X/Open Shell and Utilities Verification Suite

VSC User and Installation Guide
VSC Release 5.1.1-lite

October, 2000

X/Open Company Limited

X/Open Shell and Utilities Verification Suite
VSC User and Installation Guide

VSCU-511L

X/Open Company Limited

X/Open Shell and Utilities Verification Suite VSCU-511L
VSC User and Installation Guide

Any comments relating to the material in this document may be sent by electronic mail to the VSC support team at:

`vsc_support@opengroup.org`

X/Open Company Limited

X/Open Shell and Utilities Verification Suite VSCU-511L
VSC User and Installation Guide

X/Open Company Limited

1. FOREWORD

1.1 VSC DOCUMENTATION

The X/Open Shell and Utilities Verification Suite, known as VSC, enables you to build and execute test programs that assess system command interfaces for conformance to POSIX.2-1992, and The Open Group Commands and Utilities specification. This distribution, VSC5.1.1L is a *lite* version of the full VSC test suite restricted to a subset of the full VSC test suite. For information on licensing the support for this distribution and/or the full release please contact The Open Group.

This guide describes the use of VSC5-lite to test the following utilities:

alias	**basename**	**bg**	**cat**
cd	**chgrp**	**chmod**	**chown**
cksum	**comm**	**command**	**cp**
csplit	**cut**	**date**	**dd**
df	**dirname**	**du**	**echo**
env	**expand**	**expr**	**false**
fc	**fg**	**fold**	**head**
iconv	**id**	**jobs**	**join**
ln	**logname**	**lp**	**ls**
mkdir	**mkfifo**	**mv**	**newgrp**
nice	**nl**	**nohup**	**od**
paste	**pathchk**	**pr**	**printf**
pwd	**renice**	**rm**	**rmdir**
sh	**sleep**	**sort**	**split**
stty	**sum**	**tail**	**tee**
test	**touch**	**tr**	**true**
tsort	**tty**	**ulimit**	**umask**
unalias	**uname**	**unexpand**	**uniq**
uudecode	**uuencode**	**wc**	
who			

VSC uses part of a set of libraries and programs called the *Test Environment Toolkit* (TET). The TET has its own documentation; this manual contains sufficient information on how to use the TET in conjunction with VSC, but issues pertaining to the TET installation and use of the more extended TET functionality will require the user to refer to the toolkit manuals.

The VSC documentation is in two parts. Part 1 is the VSC User Guide, which gives information about the terminology and structure used in VSC. In addition, the User Guide tells you what resources and facilities you need to use VSC. Part 2 of the documentation is the VSC Installation Guide, which gives you all the information you need to install and run VSC. It is a good idea to read both parts before you start installing and using VSC.

1.1.1 Part 1: VSC User Guide

Contents
The terms used in the VSC documentation and the structure of VSC are explained in the User Guide, so that you are familiar with the VSC system before you start installation. The last chapter tells you the hardware, utilities, time and skills which are necessary to use VSC successfully.

Layout
The layout of the User Guide gives section and paragraph headings in the left margin. Additional sub-headings are in the body of the text.

Pages are numbered in the bottom right-hand corner, although references within the documentation are made by reference to chapter, not numbered pages. This system is used because when you format and print the VSC documentation, the pagination will vary between systems.

1.1.2 Part 2: VSC Installation Guide

Users
The Installation Guide is written for people who are familiar with their system and who have some knowledge of the utilities and options available on it. The installation of VSC should be carried out by an experienced systems administrator, because of the wide range of implementations which can be verified by VSC.

During the stages of installation, VSC needs detailed information about your system.

Contents
Each chapter in the Installation Guide corresponds with a stage of installation and use of VSC. An overview of each chapter is provided in the User Guide. You can simply read the Installation Guide for detailed information about VSC. When you want to install VSC on your system, follow the instructions in the Action Points at the end of each section. Start from the first chapter and continue until you are ready to build the testsets and execute them. The final VSC stage enables you to report on the results of the building and execution stages. The last chapter of the Installation Guide gives information about interpreting the results of the VSC tests.

When you are familiar with VSC, you can use the appendix entitled ‘‘Action Point Summary’’ as a checklist.

Layout
The layout of the Installation Guide corresponds to that of the User Guide, with the addition of Action Points for you to effect after you have read each section.

X/Open Shell and Utilities Verification Suite

Part 1: User Guide
VSC Release 5.1.1-lite

October, 2000

X/Open Company Limited

X/Open Shell and Utilities Verification Suite VSCU-511L
VSC User and Installation Guide

X/Open Company Limited

2. VSC TERMINOLOGY

2.1 INTRODUCTION

This chapter introduces the terms used in the VSC User and Installation Guides. The chapter tells you about the seven stages of using VSC, the terms and naming conventions used in describing the structure of VSC.

