

```
Eg: 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 should not be set to `Yes` for verification tests.

```
Eg: 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 should not be set to `Yes` for verification tests.

If XT_DEBUG_WINDOW_ONLY is also set to `Yes`, some tests will report

UNRESOLVED due to the fact that nothing has been tested.

```
Eg: 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 set. This should not be set to `Yes` for verification tests.

If XT_DEBUG_PIXMAP_ONLY is also set to `Yes`, some tests will report UNRESOLVED due to the fact that nothing has been tested.

```
Eg: 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 set. This should not be set to `Yes` for verification tests.

Note that the first visual returned by XGetVisualInfo may not be the default visual for the screen.

```
Eg: 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 ID's listed. Note that visual ID's 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 ID's should be entered in decimal, octal or hexadecimal and separated with commas and with no intervening spaces. This should not be set to a non-empty string for verification tests.

```
Eg. 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 the X Test Suite (this is described further in the section entitled "Examining Image Files"). This should not be set to `Yes` for verification tests.

```
Eg: 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 should always be set to NATIVE when the build configuration parameter XP_OPEN_DIS was set to XlibOpaque.c

```
Eg: 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.

```
Eg: XT_DEBUG_VISUAL_CHECK=5
```

7.2.8 Configuration Parameters used only during test development

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 development environment at UniSoft when generating known good image files.

XT_FONTDIR

The directory in which the xtest 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/xtest/fonts/`.

```
Eg: XT_FONTDIR=/usr/mit/testsuite/xtest/fonts/
```

7.3 Executing tests using the TET

The X Test Suite is executed by invoking the execute mode of the Test Case Controller.

```
cd $TET_ROOT/xtest
tcc -e [ -s scenario_file ] [ -j journal_file ] [ -x config_file ]
                                                [ -y string ] xtest all
```

`-e`

This invokes the TCC in execute mode.

`-s scenario_file`

This option executes the test sets in the named scenario file. The default is a file named `tet_scen` in the directory `$TET_ROOT/xtest`. For more details refer to the section entitled "Executing modified scenarios using the TET".

`-j journal_file`

This option sends the test results to the named journal file. The default is a file named `journal` in a newly created sub-directory of `$TET_ROOT/xtest/results`. Sub-directories are created with sequential four digit numbers, with the TCC flags (in this case "e") appended. The TCC will exit if the specified journal file already exists, thus the journal file should be renamed or removed before attempting to execute the TCC.

`-x config_file`

This is an option to run the test suite using the information in a modified execution configuration file named `config_file`. The default is `tetexec.cfg`.

`-y string`
: This option only executes tests which include the specified string in the scenario file line. This may be used to execute specific sections or individual test sets.

`xtest`
: This is the name of the test suite. It determines the directory under $TET_ROOT where the test suite is to be found.

`all`
: This is the scenario name in the default scenario file `$TET_ROOT/xtest/tet_scen`. For more details refer to the section entitled "Executing modified scenarios using the TET".

A journal file will be produced. More information on the contents of the journal file is given in appendix C.

Note: If the TCC terminates due to receipt of a signal which cannot be caught, the TCC may leave lock files in the test source directories. Subsequent attempts to restart the TCC may give error messages saying that a lock file was encountered. At this point TCC may suspend the build. It may be necessary to find and remove files or directories named `tet_lock` before continuing.

7.4 Building, executing and cleaning tests using the TET

Each test in the X Test Suite may be built, executed and cleaned before the next test set in the scenario. This mode of use has the advantage that the entire X Test Suite may be executed without necessarily building all the test sets in advance, thus ensuring disc space is conserved throughout. This mode of use has the disadvantage that you will need to rebuild a test set before rerunning, which will take considerably longer than if it is built in advance.

The X Test Suite is built, executed and cleaned by simultaneously invoking the build, execute and clean modes of the Test Case Controller.

```
cd $TET_ROOT/xtest
tcc -bec [ -s scenario_file ] [ -j journal_file ] [ -x config_file ]
                                                  [ -y string ] xtest all
```

`-b` This invokes the TCC in build mode.

`-e` This invokes the TCC in execute mode.

`-c` This invokes the TCC in clean mode.

The other options are as described in the earlier section entitled "Executing tests using the TET".

A journal file will be produced. This contains for each test set in order the results of the build, followed by the test results, followed by the results of the clean. More information on the contents of the journal file is given in appendix C.

The default journal file is named `journal` in a newly created sub-directory of `$TET_ROOT/xtest/results`. Sub-directories are created with sequential four digit numbers, with the TCC flags (in this case "bec") appended.

7.5 Executing modified scenarios using the TET

7.5.1 Format of the scenario file

The TET uses a scenario file to determine which test sets to execute. The file `$TET_ROOT/xtest/tet_scen` is the default scenario file. The format is basically a scenario name starting in column one, followed by list of test sets to be executed (each starting beyond column one). Only one scenario named "all" is provided in the default scenario file.

