VSX Extension for XSH5

VSXgen Specification
VSXGEN-SPEC-1.5

Released: 1st July 1997

This document is produced by UniSoft Ltd. at:

150 Minories
LONDON
EC3N 1LS
United Kingdom

Tel: +44 171 264 2120
Email: `vsx@root.co.uk`

1. Introduction

X/Open's plans for future additions to the VSX test suite call for the test suite to be modified to allow the use of add-on test packages. The libraries and utilities, and files used in the installation procedure, from VSX4.3.6 form the basis for a separate product called VSXgen. The tests from VSX4.3.6 are to be converted into a test package for use with VSXgen.

Any combination of one or more test packages may be used with VSXgen, and the combination installs and runs as if it were a single test-suite with the look and feel of VSX4.3.6.

This document specifies the interfaces between VSXgen and the add-on test packages, and also describes changes in the VSX installation procedure that will affect test package authors. However, it is not a complete specification of VSXgen.

2. Audience

This document is intended to be read by test package developers, and others interested in VSXgen at a similar technical level. A knowledge of VSX4.3.6 internals beyond that of a typical user is assumed.

3. Subsets

The tests in VSX4.3.6 are divided into three subsets, `base`, `dm`, and `xpg2`. Users select which of these subsets they wish to use during the configuration stage.

In VSXgen this subset mechanism is the means by which users select which tests they wish to use from the test package(s) they have installed on the target system. Each add-on test package provides one or more subsets. Information needed for the configuration and installation of tests from the add-on packages is provided to VSXgen via subset-specific files.

The naming of subsets and the division of test sections between subsets is left to the test package author. However, the number of subsets should be kept small (so that the user does not have to type in an excessively long list in order to select all but one of the available subsets), and the names of subsets and test sections must be agreed with X/Open to avoid clashes between packages. Each section must belong to exactly one subset, so there is no potential for name clashes below section level.

A separate subset should be used where part of the configuration or installation procedure makes use of optional system features (or features that might not yet be available on a system which is still under development). Those parts can then be skipped if the subset is not selected. For example, in VSX4.3.6 part of the configuration procedure for the `dm` subset is a check on the contents of the (optional) `<isam.h>` header, and during the installation stage the `isamlib` library is only built if the `dm` subset has been selected. It may also be worth using separate subsets where certain user-supplied information would not be needed if a subset is not selected, particularly if supplying the information involves significant effort for the user. The use of separate subsets for other purposes is discouraged. In particular, users have the ability to build and execute arbitrary groups of tests via the scenario file facility, so there is little point in dividing a test package into multiple subsets purely for the purpose of grouping tests.

4. Directory Structure

When VSXgen and its test packages are installed they form a merged directory structure below the test suite root. This structure is that of VSX4.3.6 with minor modifications. In order to prevent conflicts, and to ensure consistency between test packages, all test package files must be placed in specified locations and be named according to specified naming conventions.[1]

Relevant parts of the directory structure are described in the following subsections.

4.1 Libraries and Utilities

VSXgen contains most of the libraries and utilities from VSX4.3.6. Some are always built during the installation phase, but others are only built if needed by a selected subset. The following library and utility directories under `SRC/common` are always built:

Libraries	`genlib`, `tet_startup`, `vlib`, `vport`;
Utilities	`vprog`, `vtools`.

The following are only built if needed:

Libraries	`tsetlib`;
Utilities	`drivers`, `purefile`, `wchars`.

The `isamlib` library from VSX4.3.6 is not part of VSXgen.

Threads versions of the libraries can also be built if needed. These should be used instead of the normal versions in programs that are linked with the TETware thread-safe API. The threads versions have *thr* prefixed to directory names, except for the threads version of `tet_startup`, which is named `thrstartup`.
Important: Although these libraries are compiled with threads compiler options, this does not necessarily mean they contain thread-safe functions. Test package authors should ensure that functions which are not thread-safe are used in a manner which does not compromise thread safety (e.g. by using a mutex to serialize calls).

Test packages can place additional library and utility sources in directories under `SRC/common`, and their manual pages under `MAN/common`. The libraries and utilities built from these sources can be installed under `SRC/LIB`, `SRC/BIN`, `${TET_EXECUTE}/BIN`, or `BIN`; or in a package-specific or subset-specific location. Header files associated with additional libraries can be placed in subdirectories of `SRC/INC`. It is recommended that these headers are used in test sources as follows:

`#include <`*subdir*`/`*hdr_name*`.h>`

rather than compiling with `-I`*test-suite-root*`/SRC/INC/`*subdir*, so that users can locate them more easily.

VSXgen contains all of the header files in `SRC/INC` from VSX4.3.6 except for `buildisam1.h` and `isamlib.h`.

1. Where existing files in VSX4.3.6 have names which do not follow the specified naming conventions, the equivalent files in the VSX4 test package may use the same names in order to preserve the look and feel of the VSX4.3.6 test suite.

The names of additional directories under `SRC/common`, `MAN/common` and `SRC/INC`, and of files that are installed in the various `LIB` and `BIN` directories, and of externally visible symbols within libraries and headers provided in test packages, must begin with a prefix related to the package or one of its subsets, to prevent clashes. For example, the prefix `isam` could be used for the data-management subset of the VSX4 test package.

Each directory under `SRC/common` must contain a file called `Makefile.org`. This is used by `install.sh` as a template from which it creates the `Makefile` for the directory, by editing the make variables to assign the values specified in `SRC/vsxparams`. The file must contain an `INSTALL` target, which performs the necessary installation tasks.