2.2 STAGES OF VSC

The X/Open Shell and Utilities Verification Suite, known as VSC, is a series of independent stages. When you have run all of the stages, you can read the VSC reports and interpret the results of the tests to assess the conformance of your system. As each stage is independent, you can re-run the verification suite starting at any of the stages without affecting any of the earlier stages in the suite. Note that these stages are not quite the same as those for VSX.

2.2.1 Stage 1: Preparation

In this stage, you check your system has enough file space available, add entries to the password and group databases and extract the VSC software from the distribution files.

2.2.2 Stage 2: Installation

In this stage, VSC sets up the programs, libraries, and tools that are needed to build and execute the test suite.

2.2.3 Stage 3: Configuration

Next, the VSC configuration script interrogates you for information and asks you questions on the screen. From this information, VSC generates configuration parameter files (such as tetexec.cfg), which are used in the execution stages of VSC.

2.2.4 Stage 4: Building

Because the VSC test programs are written in the Shell language, they are not compiled. Thus the building stage is much simpler (and much faster) than it is in other test suites, such as VSX. This stage places results in a journal file. However, there are no assertions whose results are determined during the building stage.

2.2.5 Stage 5: Execution

When you have completed the preceding stages, you can run the tests to verify your system. VSC executes the test programs under the control of the TET, and keeps the results in a journal file for use by the reporting stage. You can choose to execute all of the installed tests at once, or choose a partial execution.

2.2.6 Stage 6: Reporting

The reporting stage generates reports from the execution journal files. You can also produce a summary report for management information.

2.2.7 Stage 7: Interpreting VSC Results

Using the report generated by VSC, the test source code and the test strategy documentation, you can interpret the results in order to identify the causes of test failures.

2.3 STRUCTURE

VSC uses a common structure for each of the building, execution and reporting stages to locate the different facilities that are to be verified. This is a three-level hierarchy consisting of the following levels:

2.3.1 Section

A section corresponds with a chapter ("Clause" in ISO parlance) of POSIX.2-1992. In addition, there is one section for commands that are specified in XPG4 or XCU5 but not in POSIX.2.

2.3.2 Testset

A testset is the subdivision of an area. A testset usually relates directly to an XPG utility definition. There is a single exception: The tests for the shell are divided into two testsets in two different areas. The tests for the shell specifications in POSIX.2 Chapter 3 can be found in the *shell* testset in the *POSIX.shell* section, while the tests for the shell specifications given in the `sh` subclause of POSIX.2 Chapter 4 can be found in the *sh* testset of the *POSIX.cmd* section.

2.3.3 Test

A test tests a particular statement in an XPG definition. A number of tests, which may depend on each other, make up the testset to assess the conformance of a particular interface.

2.4 NAMING CONVENTIONS

The naming conventions are as follows;
Section
major-section`.`*sub-section*
where *major-section* is either `POSIX` or `XOPEN`.
Testset directory
testset-name
This is the same as the name of the utility under test, except that the shell tests in the *POSIX.shell* section are all in a directory named *shell* rather than *sh*`.`
Testset source
In most cases, these are of the form

utility-name.`sh`

but in a few cases where the tests would otherwise be too large, the testset source files are named

utility-name`_XX.sh`

where `XX` are two-digit numbers starting at `01`.
Testset executable
In VSC, the testset source and executable files are the same file. To distinguish the roles, the Build phase of VSC creates a link to each source file whose name is the name of the source file with the extension `sh` replaced by `ex`.

2.5 ASSERTION NUMBERING CONVENTIONS

VSC contains tests based on three types of assertions:

1. Assertions based on POSIX.2 and taken directly from Draft 8 (or in a few cases Draft 9) of POSIX.3.2. These are numbered in the range 1-800.

2. Assertions based on POSIX.2 but written in the absence of any assertions in POSIX.3.2 Draft 8; this draft was missing much of Chapter 5 (the User Portability Utilities). These are numbered in the range 801-1000.
3. Assertions based on extensions to POSIX.2 from XPG4 or XCU5. These are numbered starting at 1001.

A consequence of this is that the assertion numbering is discontinuous. See section 4.5.2 for a description of how this affect the use of the TET.

2.6 JOURNAL FILE

A TET journal file is generated after the building, execution and cleanup stages, with the results of the stage. The journal file is an ASCII file with control information on the front of each line. It is not usually necessary to examine these files directly, but if you do, details of the file format may be found in the TET specification. The reporting stage uses the journal file as input when you are producing a formatted report.

3. VSC DIRECTORY STRUCTURE

3.1 DIRECTORY STRUCTURE

3.1.1 Introduction

This section gives a brief description of the top level, and is followed by sections with a detailed explanation of each directory hierarchy.

The top level of the VSC directory structure is the directory named `VSC5.1.1-lite`. This directory contains a copy of Release 1.10 of the TET (with some modifications described in section 4.5.2), a directory named `vsc`, which contains the VSC test suite itself, and some files and directories used to configure and build VSC. More precisely:

3.1.2 TET Directories

The contents of the following directories are further explained in the TET documentation.

bin
Delivered empty; TET binaries are installed here.

demo
A sample TET test suite.

doc
The TET documentation.

inc
Header files for the TET source.

lib
Delivered empty; TET libraries are installed here.

src
Source code for the TET.

tet_tmp_dir
An empty directory that is the default parent directory for the hierarchy in which VSC tests are executed.