The names of the test sets are given relative to the directory $TET_ROOT/xtest, and must commence with a leading slash.

7.5.2 Modifying the scenario file

The file `$TET_ROOT/xtest/tet_scen` may be modified by removing lines corresponding to test sets which are not wanted. These will then simply not be executed by the TCC. Alternatively, unwanted lines may be commented out by placing `#` at the start of the line.

If you wish to execute just a subset of the test purposes in a test set, refer to the section below entitled "Executing individual test purposes using the TET".

It is recommended that the supplied scenario file should be saved if it is modified.

7.5.3 Creating new scenario files

A new scenario file may be created in the directory `$TET_ROOT/xtest`. The TCC will use this scenario file instead of the file `$TET_ROOT/xtest/tet_scen` if it is passed via the `-s` option. For example

```
cd $TET_ROOT/xtest
tcc -e -s scenario_file [ -j journal_file ] [ -x config_file ]
                                        [ -y string ] xtest all
```

7.6 Executing individual test purposes using the TET

Each assertion in the X Test Suite has separate test code which is known as a test purpose. We have arranged that each test purpose is also a separately invocable component, and that the invocable component number is identical to the test purpose number.

The expression within the braces at the end of a line within a scenario file is an invocable component list (or IC_list). The default invocable component list `all` causes the TCC to execute all invocable components in a test set.

By altering the invocable component list for a test set, particular invocable components of interest can be executed.

The invocable component list consists of one or more elements separated by commas. Each element is either an invocable component number, or a range of invocable component numbers separated by a dash.

This is useful for quickly executing a particular test purpose of interest for example:

```
/tset/CH05/stclpmsk/Test{3}
```

This is also useful for executing all test purposes except one known to cause a system error. This may be useful if a particular test purpose causes your X server to exit (at present the TET provides no high level control facilities to conditionally cancel later test sets). For example:

```
/tset/CH05/stclpmsk/Test{1-2,4-6}
```

Note that the placement of windows used by the test suite may differ when an earlier test purpose is not executed. It is intended that test purposes produce the same results regardless of window placement.

7.7 Executing tests without using the TET

See section 11, entitled "Building, executing and reporting tests without using the TET".

7.8 Executing tests in space-saving format using the TET

Before reading this section, read the section entitled "Building tests in space-saving format". When you have built all the sections of the test suite in space-saving format, you can execute all the tests in the test suite using the instructions in this section.

A scenario named `linkexec` is provided in a scenario file named `link_scen` in the directory $TET_ROOT/xtest. This enables the TCC to execute the space-saving executable files which have been built.

Execute the command:

```
cd $TET_ROOT/xtest
tcc -e -s link_scen [ -j journal_file ] [ -x config_file ]
                                        [ -y string ] xtest linkexec
```

8. *Report writer*

A basic report writer `rpt` is included with the X Test Suite. It extracts and formats the main information from a TET journal file produced by executing the TCC in execute mode, or build-execute-clean mode. It does not format the TET journal file produced by the TCC in build only or clean only mode. The main features of the TET journal file produced by the TCC in execute or build-execute-clean mode are described in appendix C.

Execute the report writer as follows:

```
rpt [ -t ] [ -d ] [ -p ] [ -s ] [ -f file ]
```

With only the -f argument, `rpt` lists the results of each test purpose for all test sets that appear in the journal file `file`. The default is the file named `journal` in the highest numbered subdirectory of the `$TET_ROOT/xtest/results` directory that has an 'e' suffix.

The reason for any test result code which is other than PASS is printed out. This is done by copying the test information messages of type REPORT. For further details, see appendix D.

A warning message is printed if a test information message of type REPORT is given in a test purpose which produced a test result code PASS.

The results for each test set are followed by a summary of the number of test purposes in the test set which produced each result code type.

There is no overall summary list of results for all test sets in the journal file.

-t

Test information messages of type TRACE in the test purposes specified are printed. For further details, see appendix D.

-d

Test information messages of type TRACE or DEBUG in the test purposes specified are printed. For further details, see appendix D.

-p

Output is restricted to omit reporting on test purposes that resolve to PASS, UNSUPPORTED, UNTESTED or NOTINUSE — thereby reporting only tests showing possible errors.

-s

The result summaries after the end of each test set are omitted.

9. Examining image files

9.1 Generating pixmap error files

During the test run, discrepancies may be encountered between the image displayed by the server and the known good image. The known good 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`.