The header test driver in `SRC/common/drivers` is structured in such a way as to support the building of alternative drivers. This is so that header tests can be run using different compiler options and libraries (instead of VSX_CFLAGS and VSX_LIBS). Test packages which need to use this facility should place a directory under `SRC/common` where the driver will be built. The directory should contain a C source file with the following definitions:

```
char *vsx_cflags_var = "cflags_variable_name";
char *vsx_libs_var = "libs_variable_name";
```

where *cflags_variable_name* and *libs_variable_name* are the names of the config variables to use instead of VSX_CFLAGS and VSX_LIBS respectively. The `Makefile.org` in the directory should compile this source file together with the following list of objects and libraries to build the driver:

```
$(TCM) $(APILIB) ../drivers/drvlib.a $(VSXDIR)/LIB/genlib.a
$(VSXDIR)/LIB/vlib.a $(VSXDIR)/LIB/vport.a $(APILIB) $(SYSLIBS)
```

Note that `$(APILIB)` must appear twice, as shown, in order to resolve references to `tet_startup`, `tet_cleanup` and `tet_testlist` when using TETware.

4.2 Top-level Makefile Targets

The template top-level `Makefile` is constructed by `config.sh` from pieces supplied in VSXgen and pieces supplied in test packages. The pieces related to optional tasks are located in the directory `SRC/install/targets` with file names corresponding to the makefile targets they contain. Which pieces are used may depend on the subsets and the test mode selected by the user. The following VSXgen targets are always used: `privchk`, `mktroot`, `install`, `chmogpriv`. The following VSXgen targets are only used if needed by a selected subset: `dirgid`, `filldisc`. These targets perform the same tasks as in VSX4.3.6.

Targets provided in the test package are executed after the VSXgen targets, in an unspecified order. If there are multiple order-dependent tasks to perform, they should be placed in a single file under `SRC/install/targets`, so that they are called via one overall target name. The contents of this file can be altered during the configuration stage (by a subset-specific shell script) if necessary.

The names of targets provided in test packages must begin with a prefix related to the package or one of its subsets, to prevent clashes.

4.3 User-supplied Interfaces

The template `userintf.c` file is constructed by `config.sh` from pieces supplied in VSXgen and pieces supplied in test packages. Optional pieces are contained in files located in the directory `SRC/install/userintf`. Which of these files are used may depend on the subsets and the test mode selected by the user. The following VSXgen interfaces are always present in `userintf.c`: `setprv()`, `unsetprv()`, `prv_assign()`. The following VSXgen interfaces are supplied in the indicated files under `SRC/install/userintf`, and are added to `userintf.c` if they are needed by a selected subset:

File name	Interfaces
`gen_mnt`	`mnt_rw(), mnt_ro(), unmnt()`
`gen_termios`	`openctl(), openpty(), ptygetattr()`
`gen_newroot`	`newroot()`
`gen_setlimit`	`setlimit()`
`gen_pathres`	`pathdepth()`

These interfaces perform the same tasks as in VSX4.3.6. The other interfaces in the VSX4.3.6 `userintf.c` file are not provided in VSXgen.

The names of files placed under `SRC/install/userintf` by test packages, and the names of file-scope identifiers defined in those files, must begin with a prefix related to the package or one of its subsets, to prevent clashes.

4.4 Testsets

Test packages must organise their testset sources and documentation according to the VSX4.3.6 directory structure and naming conventions:

- sources are located in the directory `tset/`*sect*`/`*area*`/`*tsetdir*
- test assertions, if supplied, are contained in the file `MAN/tset/`*sect*`/`*area*`/`*tsetdir*`/T.`*testset*
- test descriptions are contained in the file `MAN/tset/`*sect*`/`*area*`/`*tsetdir*`/L.`*testset*`_t`
- strategy documentation, if supplied, is contained in the file `MAN/tset/`*sect*`/`*area*`/`*tsetdir*`/L.`*testset*`_s`
- executables are installed in the directory `${TET_EXECUTE}/tset/`*sect*`/`*area*`/`*tsetdir*, and in `${TET_EXECUTE}/tset/`*sect*`/`*area*`/M`*tsetdir* for separate macro tests (where applicable)
- within sections that are used in POSIX modes, testsets that are not needed for POSIX testing have directory names that end in `_X`

- section names end in `.os` if and only if they require automatic installation of separate function and macro tests for each testset.[2]

Further information about testset source and manual conventions appears later in this document.

4.5 Subset Information

Most subset-specific information files are located under `SRC/subsets`. This contains subdirectories with names corresponding to the available subsets. No other files or directories can be placed in `SRC/subsets`, so that listing the directory produces a list of the available subsets.

Each test package provides the following files in `SRC/subsets/`*subset* for each of its subsets. The `config_info`, `config_params` and `config_sub.sh` files are the only ones that must exist when `config.sh` is run. The others can be created if necessary during the configuration stage (by `config_sub.sh`). For example, their contents might vary depending on the test mode that has been selected.

4.5.1 `config_info`

This file has the same format as `SRC/vsxparams` in VSX4.3.6, with information presented as a series of shell variable assignments. The following variables are mandatory:

CONFIG_PACKAGE

The name and release number of the test package to which the subset belongs. E.g. for the `base` subset:

```
CONFIG_PACKAGE="VSX4.4.1"
```

CONFIG_MODES

A space-separated list of the test modes in which the subset can be used. Valid modes are POSIX90, POSIX96, FIPS, XPG3, XPG4, UNIX95 and UNIX98. The FIPS, XPG3 and XPG4 modes are the same as in VSX4.3.6, and POSIX90 mode is equivalent to POSIX mode in VSX4.3.6. The POSIX96, UNIX95 and UNIX98 modes are new.