3.1.3 VSC Build Files

The following files are used during the build phase of VSC:

Build
An executable script used during configuration.

Clean
An executable script that "cleans" the hierarchy in preparation for a new build.

Makefile
The `make` utility is used to build VSC and the TET. This is the top level make file.

SOURCE_ME
A shell script that sets the values of some important shell variables, including `TET_ROOT`.

buildexpect
A script invoked by `Build` to build the *expect* and *tcl* tools that are used to do interactive testing. You may have to modify this file before building.

cleanexpect
A script, invoked by `Clean`, to clean the *expect* and *tcl* directories.

3.2 THE vsc DIRECTORY

The `vsc` directory contains directories with the source code for the VSC tests and for various auxiliary programs that VSC needs. It also contains a nuber of scripts used by the `Build` script, and a number of scripts and files that you an use to create custom test scenarios.

3.2.1 Binaries: `Bin`

A common location for all the VSC executables that are invoked by test cases. These include binaries as well as shell scripts.

3.2.2 Results: `results`

A directory under which the journal files for the installation, building, execution and cleanup stages are written.

3.2.3 Source: `Src`

The source files for shell scripts and utilities used to configure, install and build VSC. It is further described in section 3.3.

3.2.4 Headers: `Inc`

This directory contains common headers used by several VSC utilities.

3.2.5 Library: `Lib`

This directory is a data library. It contains a subdirectory named `Data`, which in turn has subdirectories for each locale that is supported by VSC. There is only one such locale, the `POSIX` locale, at the moment. The locale directories contain tables used in testing the regular expression general assertions. They are derived directly from POSIX.3.2 drafts 8 and 9.

The `Lib` directory also contains files used to initialize `tcl`.

3.2.6 Scenario Files: `ScenDir`

This directory contains TET scenario files that can be used to build custom scenarios under the control of the `BuildScen` script.

3.2.7 Shell Functions: `Shlib`

This directory hierarchy contains source files that define shell functions. The files are sourced by test cases, which then defines the functions for those test cases.

3.2.8 Reporting Tools: `Tools`

This directory contains the source code for a number of reporting and debugging tools.

3.2.9 Testset: `tset`

This directory hierarchy contains the source files which are used to build the VSC testsets. The tset directory contains section directories.
Section
The section directories are named using the naming conventions explained in the chapter entitled ‘‘VSC Terminology’’. Each section directory contains area directories, below which are the testsets.

3.3 SOURCE DIRECTORY STRUCTURE

There are the following directories at the top of the source (or `Src`) directory structure.

3.3.1 *Test Suite API:* `API`

This contains the interface routines that the test cases use to invoke the TET interfaces. For various reasons, no direct calls to the TET interfaces are made.

3.3.2 *General Assertion Test Routines:* `GA`

This directory contains subdirectories with C programs or Shell functions that are used in testing the POSIX.3.2 General Assertions.

Note that the numbering of the General Assertions for pathname resolution follows that in Draft 8 of POSIX.3.2, not Draft 9.

3.3.3 *Implementation-specific Routines:* `ImplSpec`

The programs and scripts in this directory are not portable within the POSIX.2 domain. Most of them *are* portable within the XPG4 or XCU5 domain, but testers may have to modify or re-implement some portion of them. You are free to do so as long as you do not change the semantics of these routines.

3.3.4 *Interactive Testing Tools:* `Interact`

This directory contains two subdirectories, `expect` and `tcl`. The `expect` directory containse release 5.4 of the *expect* tool written by Don Libes of NIST. *expect* requires the support of *tcl*, and release 7.3 of *tcl* is provided in the `tcl` directory.

These tools have been successfully built on all known UNIX™ systems, but *they are not guaranteed to be portable*. In particular, they use facilities outside the XPG4 or XCU5 base domain. We believe that they can be implemented on any system that conforms to the requirements of the UNIX trademark.

Expect and tcl are required to run the VSC test suite.

3.3.5 *Shell Scripts and Functions:* `Shell`

This directory contains the source of a number of shell scripts and functions used by testcases throughout the test suite. By convention, files with the extension `.SH` contain shell functions, and such files are sourced ("dotted") by the test programs. Files with the extension `.sh` are shell scripts and the corresponding filename without the extension is placed in the `Bin` directory as an executable file.

4. RESOURCES

4.1 INTRODUCTION

VSC requires the following major resources to run successfully:

- an adequate computer hardware environment
- the correct base utilities
- enough time to complete the task
- a system administrator with the necessary skill to run VSC.