9.2 Pixmap error file naming scheme

Each invocation of the TCC creates a sub-directory in `$TET_ROOT/xtest/results`. Sub-directories are created with sequential four digit numbers, with the TCC flags ("e" or "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/CH06/drwln
```

is executed in a scenario file, pixmap error files might be produced in a directory named `$TET_ROOT/xtest/results/0001bec/tset/CH06/drwln`.

The creation of a new results directory tree for each execution of the TCC 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 - when tests are executed without using the TCC the error files are produced in the current directory.

9.3 Known good image file naming scheme

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

More details of the contents of this release are in appendix A.

9.4 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, do not rename the pixmap error or known good image files.

9.4.1 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 colour display, this option will display different pixel values in different colours corresponding to that server's colour table. No attempt is made to preserve colours between different servers.

9.4.2 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.

9.4.3 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.

9.4.4 Information displayed

The value stored in the centre 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.

9.4.5 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.

The reason that we have proposed the two different methods of displaying errors is as follows. One normally has a higher magnification in the Blowup window and the use of a cross (X) through incorrect pixels is good, and simple to remember at this level of zoom. In the Comparison window this style of display does not work well at the lower magnification levels; all the crosses merge to a blur so it is hard to see what type of error is being displayed.

9.4.6 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

```
Bad ----> Good -> Both
 ------<------<----/
```

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 colour/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.

10. *Cleaning the tests*

10.1 *Cleaning tests using the TET*

The entire X Test Suite can be cleaned by using the clean mode of the TCC. In this mode, the clean configuration parameters in the file `$TET_ROOT/xtest/tetclean.cfg` are used to clean each test set in the X Test Suite separately. Previously built test set executables and object files are removed.

```
cd $TET_ROOT/xtest
tcc -c [ -s scenario_file ] [ -j journal_file ] [ -y string ] xtest all
```

`-c`
: This invokes the TCC in clean mode.

`-s scenario_file`
: This option cleans the test sets in the named scenario file. The default is a file named `tet_scen` in the directory `$TET_ROOT/xtest`. For more details refer to the section entitled "Cleaning modified scenarios using the TET".

`-j journal_file`
: This option sends the output of the clean to the named journal file. The default is a file named `journal` in a newly created sub-directory of `$TET_ROOT/xtest/results`. Sub-directories are created with sequential four digit numbers, with the TCC flags (in this case "c") appended. The TCC will exit if the specified journal file already exists, thus the journal file should be renamed or removed before attempting to execute the TCC.

`-y string`
: This option only cleans tests which include the specified string in the scenario file line. This may be used to clean specific sections or individual test sets.

`xtest`
: This is the name of the test suite. It determines the directory under $TET_ROOT where the test suite is to be found.

`all`
: This is the scenario name in the default scenario file `$TET_ROOT/xtest/tet_scen`. For more details refer to the section entitled "Cleaning modified scenarios using the TET".

This will execute the TET clean tool in the TET configuration variable TET_CLEAN_TOOL (which is normally pclean), in each test set directory of the X Test Suite.

The journal file should be examined to verify that the clean process succeeded. The report writer `rpt` cannot interpret the contents of a journal file produced during the clean process.

Note: If the TCC terminates due to receipt of a signal which cannot be caught, the TCC may leave lock files in the test source directories. Subsequent attempts to restart the TCC may give error messages saying that a lock file was encountered. At this point TCC may suspend the build. It may be necessary to find and remove files or directories named `tet_lock` before continuing.

10.2 Cleaning modified scenarios using the TET

10.2.1 Format of the scenario file

Refer to the earlier section "Building modified scenarios using the TET".

10.2.2 Modifying the scenario file

Refer to the earlier section "Building modified scenarios using the TET".

10.2.3 Creating new scenario files

A new scenario file may be created in the directory `$TET_ROOT/xtest`. The TCC will use this scenario file instead of the file `$TET_ROOT/xtest/tet_scen` if it is passed via the `-s` option. For example

```
cd $TET_ROOT/xtest
tcc -c -s scenario_file [ -j journal_file ] [ -y string ] xtest all
```

10.3 Cleaning tests without using the TET

See section 11, entitled "Building, executing and reporting tests without using the TET".

10.4 Cleaning tests built in space-saving format

It is possible to clean the tests in the X Test Suite which were previously built in space-saving format.

10.4.1 Cleaning tests in space-saving format using the TET

A scenario named `linkbuild` is provided in a scenario file named `link_scen` in the directory $TET_ROOT/xtest. This enables the TCC to clean the space-saving executable files and remove all the required links for each test set in each section of the X Test Suite. The -y option allows a particular space-saving executable and its links for a single section to be removed.

Execute the command:

```
cd $TET_ROOT/xtest
tcc -c -s link_scen [ -j journal_file ] [ -y string ] xtest linkbuild
```

This command will execute the TET clean tool in the TET configuration variable TET_CLEAN_TOOL (which is normally pclean), in the top level directory of each section of the test suite.

10.4.2 Cleaning tests in space-saving format without using the TET

This section describes how to clean the space-saving executable files for a particular section of the X Test Suite directly without using the TET.

This can be simply done by calling pclean in the required directory. For example, to clean all the space-saving executable files for section 5 of the X Test Suite, execute the command:

```
cd $TET_ROOT/xtest/tset/CH05
pclean
```

11. Building, executing and reporting tests without using the TET

11.1 Building tests

An individual test set can be rebuilt without the need to use the build mode of the TCC. This is done by executing `pmake` directly, rather than as a TET build tool.

This is a useful facility for building a single test set after a previous build has failed.

The build configuration parameters used by `pmake` are obtained from a file named $TET_BUILDCONFIG, or, if TET_BUILDCONFIG is not set in your environment, from the file named $TET_ROOT/xtest/tetbuild.cfg.

The `pmake` command should be executed in the directory containing the source code for the test set which is to be rebuilt. For more details of the names of the directories containing the source code for the test sets, refer to appendix A.

For example

```
cd $TET_ROOT/xtest/tset/CH05/stclpmsk
pmake
```

No journal file is created when `pmake` is executed directly.

The test set can also be rebuilt using the command

```
pmake Test
```

If there is a macro version of the Xlib function, this may be rebuilt using the command

```
pmake MTest
```

11.2 Executing tests

An individual test set can be executed without the need to use the execute mode of the TCC. This is done by executing a shell script `pt`.

This is a useful facility for executing a single test set repeatedly when investigating a particular test result.

The execution configuration parameters used by `pt` are obtained from a file named $TET_CONFIG, or, if TET_CONFIG is not set in your environment, from the file named $TET_ROOT/xtest/tetexec.cfg.

The `pt` command is a shell script, which attempts to execute the binary file named `Test` in the current directory. If the file `Test` is not found, the `pt` command attempts to execute the space-saving executable file built in that directory.

The `pt` command should be executed in the directory containing the test set which has been built. Unless you have manually installed the test set elsewhere, this will be the directory containing the source code for the test set. For more details of the names of the directories containing the source code for the test sets, refer to appendix A.

For example

```
cd $TET_ROOT/xtest/tset/CH05/stclpmsk
pt
```

A TET results file is created when `pt` is executed directly. This is a file named `tet_xres` located in the directory in which the test was executed.

There are a number of options which may be passed to `pt` which alter the manner in which the test set is executed.

Execute `pt` as follows:

```
pt [ -v XT_VARIABLE_NAME ] [ -d display ] [-i IC_list ] [ -p ] [ -w ]
   [ -P ] [ -D ] [ -x debug_level ] [ -g ] [ -m ]
```

-v XT_VARIABLE_NAME=Value
: Modifies the value of the execution configuration parameter named XT_VARIABLE_NAME, assigning it a value of `Value`.

-d display
: Sets the display string to be used for the test.
The default value is taken from the environment variable DISPLAY, or, if this is not set, from the execution configuration parameter XT_DISPLAY.

-i IC_list
: The invocable components executed will be those specified in IC_list.

Each assertion in the X Test Suite has separate test code, which is known as a test purpose. We have arranged that each test purpose is also a separately invocable component, and that the invocable component number is identical to the test purpose number.

The invocable component list consists of one or more elements separated by commas. Each element is either an invocable component number or a range of invocable component numbers separated by a dash.

This is useful for quickly executing a particular test purpose of interest for example:

```
pt -i 37
```

This is also useful for executing all test purposes except one known to cause a system error. For example:

```
pt -i 1-36,38-57
```

Note that the placement of windows used by the test suite may differ when an earlier test purpose is not executed. It is intended that test purposes produce the same results regardless of window placement.

-p
: This option is equivalent to setting the execution configuration parameter XT_DEBUG_PIXMAP_ONLY to Yes.

-w
: This option is equivalent to setting the execution configuration parameter XT_DEBUG_WINDOW_ONLY to Yes.

-P
: This option is equivalent to setting the execution configuration parameter XT_DEBUG_PAUSE_AFTER to Yes.

-D
: This option is equivalent to setting the execution configuration parameter XT_DEBUG_DEFAULT_DEPTHS to Yes.

-x debug_level
: This option is equivalent to setting the execution configuration parameter XT_DEBUG to `debug_level`.

-g
: The binary file `pvgen` will be executed instead of the binary file `Test`. This option should not be used, since binary files named `pvgen` are only used in the development environment at UniSoft when generating known good image files.

-m
: The binary file `MTest` will be executed instead of the binary file `Test`. Files named `MTest` contain tests for the macro version of an Xlib function.

Note that `pt` creates a temporary file `CONFIG` in the current directory containing the configuration parameters, so write permission is required to this file (or if no file is there, to the current directory).

Note also that the binary file `Test` creates a temporary file `.tmpresfd` in the current directory containing the configuration parameters, so write permission is required to this file.

11.3 Reporting tests

The TET results file produced for an individual test set can be formatted using the basic report writer `rpt`, which is described in more detail in the section entitled "Report writer". The argument `-f tet_xres` formats the contents of the `tet_xres` file.

For convenience, a separate report writer `prp` is provided, which is identical to `rpt`, except that the default file used is `tet_xres` in the current directory.

This is a useful facility for quickly formatting the results from the execution of a test set, and looking at the summary of the result codes for each test purpose executed.

The `prp` command should be executed in the directory containing the TET results file named `tet_xres`. Unless you have manually installed and executed the test set elsewhere, this will be the directory containing the source code for the test set. For more details of the names of the directories containing the source code for the test sets, refer to appendix A.

For example