E.g. for the `base` subset:

```
CONFIG_MODES="POSIX90 POSIX96 FIPS XPG3 XPG4 UNIX98"
```

The following variables are optional (if not set, they are treated the same as an empty value):

CONFIG_NONPOSIX_SECTIONS

A space-separated list of sections that are not needed for testing POSIX. The named sections will be removed by `posixonly.sh`.

E.g. for the `base` subset:

```
CONFIG_NONPOSIX_SECTIONS="XOPEN.cmd XOPEN.hdr XOPEN.os XPG4.hdr XPG4.os lang.C"
```

CONFIG_POSIX_SECTIONS

2. VSX4.3.6 has an exception to this rule for `XPG2.os` but this will not apply in VSXgen.

A space-separated list of sections that are needed for testing POSIX, but which contain additional testsets not needed for testing POSIX. The additional testsets (with names ending in `_X`) will be removed by `posixonly.sh`.

E.g. for the `base` subset:

```
CONFIG_POSIX_SECTIONS="ANSI.hdr ANSI.os POSIX.hdr POSIX.os"
```

4.5.2 `install_info`

This file has the same format as `config_info`. It contains the following variables, which are all optional (if not set, they are treated the same as an empty value):

INSTALL_LIBS

A space-separated list of optional libraries under SRC/common that are needed to build the tests in this subset (and any additional utilities). This includes both libraries provided with VSXgen and libraries provided in the test package.

E.g. for the `dm` subset:

```
INSTALL_LIBS="tsetlib isamlib"
```

(Note: `isamlib` is not provided in VSXgen.)

INSTALL_THRLIBS

A space-separated list of threads libraries under SRC/common that are needed by the subset. All threads libraries are optional, so if it is not empty, this variable will normally contain at least `thrvport` and `thrvlib`, and may also contain `thrgenlib` and `thrtsetlib`. If INSTALL_THRLIBS is not empty, the threads version of `tet_startup` will also be built, and installed as `SRC/LIB/thrstartup.o`.

INSTALL_UTILS

A space-separated list of optional utilities under SRC/common that are needed by the tests in this subset. This includes both utilities provided with VSXgen and utilities provided in the test package.

E.g. for the `base` subset, with XPG4 mode selected:

```
INSTALL_UTILS="drivers purefile wchars"
```

(With XPG3 mode selected, `wchars` would be omitted.)

INSTALL_USERINTF

A space-separated list of files under `SRC/install/userintf` to be included in the `userintf.c` template. The list can specify both files provided with VSXgen and files provided in the test package.

INSTALL_USER_TARGETS

A space-separated list of optional targets to be included in the top-level `Makefile`, in the section that is edited by the user. This can include the `dirgid` and `filldisc` targets provided with VSXgen as well as targets provided in the test package.

INSTALL_FIXED_TARGETS

A space-separated list of targets (provided in the test package) to be included in the top-level `Makefile`, in the section that is not edited by the user.

4.5.3 `config_params`

This is a template file containing additional configuration parameters to be added to `SRC/vsxparams` by `config.sh`. Its contents should be in the same style as the file `SRC/install/params.data/.defaults` in VSX4.3.6. The file is sourced by `config.sh` to obtain default parameter values (if the user does not specify a saved parameter file, and `SRC/vsxparams` does not exist).

To prevent clashes between test packages, the names of the parameters in this file must begin with a prefix related either to the subset or to the package that contains the subset. For example, the prefix `ISAM` could be used for the data-management subset of the VSX4 test package.

4.5.4 `config_sub.sh`

This file contains shell commands which ask the user to enter a value for each parameter in the `config_params` file. The file is sourced from `config.sh` so that it can make use of shell functions defined in `config.sh`, and so that the parameter values obtained from the user are available for `config.sh` to enter into the `SRC/vsxparams` file.

The commands used should have the same form as equivalent commands in `config.sh`. In VSXgen this differs slightly from VSX4.3.6, in that `/tmp/getans`, `/tmp/mywhere` and `$checklibx` are replaced with shell functions. The following example uses the `getans` and `checklib` functions:

```
if test -z "$LIBISAM"
then
        default="none"
else
        default=$LIBISAM
fi
while :
do
        getans "
Which library is your isam library [$default]?
(should be of form: -l<name> or full pathname): "
        case $ans in
        "")
                break
                ;;
        -l*)
                if checklib "$ans"
                then
                        LIBISAM="$ans"
                        break
                else
                        echo "*** Could not find library $ans"
                fi
                ;;
        /*)
                if test -f "$ans"
                then
                        LIBISAM="$ans"
                        break
                else
                        echo "*** Could not find library $ans"
                fi
                ;;
        *)
                echo "*** $ans not of form: -l<name> or full pathname"
                ;;
        esac
done
```

The next example uses the `mywhere` function:

```
if test -z "$CHMOD"
then
        CHMOD="chmod"
fi
if test -z "‘mywhere $CHMOD‘"
then
        while :
        do
                echo "*** Could not find the program $CHMOD in your PATH"
                getans "
Which command changes file modes [$CHMOD]? "
                if test -z "$ans"
                then
                        ans="$CHMOD"
                fi
                if test ! -f "$ans"
                then
                        if test -z "‘mywhere $ans‘"
                        then
                                echo "*** Could not find $ans"
                        else
                                CHMOD="‘mywhere $ans‘"
                                break
                        fi
                else
                        CHMOD="$ans"
                        break
                fi
        done
fi
```

The `config_sub.sh` file is sourced by `config.sh` before it asks for values for any optional VSXgen parameters, but after it has asked for the other VSXgen parameters. Values for the optional parameters are only requested from the user if one or more subsets have indicated that a given parameter is needed by setting a corresponding shell variable to `Y` in the `config_sub.sh` script. The name of the shell variable is formed by prefixing ASK_ to the parameter name. For example, the CFPURE parameter is needed by the `base` subset if a non-POSIX mode has been selected, so the `config_sub.sh` file for the `base` subset could indicate this as follows:

```
case $TEST_MODE in
POSIX*|FIPS*) ;;
*) ASK_CFPURE=Y ;;
esac
```

Further details on VSXgen configuration parameters may be found in the section entitled ‘Installation Differences’.