4.2 COMPUTER HARDWARE

4.2.1 Disk Space

Installation
VSC uses a considerable amount of disk space. This is described in detail in Chapter 5, Section 5.2.1.

4.2.2 Exclusive Use

It is recommended that you have exclusive use of the machine when you execute the tests. **In particular, a test for branding purposes should only be executed when exclusive use is available since these tests may affect other users of the system.**

4.2.3 Devices

In order to execute all of the tests in VSC you will need the following devices to be available on your machine.

1. A small empty file system which should be mounted read-write before executing the tests.
2. A second small empty file system which should be mounted read-only before executing the tests.
3. A terminal at which a user is logged on and remains logged on while the tests are executing.
4. A character special file, if your system supports such file types.
5. A block special file, if your system supports such file types.

4.3 UTILITIES

Since VSC tests the XPG4/XCU5 shell and utilities, it assumes the presence of many of these utilities in building and executing the tests. If you are only testing a subset of XPG4/XCU5 or of POSIX.2, there are certain utilities that you will need that may be outside the scope of that subset.

4.3.1 POSIX Shell

The configuration and installation stages use scripts which are written for a POSIX shell. It must be named `sh`. That is, when a utility named `sh` is invoked, it must be a POSIX shell.

4.3.2 `make`

The installation and building stages use `make` files.

4.3.3 Compiler

VSC requires a C compiler to build TET, expect, tcl and a number of utilities, all of which are in the *$TET_ROOT/Src/ImplSpec* directory. If you claim conformance to XCU5, XPG4 or to POSIX.2 with the `POSIX2_C_DEV` option, then you must have a C compiler named `c89`.

4.3.4 Library Archiver

VSC requires a library archiver and other software to order the libraries. The ordering software may be inherent in the library archiver, the `ranlib` utility or the utility pair `lorder` and `tsort`.

4.3.5 Miscellaneous Utilities

The version of the `privscript` script that is shipped with the test suite uses the `su` utility, although testers are free to revise this script as appropriate for their systems.

4.4 TIME

The installation and execution of VSC (by an experienced VSC user) should take less than 45 hours. A longer period may be needed on slower processors or on systems where there is a large amount of overhead associated with initiating a new process; as is typical of shell scripts, VSC initiates many, many processes during its execution.

The large file system tests take a significant portion of the time when configured properly for the implementation maximum file size and can contribute 10 hours or more to the total run time depending on the filesystem implementation.

The design of VSC enables you to run the building and execution stages without intervention and to review the results afterwards.

4.5 SKILLS

4.5.1 Using VSC

To install, build and execute the tests correctly, you must be able to give detailed information about your system to VSC. VSC is not a product which can be loaded and run without assistance. VSC is designed for use by an experienced systems administrator who has considerable knowledge of the utilities available, the devices and their associated device files on the system. For example, you will need to know whether your system can create links across file systems, or whether the system requires write permission in a directory in order to rename a file in that directory with the `mv` utility. Without this level of information, you may find difficulty in using VSC.

4.5.2 Experience With the TET

You should have some experience using TET-based test suites, or you should familiarize yourself with the TET documentation shipped in *$TET_ROOT/doc*.

A slightly modified version of TET Version 1.10 is shipped with VSC. More precisely: VSC comes with two versions of the Shell API and Test Case Manager. Both have been modified from Release 1.10 of the TET. They can be found in subdirectories named `xpg3sh` and `posix_sh` under the `$TET_ROOT/src` directory.

VSC uses an assertion numbering scheme (described in section 2.5) that creates large gaps in the numbers used by test purposes and invocable components. The unmodified TET 1.10 release writes a warning message whenever it encounters a missing invocable component number. Both versions of the VSC copy of TET have suppressed the code in `tcm.sh` that issues these warnings, since otherwise users would get thousands of such warnings each time VSC is run.

The second modification only appears in the code in the `posix_sh` directory. It is a performance enhancement, and takes advantage of the shell built-in arithmetic that is specified in the POSIX.2 shell but was not part of the XPG3 shell specification. Using this version of the TET Shell API will cause execution of VSC to run noticeably faster. If your shell supports POSIX.2-stype shell arithmetic (as all XPG4/XCU5 conformant shells must) then we recommend that you use this version of the TET shell API. During the Installation stage of VSC you will be queried as to which version you wish to use. These instructions are written with the assumption that the `vsc` directory is located directly in the `$TET_ROOT` directory, as will be true if you use the TET that comes with VSC. It is important that the TET `tet_tmp_dir` directory exist in the same filesystem as the test suite, since a number of the tests depend on this.

Both the `xpg3sh` and `posix_sh` versions have been modified to wrap long lines at 75 characters.

4.5.3 Interpreting Results

When VSC reports on the results of the tests, the report may show that a facility does not work in the way that VSC expects. To investigate the cause of failed tests, you may need more information than is available from the VSC report. Considerable skill and an understanding of the shell and utilities are necessary to gather this information and to investigate the results thoroughly.