```
cd $TET_ROOT/xtest/tset/CH05/stclpmsk
prp
```

11.4 Cleaning tests

An individual test set can be cleaned without the need to use the clean mode of the TCC. This is done by executing `pclean` directly, rather than as a TET clean tool.

The clean configuration parameters used by `pclean` are obtained from a file named $TET_CLEANCONFIG, or, if TET_CLEANCONFIG is not set in your environment, from the file named $TET_ROOT/xtest/tetclean.cfg.

The `pclean` command should be executed in the directory containing the test set which was built. For more details of the names of the directories containing the source code for the test sets, refer to appendix A.

For example

```
cd $TET_ROOT/xtest/tset/CH05/stclpmsk
pclean
```

No journal file is created when `pclean` is executed directly.

12. *Appendix A - Contents of X Version 11 Release 6.1*

This section describes the contents of the directories in the `TET_ROOT` directory which are supplied in this release of the X Test Suite. The revised X Test Suite has been developed from the T7 X Test Suite. This section therefore also explains how the arrangement of the revised X Test Suite compares with the T7 X Test Suite.

12.1 *tet*

This contains the source files and include files needed to build the Test Environment Toolkit (TET). The contents of the subdirectories are as follows:

12.1.1 *tet/src*

This contains the source files for the TET.

12.1.2 *tet/inc*

This contains the include files for the TET.

12.1.3 *tet/lib*

This contains the libraries and object files when the TET has been built.

12.1.4 *tet/bin*

This should be empty since the TET utilities will be copied into `$TET_ROOT/xtest/bin`, rather than this directory, when using the modified Makefiles supplied with the TET.

12.1.5 *tet/doc*

This contains the release notes and man pages for the TET.

12.1.6 *tet/demo*

This contains a demonstration program for the TET.

12.2 *xtest*

This contains the tests included in the revised X test suite which are stored as a complete TET test suite. This includes all necessary configuration files and scenario files to enable you to use the TET following the instructions in the documentation.

12.3 *xtest/bin*

This contains commands you will need to install, configure, build and execute the X Test Suite. After installation, this directory contains shell script commands. After configuration and building the X Test Suite, this directory also contains executable programs built for your system.

12.4 *xtest/doc*

This contains the documentation. It contains this user guide, the programmers guide and the release notes. These are supplied in *troff(1)* format requiring the `mm` macro package, and also in PostScript format. It also contains a template for error reports and a description of how to submit them.

It also contains a file `paper.mm`, which is a copy of the file `Xproto_verf/doc/paper.ms` originally supplied in the T7 X Test Suite, converted to use the *mm* macro package. This file contains a paper entitled "An Approach to Testing X Window System Servers at a Protocol Level".

This is a technical paper, which defines in outline terms the areas of the X Window System server which should be tested at the X Protocol level rather than the Xlib level.

The approach recommended in this paper, and adopted in the design of the T7 X Test Suite, has been maintained in the revised X Test Suite. The paper explains the choice of test cases and division of tests between the X Protocol tests and Xlib tests.

Before the revision of the X Test Suite, UniSoft recommended that this paper should be left "as is". As a result, some sections of this paper are out of date in that they refer to development schedules for a previous software development project, which have now been superseded with the production of the revised X Test Suite.

12.5 xtest/fonts

This contains test fonts which should be installed using the instructions in this user guide. It also contains a software library describing the fonts which is used by the tests for text drawing.

12.6 xtest/include

This contains include files for the software in the xtest/src and xtest/tset directories.

12.7 xtest/lib

This contains libraries and other common software which are used by the tests in the xtest/tset directory. The libraries are built using the instructions in this user guide.

12.8 xtest/results

This is an empty directory which is used by the TET to store journal files produced when executing the X Test Suite.

12.9 xtest/src

This contains the source for the libraries and utilities. These are built using the instructions in this user guide.

12.10 xtest/tset

This contains the source for the tests for sections 2 to 10 of the X11R4 Xlib specifications, in directories CH02 to CH10. These are built using the instructions in this user guide. In the rest of this document, these are refered to as the "Xlib tests".

Each of the directories CH02 to CH10 contains further subdirectories which are known as test set directories. Each of these contain all the test code for a single Xlib function. The name of the directory is derived from the name of the Xlib function by a scheme which is described in appendix B.

So, for example for `XSetClipMask` we have the following:

```
tset/CH05/stclpmsk
tset/CH05/stclpmsk/stclpmsk.m
```

The file `stclpmsk.m` is the source file, which is also known as a dot-m file. The format of the dot-m file is described further in the "Programmers Guide".

The Xlib tests are designed to accomplish the following:

1. Test the ability of the Xlib function to behave as specified in the X11R4 Xlib specifications, in situations where no X Protocol error should be generated. This is tested by a series of separate tests known, as "test purposes", each of which is designed to test a single statement in the Xlib specifications. The statement which is tested is contained in an "assertion", which is also contained in the dot-m file, and precedes the test code for that test purpose.
2. Test the ability of the Xlib function to produce the expected X Protocol errors in specified situations. This is tested in further test purposes, each preceded in the dot-m file by an assertion describing the situation which should produce the error.

12.11 xtest/tset/XPROTO

This contains the source for the touch tests for the X Protocol (version 11). These are built using the instructions in this user guide. In the rest of this document, these are refered to as the "X Protocol tests".

These tests were in a separate test suite in the T7 X Test Suite, which was located in a directory Xproto_verf. This included separate documentation, drivers, parameter files, include files and libraries. In the revised X Test Suite, the directory XPROTO only contains the source of the tests - the other items are integrated within the X Test Suite as described in this user guide.

The directory XPROTO contains further subdirectories which are known as test set directories. The structure of test set directories is exactly as for the Xlib tests, described in the previous section.

The X Protocol tests tests are designed to accomplish the following:

1. Test the ability of the X Window System server to accept all legal message types and respond appropriately.
2. Ensure that the server capabilities which Xlib testing depends on work in the simplest cases.
3. Test that the X server adheres to the canonical byte stream of the X Protocol, independent of the host byte sex or compiler structure packing.

For further details of the choice of test cases and division of tests between the X Protocol tests and Xlib tests, refer to the document entitled "An Approach to Testing X Window System Servers at a Protocol Level" contained in file xtest/doc/paper.mm.

13. Appendix B - File naming scheme

A file naming scheme has been devised which is for naming the directories containing dot-m files and the dot-m files themselves.

The file naming scheme converts from an X Window System name to an abbreviated name. This is done as follows:

— Remove leading X.

— Replace:

  Background -> Bg
  Subwindow -> Sbwn
  String -> Str
  Window -> Wdw

— Remove all lowercase vowels aeiou.

— Truncate to 10 chars. We have already checked that truncation to ten characters still gives uniqueness.

— If the string before truncation ended in "16" then force truncated string to end in "16".

— convert to lowercase.

— add ".m" suffix to get name of source file containing C code, assertions and strategies.

14. Appendix C - Format of the TET journal file

This appendix describes the manner in which the X Test Suite uses some of the TET journal file facilities. The format of the TET journal file is not fully described here - only the main features used by the X Test Suite are described. In a future release of the X Test Suite the format is expected to be described fully in the TET documentation.

Journal files are produced by the TCC during the build and execute stages.

The journal file produced during the execute stage contains two basic sections:

1. Details of the configuration parameters and environment in which the tests were executed. This may also be preceded by build configuration parameters and/or followed by clean configuration parameters, if the TCC was invoked in build-execute-clean mode.

2. Details of the test results. This includes the test result codes and test information messages output by the test suite.

Each line in the second section of a journal file is made up from three components separated by a vertical bar:

1. Message type. There is a unique numeric code for each message type which is always the first field on a line.

2. Message parameters. These contain serial number and similar information.

3. Message area. The format of this area is specific to the Message Type.

An example of the second section is as follows:

```
10|53 /tset/CH02/vndrrls/vndrrls 13:41:12|TC Start, scenario ref 85-1, ICs {all}
15|53 1.9 1|TCM Start
520|53 0 7457 1 1|TRACE:NAME: XVendorRelease
400|53 1 1 13:41:14|IC Start
200|53 1 13:41:14|TP Start
520|53 1 7457 1 1|REPORT:XVendorRelease() returned 5000 instead of 1.
220|53 1 1 13:41:14|FAIL
410|53 1 1 13:41:14|IC End
80|53 0 13:41:15|TC End
```

This consists of a block of information for each test set executed which contains the following lines:

1. Message Type 10 - Test Case Start (TC Start)
A single message indicating that the Test Case Controller (the part of the TET which executes test sets) is about to execute a test set. This also indicates the start of the results for this test set (which in TET terminology is known as a test case). This line indicates the name of the test set obtained from the scenario file.

2. Message Type 15 - Test Case Manager Start (TCM Start)
A single message indicating that the Test Case Manager (the part of the TET which calls the test purposes) has started executing.

3. Each assertion in the X Test Suite has separate test code, which is known as a test purpose. We have arranged that each test purpose is also a separately invocable component, and that the invocable component number is identical to the test

purpose number. For each test purpose these lines follow:

1. Message Type 400 - Invocable Component Start (IC Start)
   The second field of the Message Parameters gives the IC number.
2. Message Type 200 - Test Purpose Start (TP Start)
   The second field of the Message Parameters gives the number of the test purpose within the test set.
3. Message Type 520 - Test Case Information
   The second field of the Message Parameters gives the number of the test purpose within the test set.
   The Message Area contains a text message output by the X Test Suite - the possible types of message are described further in appendix D "Interpreting test results" in the section entitled "Test information messages".
4. Message Type 220 - Test Purpose Result
   The second field of the Message Parameters gives the number of the test purpose within the test set.
   The Message Area contains the test result code - the possible test result codes are described further in appendix D "Interpreting test results" in the section entitled "Test result codes".
5. Message Type 410 - Invocable Component End (IC End)
   The second field of the Message Parameters gives the IC number.

4. Message Type 80 - Test Case End (TC End)
   A single message indicating the end of the results for this test set (which in TET terminology is known as a test case).

15. Appendix D - Interpreting test results

This section includes information describing the significance of the test result codes and the accompanying test information messages that may appear in a TET journal file.

15.1 Categorisation of assertions

The test result codes which are output for each test purpose are dependent on the category of the assertion.

The model for the categorisation of assertions which is used in the X Test Suite is described in POSIX.3.

There are four categories of assertions described by POSIX.3 which are designated A, B, C and D.

If the assertion tests a conditional feature, it is categorised as type C or D, otherwise it is categorised as type A or B.

If the assertion is classified as an "extended assertion", it is categorised as type B or D. Otherwise it is categorised as type A or C and is known as a "base assertion".

Tests are always required for base assertions. Tests are not required for extended assertions, but should be provided if possible. There are a number of "extended assertions" for which tests have been written in the X Test Suite. Extended assertions are used to describe features that may be difficult to test conclusively.

	Base Assertion	Extended Assertion
Required Feature	A	B
Conditional Feature	C	D

15.2 Test result codes

The following test result codes may be found within the TET journal file. These will be found in Test Purpose Result lines with Message Type 220 (described in appendix C).

The reason for any result codes NORESULT, UNRESOLVED and UNINITIATED should be determined and resolved.

PASS The implementation passed the test for the corresponding assertion.

FAIL The implementation failed the test for the corresponding assertion.

UNRESOLVED The test for the corresponding assertion commenced, but was unable to establish the required conditions to complete all required stages of the test. The reasons for this should be investigated and if necessary the test rerun.

In some tests, reliance is made on the successful behaviour of another area of the X Window System. Where this reliance is made, and that area of the X Window System 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 you execute the tests using debug options. For example, the message occurs when you execute tests which normally loop over windows and pixmaps and set XT_DEBUG_PIXMAP_ONLY=Yes or XT_DEBUG_WINDOW_ONLY=Yes or XT_DEBUG_DEFAULT_DEPTHS=Yes or XT_DEBUG_VISUAL_IDS=Yes.

NOTINUSE Although there is an assertion within the test set, there is no specific test provided for the assertion. This might be because the assertion is tested adequately as part of the test for another assertion, or because the assertion has been automatically included into a test set in which it is not applicable.

In either case, tests which report the result code NOTINUSE do not need to be investigated further.

UNSUPPORTED This result code may only be used for assertions in category C or D (conditional assertions).

The implementation does not support some optional feature of the X Window System, which is needed to test the corresponding assertion. In this case, the assertion will normally make clear what optional feature is required, and there will be an accompanying test information message describing the feature which was found to be unsupported.

UNTESTED This result code may only be used for assertions in category B or D (extended assertions).

The implementation could not be conclusively tested to determine the truth of the corresponding assertion.

Note that this does not mean that no testing was performed in the X Test Suite. There are a number of "extended assertions" for which we have provided tests where possible (to test some likely problem areas, for example).

UNINITIATED The test for the corresponding assertion could not commence due to a problem in an earlier test purpose or an initialisation routine.

Tests which produce this result code should be resolved as for those reporting UNRESOLVED.

NORESULT Each test purpose should output a test result code before completing. This special result code will be inserted by the TET in the journal file, if the test purpose did not output a test result code. This indicates a major fault during the execution of the test set which should be investigated.

WARNING — The implementation passed the test for the corresponding assertion. Whilst the behaviour of the implementation was found to be acceptable, it behaves in a manner which is not recommended in the X11 specification on which the assertion is based.

FIP — The contents of the journal file must be examined by hand to determine whether the implementation passed the test for the corresponding assertion. This is used for testing functions which produce output whose correctness cannot be easily determined automatically by the test suite.

15.3 Test information messages