Additional tasks can be performed by `config_sub.sh` if necessary. For example, it can create files under `SRC/subsets` or `SRC/install/targets` that have mode-dependent contents.

4.5.5 `Inc.`*mode*

One of these files must be provided for each mode listed in the CONFIG_MODES variable in `config_info`. The files have the same format as `SRC/install/base/Inc.list` in VSX4.3.6. They provide search patterns for the header symbols that are needed when building the subset’s testsets in each mode. Normally the files include all of the symbols defined by the specification (or part of it) tested by the subset, rather than just those used in the testset sources.

The files are used to drive the header checks performed by `config.sh`. The `Inc.`*mode* files for the selected subsets, together with a list of the symbols in POSIX.1-1990, are merged to produce the list used by `config.sh`. Therefore the files for each subset need not include symbols that are in POSIX.1-1990. (These are the symbols listed in `SRC/install/base/Inc.list` in VSX4.3.6.) However, XPG3 mode is treated as a special case, since XPG3 predates POSIX.1-1990. The list of symbols in POSIX.1-1990 is not used if XPG3 mode is selected, so `Inc.XPG3` files must contain all the symbols to be checked.

The headers `<stdarg.h>` and `<varargs.h>` are another special case. Since only one of the two needs to exist, they are not checked in the same way as the other headers. Instead, a special check is done by `install.sh`, which verifies not only the content but also the correct working of the appropriate header.

The files are used in conjunction with files under `SRC/install/incrpt.data` and `SRC/install/check.data` which are part of VSXgen. These only provide support for the headers and symbols in XSH5. If any headers or symbols not specified by XSH5 appear in an `Inc.`*mode* file, this may result in error messages during the header checking procedure and/or spurious entries in the `SRC/vsxconfig.h` file.

Where a symbol has multiple uses, the search patterns must be carefully designed to distinguish between them. For example, `stat` in POSIX.1 is both a function and a structure tag. The search patterns used for these in VSX4.3.6 are:

```
stat[ ˆI]*([ˆ)]*
struct[ ˆI]*stat
```

where `ˆI` represents a TAB character.

When the patterns are used, they are combined with additional delimiters to avoid problems with symbols that are substrings of other symbols. The additional delimiters are: beginning of line or `[ ˆI,*(]` for the front of the pattern; end of line or `[ ˆI,;()[{]` for the end of the pattern. (Again, `ˆI` represents a TAB.) Notice how this affects the pattern for the `stat` function above. Without the `[ˆ)]*` on the end, the part of the line after the pattern match might well be `const char *`, which would not match the end delimiter.

4.5.6 `install_sub.sh`

This is a shell script to be executed from `install.sh`. It can be used to perform tasks such as appending definitions of new symbols to `SRC/INC/std.h` corresponding to parameters in `config_params` which have compile-time effect. The names of any symbols added to `SRC/INC/std.h` must begin with a prefix related to the package or one of its subsets, to prevent clashes.

Since this script is executed rather than sourced, it must have execute permission for user `vsx0`. The script is executed by `install.sh` after it has created the `Makefile` in each directory under `SRC/common` but before it executes *make* in those directories. The environment passed to the script includes the following parameters from `SRC/vsxparams`: CC, COPTS, DEFINES, INCDIRS, LDFLAGS, PATH, SUBSETS, SYSLIBS, TEST_MODE, TET_EXECUTE, THR_COPTS, VSXDIR, and all of the parameters from `config_params`, plus the following TET-related variables (as used in makefiles): TETINC, APILIB, TCM, TCMCHILD, THRAPILIB, THRTCM, THRTCMCHILD.

4.5.7 `chmog_privs`

All testset executables that need to be assigned privileges by `chmog` are listed in this file together with the set of privileges they require. The section name that each testset belongs to is also included, to distinguish between multiple executables with the same name. For example:

```
{ "POSIX.os", "mkdir_su", PRV_MOUNT, PRV_SETID, PRV_ACCESS },
```

Each privilege that the executable may request via a `setprv()` call must be included here, otherwise the `setprv()` call may fail on systems which do not have a *super-user* privilege paradigm.

4.5.8 `scen.bld` and/or *mode*`.bld`

One of these files from each subset is used by `install.sh` to create the combined build scenario file for use with `tcc`. If a *mode*`.bld` file corresponding to the selected test mode exists it will be used, otherwise `scen.bld` will be used.

The format of these files is as follows:

```
all
        /tset/sect/area/tsetdir/T.testset
        .
        .
```

In the case of testsets with shared source (such as `printf/fprintf/sprintf`) only one scenario line is included, with the final *testset* name usually chosen to match the testset directory name *tsetdir*.

4.5.9 `scen.exec` and/or *mode*`.exec`

One of these files from each subset is used by `install.sh` to create the combined execution scenario file for use with `tcc`. If a *mode*`.exec` file corresponding to the selected test mode exists it will be used, otherwise `scen.exec` will be used.