X/Open Shell and Utilities Verification Suite

Part 2: Installation Guide
VSC Release 5.1.1-lite

October, 2000

X/Open Company Limited

X/Open Shell and Utilities Verification Suite VSCU-511L
VSC User and Installation Guide

X/Open Company Limited

5. PREPARATION

5.1 INTRODUCTION

Before you install VSC you must first check there is file space available and then create apropriate user accounts and groups an install the users in the groups. When you are ready to load the VSC distribution, you must unpack the distribution files and then check that the contents were extracted correctly.

5.2 PREPARING YOUR SYSTEM

5.2.1 File Space Requirements

The target file system must have enough free space available to build and execute the test suite. Because very little of VSC is compiled, the amount of space required is not much greater than the amount of space occupied by the delivered test suite, and does not vary greatly between CISC and RISC systems. The delivered system occupies about 40 megabytes, and you will need another 10 megabytes to install it.

Note the following points:

1. You will need space to hold the reports. Allow 5Mb minimum.
2. The tests also use some temporary file space during execution.

Action Points

1. Check there is enough free space available to unpack and install the software.

5.2.2 VSC User and Group Accounts

You must add a user account which will be the VSC Test User to your system. You should also add a second account. Neither of these accounts should be the accounts of actual users, since the execution of VSC modifies part of the user environments for these users, such as their mail files. You can choose the names of these users to be any valid user names.

Note: The VSC Test User's account must not have any commands in a *.profile* or other startup shell file that change the value of the `PS1` prompt.

You must have available three user IDs that are not the IDs of either of the accounts that were just described. These should be IDs that can be used as arguments to a `chown` command. On many systems, any valid user ID can be used, regardless of whether or not it corresponds to an existing account. But if your system is one on which a numeric argument to `chown` must be the ID of an existing account, then you should create three more user accounts.

The VSC Test User must be made a member of at least two groups. Because no aspect of the group environment is modified by the test suite, it makes no difference whether or not these are new groups specifically created for VSU or pre-existing groups.

You must have available three group IDs that are not the IDs of any groups to which the VSC Test User belongs. They must be valid when used as arguments to a `chgrp` call. As with the user IDs, these may or may not have to be the IDs of real groups, depending on the behavior of your system.

The VSC Test User must have a login shell that is a conforming POSIX.2 shell.

If your system has extensions which are enabled by environment variables, and the default settings of these variables would cause behaviour to differ from that required for compliance, you must ensure that these variables are set so as to disable the extensions in the login script for the VSC Test User. For example, if the setting of the `LANG` environment variable is such that the shell has a locale setting other than the POSIX locale then this would typically be disabled by adding the line

```
unset LANG
```

to the `.profile` for the VSC Test User.

Action Points

1. Create the appropriate group entries, usually in the file `/etc/group`.
2. Create the appropriate user entries and login shell, usually in the file `/etc/passwd`.
3. Ensure that any extensions enabled by environment variables that would cause non-compliant behaviour are disabled in the login script for the VSC Test User.

5.3 LOADING THE VSC DISTRIBUTION

5.3.1 Unpacking the Distribution File

VSC is packed into the file `VSC511L.t.Z` for distribution. You can unpack it at any convenient place in the file hierarchy (taking account of the space requirements mentioned above).

When you unpack the distribution file, the contents are installed in a hierarchy which starts from the current working directory. Make sure that you log in as the VSC Test User. (If you extract the hierarchy while logged in as another user, then before running the `Build` script described in the next chapter you must change the ownership of the entire hierarchy to be owned by the VSC Test User. This will ensure that the access permissions for the files are correct.)

Action Points

1. Log in to the test system as the VSC Test User.
2. Ensure you have write permission in the directory where you wish to extract the test suite..
3. Unpack the distribution file using an appropriate command to decompress the file and extract all files from the resulting POSIX `tar` archive, e.g.:

```
zcat VSC511L.t.Z | tar xof -
```

6. INSTALLING VSC

6.1 INTRODUCTION

The installation of VSC consists of running the `Build` script in the top level. This script starts by asking you three questions, and then proceeds to traverse the source hierarchy for VSC, excluding the testset hierarchy, and `make` everything that it finds. This includes the TET. You may have another copy of TET 1.10 on your system, but you must use the one delivered with VSC for executing VSC tests. (See section 4.5.2.)

6.2 BEFORE RUNNING Build

Before you can run the VSC installation script, you may need to modify some files in the distribution in a manner appropriate for your system.

6.2.1 Set Shell Variables

Log in as the VSC Test User, and ensure that you own all of the files in the distribution hierarchy. Change directories to `VSC5.1.1-lite`. If the shell variable `TET_ROOT` is set, unset it. Then, using the '.' (dot) command, read the file named `SOURCE_ME` by typing:

```
. ./SOURCE_ME
```

This will set `TET_ROOT` to the current directory, and will also set the values of a series of related test suite variables that describe the locations of various parts of VSC.