There are four types of test information messages which are output by the X Test Suite. Each one results in a TET journal file line with Message Type "Test Case Information" (520), and with the Message Area beginning with one of the following keywords:

REPORT — This keyword is used to report the reason for any test result codes whch are other than PASS. A warning message is printed by the report writer `rpt`, if a test information message of type REPORT is given in a test purpose which produced a test result code PASS.

CHECK — This keyword is used to record the passing of a particular checkpoint in the test suite code. These messages contain the checkpoint number within the test purpose, and the line number within the source code.

These messages are not output to the journal file if the execution configuration parameter XT_OPTION_NO_CHECK is set to Yes. This option can reduce the size of the journal file considerably.

TRACE — This keyword is used for any messages describing the state of the test being executed which are not failure messages.

When running the X Protocol tests, messages with this keyword are output to the journal file, to describe briefly the interaction between the X server and the test program.

These messages are not output to the journal file, if the execution configuration parameter XT_OPTION_NO_TRACE is set to Yes. This option can reduce the size of the journal file considerably.

DEBUG — This keyword is used for debug messages inserted during the development of the X Test Suite.

When running the X Protocol tests, messages with this keyword are output to the journal file, when the debug level is greater than zero, to describe in detail the interaction between the X server and the test program. This includes the contents of requests, replies and events.

This is output if the value of the execution configuration parameter XT_DEBUG is greater than or equal to the level of the debug message. XT_DEBUG may be set from 0 to 3.

16. Appendix E - Outline of Test Environment Toolkit

The "Test Environment Toolkit" is a software tool developed by X/Open[8], UNIX International (UI)[9], and the Open Software Foundation (OSF)[10]. The project which produced this software and the associated interface specifications was also known as Project Phoenix.

The TET consists of a user interface program known as the Test Case Controller (TCC). This enables test software to be built and executed. The TCC uses configuration files to specify the environment for both the build and execute operations. The TCC also uses a scenario file to control which tests to build or execute.

The TCC produces a journal file which is intended to be formated by a test suite specific report writing program.

The TET also includes an Application Programming Interface (API) Part of the API is the Test Case Manager (TCM). This includes a **main()** function which calls user supplied test functions. The API also includes a library of functions to manage the test functions and perform output operations to the journal file in a structured fashion.

Since the developers of the TET have indicated a commitment to develop software test suites that execute within this environment, the TET can be seen as an emerging de facto standard environment for test suite execution.

During stage one of the X Test Suite development project we identified that the TET provides features which are required by the revised test suite.

For this reason we have developed the revised test suite within the TET environment, and supplied a copy of the TET with the revised test suite.

Release 1.9 of the TET was issued by the developers during March 1992, and is included in this release of the X Test Suite. The software is complete in that the functionality is stable and the implementation agrees with the TET specification. Documentation including release notes and `man` pages for the TET utilities, are included in this release of the X Test Suite. However, this release does not contain a Programmers Guide or Users Guide. These are under development by UNIX International, but are not complete at the time of this release, and so are not part of the current version of the TET.

---

8. X/Open is a trademark of the X/Open Company, Ltd. in the U.K. and other countries.

9. UI is a trademark of UNIX International.

10. Open Software Foundation, OSF and OSF/1 are trademarks of the Open Software Foundation, Inc.

17. Appendix F - Glossary

assertion
: A statement of functionality for an *element* of the X Window System, phrased such that it will be **true** for a system conforming to the X Window System specifications. An example would be a *test description* phrased according to the requirements of POSIX.3.

assertion test
: Synonymous with *test purpose*.

base assertion
: An *assertion* for which a test suite must provide an *assertion test*. Every assertion that is not an extended assertion is a base assertion.

element
: A particular X Window System interface such as an Xlib function, header file or X11 Procotol.

extended assertion
: An *assertion* for which a test suite is not required to provide an *assertion test*. An *assertion test* should still be provided if possible.

Reasons why a test suite is not required to provide a test are given in appendix A of the Programmers Guide.

POSIX.1
: Part one of the IEEE POSIX 1003 standards, the document† entitled *System Application Program Interface (API) [C language]*. Also known as P1003.1.

POSIX.3
: Part three of the IEEE POSIX 1003 standards, the document† entitled *Test Methods for Measuring Conformance to POSIX*. Also known as P1003.3.

Project Phoenix
: Synonymous with TET.

SVID
: System V Interface Definition.

TCC
: The Test Case Controller. This is part of the TET. This is a user interface program which enables the test software to be executed. See appendix E for more details.

TCM
: The Test Case Manager. This is part of the TET. This is an object file containing a **main()** function which calls user supplied test functions. See appendix E for more details.

TET
: The "Test Environment Toolkit". This is a software tool which provides a framework within which tests may be developed and executed. More information is given in appendix E.

---

† Obtainable from Publication Sales, IEEE Service Center, P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08854-1331, (201) 981-0060

test case
: Synonymous with *test set*.

test description
: The description of a particular test to be performed on an *element*. This is presented in functional terms and describes precisely what aspect of the X Window System is to be tested for that *element*. An *assertion* is an example of a *test description*, but the reverse is not the case.

test program
: Synonymous with *test set*.

test purpose
: The software which tests the conformance of an implementation of the X Window System to an *assertion*.

test set
: The software containing all the *test purposes* for an *element*.

test strategy
: A description of the design and method used to implement a *test purpose*. This should say how a *test purpose* is implemented rather than what feature is being tested.

XPG
: The X/Open Portability Guide.

Xlib tests
: These are the tests for sections 2 to 10 of the X11R4 Xlib specifications. They are stored in subdirectories of the directories `CH02` to `CH10` (which are to be found in the directory `$TET_ROOT/xtest/tset`).

X Protocol tests
: These are the touch tests for the X Protocol (version 11). They are stored in subdirectories of the directory `XPROTO` (which is to be found in the directory `$TET_ROOT/xtest/tset`).

CONTENTS