The format of these files is as follows:

```
all
        "total tests in sect1 999"
        /tset/sect1/area/tsetdir/T.testset
        .
        .
        "total tests in sect2 999"
        /tset/sect2/area/tsetdir/T.testset
        .
        .
```

In the case of testsets with shared source, a separate scenario line is included for each testset. Also, unlike the build scenario files, these files include separate lines for function and macro tests where applicable.

The test total figures for each section are used by `vrpt` in producing the conformance summary tables in the test report.

4.5.10 `exec_params`

This is a template file containing additional configuration parameters to be added to `tetexec.cfg` by `install.sh`. Its contents should be in the same style as the optional sections in the file `SRC/install/tetexec.cfg` in VSX4.3.6, including a suitable section heading. For example:

```
#  ISAM Library Characteristics - required for 'dm' sub-set
```

To prevent clashes between test packages, the names of the parameters in this file must begin with a prefix related either to the subset or to the package that contains the subset. For example, the prefix `ISAM` could be used for the data-management subset of the VSX4 test package.

4.6 User-modifiable Scripts

Test packages can place shell scripts in `TESTROOT/BIN` that are to be edited by the user. For example, the `base` subset uses such scripts to create and extract *tar* and *cpio* format archives using implementation-dependent commands. Script names should end in `.sh`, so that `install.sh` will make them writable, and should begin with a prefix related to the package or one of its subsets, to prevent clashes.

4.7 Release Identification File |

Each test package must include a *release identification file* in the test suite root directory. The name of the file is of the form *pkg*`rel`*num*, where *pkg* is the package name (e.g. `VSX4`) and *num* is the release number of the package. |

The file should be distributed with the test package source files in such a way that it is the last file to be extracted when the sources are unpacked. This is so that its presence shows that all of the sources have been unpacked, as indicated in the VSX user guide. |

4.8 Additional Files

The recommended location for test packages to place any additional files needed for purposes not covered by this specification, is anywhere below one of the `SRC/subsets/`*subset* directories. If this is not suitable then other locations can be used, within reason and in keeping with the aim of providing users with as much consistency between packages as possible. Any such locations must have names which begin with a prefix related to the package or one of its subsets, to prevent clashes.

5. Testset Source Conventions

Various aspects of the testset building stage depend on certain conventions in the testset sources and associated files. These conventions are mostly related to the actions of the script `vmake.sh`, which is executed from the VSXgen build tool, `vbuild`.

5.1 Shared Source Directories

VSXgen supports the installation of multiple testsets from shared sources. This feature can be used when implementing tests for closely related functions, such as `printf()`, `fprintf()`, and `sprintf()`. Testsets installed from shared sources have the same testset directory component in their pathnames, but different testset names. For example:

```
ANSI.os/streamio/printf/T.fprintf
ANSI.os/streamio/printf/T.printf
ANSI.os/streamio/printf/T.sprintf
```

The executables are installed as hard links to each other by a single `chmog` command in the testset `Makefile`.

Shared source directories must contain a file called `tset_list` which contains a list of testset names, one per line. For example:

```
fprintf
printf
sprintf
```

5.2 Conventions Related to Macro Tests

In order to provide automatic detection and installation of separate macro tests in cases where an interface is implemented as both a function and a macro, testset sources in sections with names ending `.os` must follow a number of conventions.

The source directory must contain a C source file which has the same name as the directory (with `.c` appended), and which contains:

- `#include` lines for the system headers that are used to obtain declarations of the interfaces tested by the testset (even if the interfaces are not used in the file);
- preprocessor directives to undefine any macro versions of these interfaces when compiled with `-DUNDEF_MACROS`, for example:

  ```
  #ifdef UNDEF_MACROS
  #undef hsearch
  #undef hcreate
  #undef hdestroy
  #endif
  ```

  (these three interfaces are all tested by the `hsearch` testset in the VSX4 test package); or, in the case of interfaces that are allowed to be implemented only as macros, such as `assert()` and `setjmp()`, the following lines:

```
#ifdef UNDEF_MACROS
#undef UNDEF_MACROS
#endif
```

- a line of the form:

  ```
  #define NO_TESTS    99
  ```

  which is used, when no macros are detected, to create a dummy macro testset that will produce the correct number of NOTINUSE results.

The `#include` and `#undef` lines (except `#undef UNDEF_MACROS`) must also appear in all other testset source files which use the interfaces. The `#undef` lines and their surrounding `#ifdef` and `#endif` must appear exactly as shown above, as they are examined by `vmake.sh` to identify the names of the interfaces being tested. Source files should not include any system headers after the `#undef` lines.

5.3 Testset Makefiles

The `Makefile` in the testset directory must contain the following targets:

`INSTALL`	creates all of the files needed for execution of the testset (by compiling the testset sources, or running the `hdrdefs` and `hdranal` tools in the case of header tests) and installs them in the directory specified by the INSTDIR variable, using `chmog` to ensure the owner, group, mode and any necessary privileges are assigned correctly.
`CLEAN`	removes files from the testset source directory that are produced by the `INSTALL` commands (or might be left behind on failure), such as object files and local copies of the executables.
`CLOBBER`	performs a `CLEAN` and then removes the files from `$(INSTDIR)` that were placed there by `INSTALL`.

Normally the default target is `all`, which builds local copies of the testset executables but does not install them under the testroot.