Action Points

1. Log in as the VSC Test User.
2. Change directories to `VSC5.1.1-lite`.
3. Type `. ./SOURCE_ME`

6.2.2 Configure the TET.

The principal task is to edit the makefiles supplied with the TET source in a manner appropriate to your system. The included makefiles are

- *$TET_ROOT/src/xpg3sh/api/makefile*
- *$TET_ROOT/src/posix_sh/api/makefile*
- *$TET_ROOT/src/posix_c/api/makefile*
- *$TET_ROOT/src/posix_c/tcc/makefile*
- *$TET_ROOT/src/posix_c/makefile*

You must supply correct signal numbers in the first of these makefiles, and the proper names for some `make` variables such as `CC` in the others. The sample values provided should be suitable for Linux systems.

Action Points

1. Check the TET makefiles to ensure the correct signal numbers and `make` variable values are suitable for your system.

6.2.3 If Necessary, Modify Implementation-Specific Source Files

The actions in this subsection are implementation-dependent. There is no good method, short of trial and error, to determine which of these you must carry out.

If necessary, edit the file `buildexpect` in `$TET_ROOT` to export values for `CPP` and `CFLAGS` that will use the largest namespace from your system's headers.

If necessary, edit the files named `Makefile.in` in the `tcl` and `expect` subdirectories of `$TET_ROOT/vsc/Src/Interact`. These files are processed by a configuration script to produce `Makefile`s that are then used to build `tcl` and `expect` respectively. In each of the `Makefile.in` files is a clearly marked section near the start (delimited by the lines *"The following lines are things you may want to change"* and *"End of things you may want to change"*) where you may want to change some configurable variables. We have found that on many Unix systems no changes are required. However, you may need to use trial and error in order to determine what changes, if any, are needed for your system.

If necessary, edit (or in extreme cases re-implement) the programs in `$TET_ROOT/vsc/Src/ImplSpec`. These are implementation-specific (i.e. nonportable) tools that are used by the test suite. The versions provided are likely to work on many Unix-like systems that conforms to POSIX.1. The shell script `privscript` in `$TET_ROOT/vsc` is also implementation-specific; it uses the `su` utility, which is not portable.

If you make any changes to the C source files in the `ImplSpec` directory, you may need to change the set of feature-test macros defined in the `CFLAGS` variable in `$TET_ROOT/vsc/Src/ImplSpec/Makefile`. The default is `-D_XOPEN_SOURCE=1 -D_XOPEN_SOURCE_EXTENDED=1`, which restricts the visible namespace to just UNIX95 symbols (on systems that comply with XSH4.2).

Action Points

1. If necessary, edit the `buildexpect` script to set the values of `CPP` and `CFLAGS`.
2. If necessary, edit the files named `Makefile.in` in the `tcl` and `expect` source directories.
3. If necessary, edit or re-implement the implementation-specific programs in the `ImplSpec` directory. Modify `CFLAGS`, if necessary, in `ImplSpec/Makefile`.

6.3 THE Build DIALOGUE

6.3.1 Set Shell Variables

If you have logged off since you configured the TET, then again change directories to `VSC5.1.1-lite`. If the shell variable `TET_ROOT` is set, unset it. Then, using the '.' (dot) command, read the file named `SOURCE_ME` by typing:

```
. ./SOURCE_ME
```

6.3.2 Run 'Build'

Run the `Build script by typing`

```
./Build
```

When you invoke `Build`, the first question that you will be asked is:

```
Do you wish to use the POSIX Shell version of the TET Shell API and
tcm?  This version uses built-in shell arithmetic rather than calls
to 'expr', with a consequent gain in performance.  The default is
"no"; that is, if you do not respond, you will use the standard
TET modules: [y/n]
```

Answer `y` if you wish to use the modified TET Shell API.

The second question is:

```
What is the name of the command to be used to invoke the C compiler to
build the TET and the implementation-specific C programs in the ImplSpec
directory?  The default is "c89":
```

If your C compiler is not named `c89`, then supply its name here. If you claim conformance to XPG4/XCU5, then you must have a C compiler named `c89`. This is the default response.

The third question has to do with the `tcl` and `expect` tools. Because these are not XPG4/XCU5-clean, you may need to use another compiler that has more relaxed namespace rules in order to compile `tcl` and `expect`. (There's really no good way to know this short of experimenting.) The third question is:

```
What is the name of the command to be used to invoke the C compiler to
build tcl and expect?  The default is "XXX":
```

where "XXX" is the response to the previous question. On some systems it's easiest to use `c89` to install most of the test suite and `cc` to install `tcl` and `expect`.

6.3.3 Building the VSC Tools and Utilities

After you respond to these three questions, you should watch while `Build` proceeds to try to build the VSC tools. If an error occurs, you should analyze it and do whatever is appropriate on your system to proceed. **Before attempting to run `Build` again, always run the `Clean` script in `$TET_ROOT`.** This removes files created by `Build`, so that another attempt to run `Build` will not fail due to the existence of targets that you are trying to create.)

Although the system may run for quite a few minutes on its own, you should be prepared: the `Build` script automatically invokes the `vsc/configure` script when it has finished building all the VSC tools and utilities. This script takes the user through a long dialogue that is described in the next section.

Action Points

1. Run the `Build` script. If it fails, run `Clean` and analyze the causes of the failure. If **Build** succeeds, it will lead directly to the configuration dialogue shown in the next chapter.

6.4 CLEANING VSC

If for any reason you wish to remove and re-install the VSC tools and utilities, you can execute the `Clean` script located in the `$TET_ROOT` directory. This removes all the executable programs and shell scripts from the places where they have been installed by `Build`. You can also clean (uninstall) any subdirectory of the source hierarchy by changing to that directory and issuing the command

```
make clean
```

7. CONFIGURING VSC

7.1 INTRODUCTION

The VSC test suite needs information about optional behavior of your system, and information about the way that you have configured the test users and groups, before it can be executed. You provide this information during the configuration of VSC. Unlike VSX, VSC combines this configuration with the Installation phase, that is, the building of the test tools and the TET. This is not the same as the build phase of the test suite itself.

Any time after the Install phase is completed, you can reconfigure VSC by running the script named `configure` in the `$TET_ROOT/vsc` directory.

VSC Lite supports configuration in one mode only: POSIX.2 mode.

During the configuration stage, VSC finds out the specific details about your system and uses the information to generate the `tetexec.cfg` file for the test. VSC finds the information both by interrogating your system and by asking you questions.

Before you start, you must find out the information that is needed for configuration. Read this chapter and write the information for your system next to each question. When you have finished configuring VSC, verify that the `tetexec.cfg` file is correct for your system.

7.2 CONFIGURATION

7.2.1 Introduction

The shell script begins interrogating your system and asking you additional questions. In some cases it offers VSC default values which you can use. If you have congfigured VSC before, you can use an existing `tetexec.cfg` file to provide defaults. The confguration script will ask you if you want to do this, and if you respond affirmatively, it will ask for the name of the existing file.

Before you can configure VSC, you must prepare a number of data files and disk partitions. You will be asked for the names of two codesets such that your system's `iconv` utility can translate one to the other. You must have a file prepared that uses the first (source, or "from") codeset.

You should have a small file system on which the tester has permission to create files and directories.

You should have a small file system mounted read-only, if your system supports read-only file systems.

VSC includes a *vrpt* program similar to the one used in VSX4. To support *vrpt*'s reporting requirements, the VSC4 configuration script starts by asking for three additional pieces of information:

Test Engineer	The script asks for the full name of the Test Engineer who is running the tests.
Test Organization	The script asks for the name of the organization that is running the tests.
Organization Address	The script asks for the address of this organization. Up to five lines of address information may be supplied. At present this information is unused. but it may be added automatically to the *vrpt* summary report at a later date.

System Identification The name and model number of the system under test, and the version of the operating system. Please use just one line.

The answers to these questions are not used during the execution of VSC.

7.2.2 Configuration Questions

You must prepare answers to a number of questions which you will be asked during VSC configuration and which affect the execution of the test suite. In some cases you will be asked to provide file names. You should provide either absolute pathnames or pathnames relative to the directory `$TET_ROOT/vsc`.

The following is a complete list of all questions that may be asked during configuration. Questions specific to certain test modes are marked with those test modes in parenthesis.

- `Enter the login name under which the tests will be run.  This login name must be a member of at least two groups:`

  This should be a user whose mail files can routinely be destroyed without consequences, since this will occur in the `mailx` *test case.*

- `Enter the names of two groups.  Separate the names with a space; do not use a comma.   The user running the tests must be a member of both groups:`

  These groups will be used as arguments to `chgrp` *commands.*

- `Enter the name of a group that is configured in the group database and to which the tester DOES NOT belong:`

- `Enter a user ID that is valid (i.e. within the range accepted by your system) but that is NOT the UID of any user configured in the system’s user database:`

  This user ID will be used, for instance, in tests of the `ls` *utility to verify that when a user ID cannot be found in the system’s user database then the numeric ID is displayed in place of a name.*

- `Enter a group ID that is valid (i.e. within the range accepted by your system) but that is NOT the GID of any group configured in the system’s group database:`

  Similar use to the previous item.

- `Enter three numerical user IDs of users other than the tester.  Separate the IDs with spaces; do not use commas.  Do not choose user IDs that are associated with privileges.`

  These user IDs need not correspond to actual users in the user database, but they should be IDs that can be successfully used as arguments to a `chown` *command. If your system requires that such IDs be configured then add them as real users.*

- `Enter three numerical group IDs of groups other than the tester’s primary group or supplementary groups.  Separate the IDs with spaces; do not use commas. Do not choose group IDs that are associated with privileges.`

  Similar to the previous remark, but for `chgrp` *rather than* `chown`*.*

- `Enter a valid username of a user other than the tester.  This should be a user name that is in the system’s database, but the user named here must not be logged in to the test system during testing:`

  As the question indicates, this should be a "real" user, i.e. one that is configured in the user database. It is best to chose a name that nobody ever uses to log in, as the named user must not be logged in during testing.