The `Makefile` can use whichever of the following *make* variables are appropriate: TETINC, APILIB, TCM, TCMCHILD, AR, CC, COPTS, DEFINES, INCDIRS, LDFLAGS, LORDER, MLIB, RANLIB, SYSLIBS, TET_EXECUTE, THRAPILIB, THRTCM, THRTCMCHILD, THR_COPTS, TSORT, VSXDIR, and any variables corresponding to additional subset-specific configuration parameters (listed in the `config_params` file of a selected subset). These are set on the *make* command line by `vmake.sh`, and so their configured values will override any default values set in the `Makefile`. The value of INSTDIR is set on the *make* command line when installing macro versions of tests in `.os` sections, but the value in the `Makefile` is used when installing the function version.

5.4 Header Test Driver

Header tests all use the same testset executable `${TET_EXECUTE}/BIN/driver.hdr`. Alternative versions of the driver can be built - see the section entitled ‘Libraries and Utilities.’ If an alternative is used, substitute its name for `driver.hdr` in the following discussion.

In order to reduce the disk space required, the installed testset executables are hard links to copies of the driver placed in the parent directories (i.e. one per test *area*). The `Makefile` for each header testset copies `$(TET_EXECUTE)/BIN/driver.hdr` to

`$(INSTDIR)/../driver.hdr` if the latter does not already exist, and then creates the testset executable as a link to this file. The name of the testset executable must begin with a `T.` prefix.

The header tests themselves consist of C source files placed into an archive using the `cs_arc` tool, where the name of the archive is the same as the testset executable with the initial `T` replaced by an `L`. The source files have names which are meaningful to the header driver. Examples of the names used in VSX header tests are:

`cc01cs.c`	test 1 source file to be compiled only;
`cc02es.c`	test 2 source file to be compiled, linked, and executed;
`cc04e0.c, cc04e1.c`	test 4 source files to be compiled, linked together, and executed;
`cc22ew.c`	test 22 source file to be compiled, linked and executed, with warning result if the compile fails.

The test results produced by the header driver are based on the compiler exit status for compile-only tests, and on the execution exit status for compile-and-execute tests. Symbolic constants for use in `exit()` calls in execution tests are defined in `SRC/INC/HEADER.h`.

A mechanism to allow header tests to access parameter values from `tetexec.cfg` is provided by the header driver. Parameters listed in the file `SRC/common/drivers/testvars.h` are copied into the environment passed to executed tests, and the values can therefore be obtained using `getenv()`. The file `testvars.h` is created by `install.sh` and lists the parameters VSX_CHRDEV_FILE and VSX_BLKDEV_FILE by default. Additional parameter names can be appended to the file by the `install_sub.sh` scripts of subsets that need access to more parameter values in their header tests.

The header driver has a list of feature test macro combinations to be used when compiling header test sources. The contents of the list depends on the test mode selected when the driver was built. Each header test is performed multiple times, once for each feature test macro combination. In each mode except UNIX98 all the valid combinations are used, but in UNIX98 mode only a representative selection is used. (In UNIX98 mode any value of _POSIX_C_SOURCE greater than zero and less than or equal to 199506 could be included. The values used have been chosen because they are likely to be ones that affect the visibility of symbols in headers.)

The feature test macro combinations used in each test mode are:

POSIX90 and FIPS

1. `-D_POSIX_SOURCE`

POSIX96

1. `-D_POSIX_C_SOURCE=199506`
2. `-D_POSIX_C_SOURCE=199506 -D_POSIX_SOURCE`

XPG3

1. `-D_XOPEN_SOURCE`

XPG4

1. `-D_XOPEN_SOURCE`

2. `-D_XOPEN_SOURCE -D_POSIX_SOURCE`
3. `-D_XOPEN_SOURCE -D_POSIX_C_SOURCE=1`
4. `-D_XOPEN_SOURCE -D_POSIX_C_SOURCE=2`
5. `-D_XOPEN_SOURCE -D_POSIX_SOURCE -D_POSIX_C_SOURCE=1`
6. `-D_XOPEN_SOURCE -D_POSIX_SOURCE -D_POSIX_C_SOURCE=2`

UNIX95

1. `-D_XOPEN_SOURCE_EXTENDED -D_XOPEN_SOURCE`
2. `-D_XOPEN_SOURCE_EXTENDED -D_XOPEN_SOURCE -D_POSIX_SOURCE`
3. `-D_XOPEN_SOURCE_EXTENDED -D_XOPEN_SOURCE -D_POSIX_C_SOURCE=1`
4. `-D_XOPEN_SOURCE_EXTENDED -D_XOPEN_SOURCE -D_POSIX_C_SOURCE=2`
5. `-D_XOPEN_SOURCE_EXTENDED -D_XOPEN_SOURCE -D_POSIX_SOURCE -D_POSIX_C_SOURCE=1`
6. `-D_XOPEN_SOURCE_EXTENDED -D_XOPEN_SOURCE -D_POSIX_SOURCE -D_POSIX_C_SOURCE=2`

UNIX98

1. `-D_XOPEN_SOURCE=500`
2. `-D_XOPEN_SOURCE=500 -D_POSIX_SOURCE`
3. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=1`
4. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=1 -D_POSIX_SOURCE`
5. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=2`
6. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=2 -D_POSIX_SOURCE`
7. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=199309`
8. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=199309 -D_POSIX_SOURCE`
9. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=199506`
10. `-D_XOPEN_SOURCE=500 -D_POSIX_C_SOURCE=199506 -D_POSIX_SOURCE`

6. Testset Manual Conventions

Conventions related to the files in the `MAN/tset` hierarchy are described in the following sections.

6.1 Assertions

The test assertions file `T.`*testset* must contain *nroff*/*troff* source using the *man* macros. The local macros such as `TL` and `TI` in VSX4.3.6 assertion files need not be used.

The content after formatting should follow the guidelines in the file `MAN/tset/Intro.man`.

6.2 Test Description and Strategy Files

The text of each assertion forms the 'test description' for each test, contained in the file `L.`*testset*`_t.` This file is created using the `cs_arc` utility (from `SRC/common/vtools`) with archive entries formed by concatenating the testset name and test number.

Descriptions of the strategy employed in each test are contained in the file `L.`*testset*`_s,` which has the same format as `L.`*testset*`_t.`

These files are where the `vrpt` report writer obtains the test descriptions and strategies to place in reports.

7. Documentation

Since several different 'VSX' test suites can be formed by using VSXgen together with different combinations of add-on test packages, the VSX user guide sources are structured in a similar way. VSXgen contains the *troff* sources for the generic parts, and each test package contains additional source files with supplementary information relating to that package. By extracting the relevant sources into a single directory, a user guide covering any combination of test packages can be printed.

Each test package should be distributed with a postscript version of the user guide which has been produced from just the VSXgen sources and the sources for that one test package. A postscript version covering all the test packages is planned to be produced by X/Open.

7.1 User Guide Source Files

The user guide source files are packaged separately from the test sources. To assemble the combined user guide source files, test suite users change directory to an empty directory where they unpack the relevant distribution file from VSXgen and from each test package.

Chapters 5 to 11 inclusive, and appendices B, C and D can each have package-specific parts. These chapters/appendices are currently processed using file names of the form *NN*`*.mm` where *NN* is the chapter number or AB, AC or AD. The generic part of the chapter is contained in the file *NN*`g.`*name*`.mm` and each package-specific part is contained in the file *NN*`p.`*pkg*`.mm` where *pkg* is the name of the test package. Thus the package-specific parts of each chapter appear at the end of the chapter, in alphabetic order of package name.

Each package must also provide a file named `RLp.`*pkg* containing one line identifying the package name, release number, and month and year of release. The line must be enclosed within a `.Xx/.Xe` pair if the package does not support XNFS testing, and within a `.Px/.Pe` pair if it does not support any POSIX or FIPS modes.

7.2 Content of Package-specific Files

The need for package-specific information in a particular chapter can arise in two ways. Firstly, where the generic part of the chapter contains a reference to package-specific information, the indicated information must be provided in the same form (i.e. if the reference is within an Action Points list, the package-specific information must also be within an Action Points list) and within the same version selector macros (`.Xx`, etc.) Secondly, information should be provided as appropriate to document the additional configuration and installation requirements of the package. E.g. questions asked by `config_sub.sh` should be documented in the chapter entitled ‘‘CONFIGURING VSX.’’

8. Installation Differences

The following sections describe the differences between the installation procedures for VSX4.3.6 and for VSXgen-based test suites that will affect test package authors.

8.1 Configuration Stage

When `config.sh` asks which test mode the user requires, only the modes supported by one or more available subsets are offered.

When `config.sh` asks which subsets the user wants to install, only those subsets which support the selected test mode are offered. If only one subset supports the selected mode, the ‘subset’ question is skipped.

VSXgen contributes the parameters listed below to `SRC/vsxparams`. Test packages should not use any other parameters from VSX4.3.6 (except when they are in the subset’s `config_params` file.)

- The following parameters are always set by `config.sh`: AR, CC, CHGRP, CHMOD, CHOWN, COPTS, DEFINES, INCDIRS, LDFLAGS, LONG_DOUBLE_SUPP, LORDER, PATH, RANLIB, SPEED, SUBSETS, SYSLIBS, TEST_MODE, TET_EXECUTE, TSORT, VPRINTF_SUPP, VSXDIR, VSX_OPER, VSX_ORG, VSX_SYS.
- The following parameters are only set by `config.sh` if they are needed by one or more selected subsets: CFPURE, MLIB, NOSPC_DEV, THR_COPTS. Subsets indicate which of these parameters they need by setting the corresponding shell variables ASK_CFPURE, ASK_MLIB, ASK_NOSPC_DEV and ASK_THR_COPTS in their `config_sub.sh` file (described earlier).

The DEFINES parameter is set by `config.sh` according to the selected test mode. The values for each mode are:

Mode	value
POSIX90	`-D_POSIX_SOURCE`
FIPS	`-D_POSIX_SOURCE`
POSIX96	`-D_POSIX_C_SOURCE=199506`
XPG3	`-D_XOPEN_SOURCE`
XPG4	`-D_XOPEN_SOURCE`
UNIX95	`-D_XOPEN_SOURCE -D_XOPEN_SOURCE_EXTENDED`
UNIX98	`-D_XOPEN_SOURCE=500`

Note that `_XOPEN_SOURCE` is defined in all non-POSIX modes, so that test code can use `#ifdef _XOPEN_SOURCE` to tell whether it is being used in one of the X/Open modes or one of the POSIX modes (with FIPS treated as a POSIX mode).

8.2 Installation Stage

In VSX4.3.6 the values of TEST_XPG and TEST_FIPS set in `SRC/INC/std.h` are used to determine, at compile time, the test mode selected by the user. The value of TEST_XPG matches the value of _XOPEN_VERSION defined in `<unistd.h>` for the specification being tested (or is zero for POSIX mode), and TEST_FIPS is a simple boolean used to distinguish between POSIX and FIPS.

In VSXgen this scheme is extended for the new test modes, maintaining the relationship with symbols in `<unistd.h>`, by setting TEST_XPG to 500 in UNIX98 mode, and setting a new symbolic constant, TEST_UNIX, to 1 for UNIX95 and UNIX98 modes, and zero for all other modes.

The full set of correspondences are:

Mode	TEST_FIPS	TEST_XPG	TEST_UNIX
POSIX90	0	0	0
POSIX96	0	0	0
FIPS	1	0	0
XPG3	0	3	0
XPG4	0	4	0
UNIX95	0	4	1
UNIX98	0	500	1

There is no symbol in `SRC/INC/std.h` to distinguish between POSIX90 and POSIX96 modes, since the value of _POSIX_C_SOURCE can be used for this purpose.

When `install.sh` creates the `Makefile` in each subdirectory of `SRC/common` it sets the following variables in the `Makefile`: AR, CC, CFPURE, CHGRP, CHMOD, CHOWN, COPTS, DEFINES, LDFLAGS, LORDER, RANLIB, SYSLIBS, TET_EXECUTE, THR_COPTS, TSORT, and any variables corresponding to additional subset-specific parameters (listed in the `config_params` file of a selected subset) to the value of the corresponding parameters in `SRC/vsxparams`. It also sets TETINC, APILIB, TCM, TCMCHILD, THRAPILIB, THRTCM and THRTCMCHILD in the `Makefile` according to which version of TET is being used. (THRAPILIB, THRTCM and THRTCMCHILD are new in VSXgen: they contain the names of the TETware thread-safe equivalents to APILIB, TCM and TCMCHILD.)

VSXgen has a different location and naming convention from VSX4.3.6 for the subset- and mode-specific files which are combined to produce the scenario files used with `tcc`. As in VSX4.3.6 these files only contain an `all` scenario, used to perform complete runs suitable for inclusion in branding applications. Test packages can include additional separate scenario files for other purposes, such as development testing, in a subset-specific location.

8.3 Building Stage

The VSX4.3.6 `tetbuild.cfg` file contains `Makefile` variables for use when building testsets. In VSXgen only the parameters needed by `vbuild` (or `vrpt`) are contained in this file. The others are obtained from `SRC/vsxparams` instead of `tetbuild.cfg`. Therefore VSXgen has no provision for additional parameters to be added to `tetbuild.cfg`.

The mechanism that installs separate macro and function tests differs from VSX4.3.6.

8.4 Execution Stage

VSXgen contributes the parameters indicated below to `tetexec.cfg`. Test packages should not use any other execution parameters from VSX4.3.6.

- All parameters in sections of the VSX4.3.6 `tetexec.cfg` template with the following section headers:

  ```
  #  General Parameters - required for all sub-sets

  #  Compiler Characteristics - required for all sub-sets

  #  Terminal Interface Parameters - required for 'base' subset

  #  Fixed config variables - these must not be changed
  ```

- VSX_BLKDEV_FILE, VSX_CHRDEV_FILE, VSX_FCNTL_EDEADLK, VSX_FCNTL_MAXLOCK, VSX_INVALID_GID, VSX_INVALID_GNAME, VSX_INVALID_PNAME, VSX_INVALID_UID, VSX_INVALID_WHENCE, VSX_INVAL_SIG, VSX_MOUNT_DEV, VSX_NOSPC_DEV, VSX_READDIR_EBADF, VSX_ROFS, VSX_SIGSET_EINVAL, VSX_SYS_OPEN_MAX, VSX_TTYNAME, VSX_TTYUSER, VSX_ULIMIT_BLKS, VSX_UNLOCKABLE_FILE, VSX_UNUSED_GID, and VSX_UNUSED_UID from the section headed: base sub-set (POSIX and X/Open modes).
- VSX_INVALID_FCNTL_CMD and VSX_PURE_FILE from the section headed: base sub-set (XPG3 and XPG4 modes).

The VSX4.3.6 section headings are given for reference only – the section organisation differs in VSXgen.

8.5 Reporting Stage

In the conformance summary printed by *vrpt*, the table giving the total number of each result code for each test section is not divided into subsets.

9. Library and Utility Differences

The following sections describe the differences between the library and utility sources in VSX4.3.6 and VSXgen that will affect test package authors. These sections only cover differences not already described elsewhere in this document.

9.1 `genlib`

The `errname()` function accepts all of the `errno` values defined in XSH5.

9.2 `tsetlib`

The contents of the global `sig_type[]` vary according to the selected test mode. In UNIX98 mode it contains entries for all of the signal numbers defined in XSH5 (after it has been initialised by the `vsx_sigs()` function).

9.3 `vlib`

The `signame()` function accepts all of the signal numbers defined in XSH5.

The contents of the globals `sig_name[]` and `sig_array[]` vary according to the selected test mode. In UNIX98 mode they contain entries for all of the signals defined in XSH5, indexed using the corresponding symbols in `SRC/INC/vsx_signal.h`.

9.4 `vport`

The default versions of `setprv()`, `unsetprv()` and `prv_assign()` handle the following additional privilege types: `PRV_LIMITS`, `PRV_MEMLOCK`, `PRV_GETRTSCHED`, `PRV_GETTHRSCHED`, `PRV_SETRTSCHED`, `PRV_SETTHRSCHED` and `PRV_SETTIME`. These are defined in `SRC/INC/setprv.h`.

9.5 `vtools`

The rules for reserved identifiers in headers, implemented in `hdrallow.c` (used by the `hdrdefs` and `hdranal` tools) cover all of the the reserved prefixes and suffixes specified in XSH5.

CONTENTS

" 2