- `Does the system support the POSIX.1 C language interfaces? [y/n]`

  The response to this question determines whether or not General Assertion 04 (that the utility can be executed via the POSIX.1 exec() family of functions) is tested for each utility.

- `Does your system support the use of numeric signal numbers in the shell's "trap" command? [y/n]`

  Shell traps can be triggered by the receipt of signals. POSIX.2 specifies a standard set of signal names that can be used in the `trap` *statements, and optionally allows signal numbers to be used in place of the names. Indicate whether or not your implementations's shell supports the use of numbers.*

-
  ```
  (XCU5 only)
  Specify the base of a range of 16 key values that may be used
  when creating semaphores, shared memory segments or message
  queues.  This value may be specified in octal (0########), decimal,
  or hex (0x########) format.  [Numeric answer]
  ```

- `Does your system support the "ps" utility as specified in POSIX.2 subclause 5.22? [y/n]`

  Support of POSIX.2 chapter 5 is optional. However, the tests of certain required utilities depend on the availability of the `ps` *utility. If you system supports* `POSIX2_UPE` *as determined by a call to* `getconf` *then this system is skipped, with a* `yes` *answer assumed.*

- `Does your system's implementation of the "printf" utility support floating point conversions? [y/n]`

  Enter "y" if the `printf` *utility on your system supports floating point conversions such as %e, %f and %g.*

- `Does your system support a communications line that is configured to use the POSIX.1 termios interface?  [y/n]`

  Enter "y" if you can run the test suite from a port that supports POSIX.1-stype termios features. (This can be a console, a pty, a physical terminal, or any other type of connection that has termios semantics.) The answer to this question is used in the `stty` *test: If you answer "y" then the* `stty` *test case assumes that it can set the various parameters specified in POSIX.1 termios and POSIX.2* `stty`*.*

- `Enter the size in bits of one byte on the system under test: (Default is "8")`

  The test suite must know the size in bits of a byte. If, as on most systems, the value is 8, you can respond by just pressing <Enter>.

At this point an optional question may be asked, depending on the configuration of your system. If your system supports the POSIX2_SW_DEV or the XPG4/XCU5 DEV options, **and** *the file $TET_ROOT/src/posix_c/api/tcm.o does not exist, then you will be prompted for the following piece of information.*

- `Enter the pathname of a prepared object file.`

  A number of test cases, such as `strip`*, require the name of an unstripped object file. The configure script looks for the file* `$TET_ROOT/src/posix_c/api/tcm.o`*, and if that exists it is used and this question is skipped.*

- `Enter the pathname of an unstripped executable file.`

  A number of test cases require the name of a binary executable file. The configure script looks for the file `$TET_ROOT/bin/tcc`*, and if that exists it is used and this question is skipped.*

- `Enter the location (absolute pathname) of the directory containing the default libraries searched by the "fort77" compiler.`

  This question is only asked if `getconf` *indicates that the system supports the* `POSIX2_FORT_DEV` *option.*

- `Enter a valid destination value for use with the "lp" command.`

  A "destination" is an argument (i.e. a printer name) that can be used as a valid argument to the `-d` *option.*

- `Now please enter an invalid destination value for use with the "lp" command.`

- `Enter the local mail address of the user who will be running the tests. Typically this is just the login name of the user.`

- `Does the system support character-mode terminals? [y/n]`

- `Does the system support block-mode terminals? [y/n]`

- `(XCU5 only)`
  `Enter the full path to the block device of the file system on which the VSC test hierarchy resides:`

- `Enter the mount point of a read-only file system.  If the system does not support read-only file systems, then just press <Enter>.`

  Support for read-only file systems is not required. However, if your system supports them, you should provide a small file system that has been mounted read-only. The tests that use this file system generally are tests that ensure that certain operations will fail on read-only file systems, and report that failure.

- `Enter the full pathname of a character special file.  If the system does not support character special files, then just press <Enter>:`

- `Enter the full pathname of a block special file.  If the system does not support block special files, then just press <Enter>:`

- `In the system's handling of Basic Regular Expressions, does a ^ or $ character in a subexpression anchor that subexpression? [y/n]`

  POSIX.2 allows two different behaviors: treating these characters as anchors in subexpressions, or treating them as themselves. Respond "yes" if your systems treats them as anchors, and "no" otherwise. It is conceivable that a system might treat one of ^ or $ as an anchor in subexpressions but not the other. VSC currently has no provision for this behavior, which we have never encountered.

- `Does the implementation's "mv" utility support moving files across file systems? [y/n]`

- `Does the system require write permission for an existing directory in order for "mv" to rename that directory? [y/n]`

- `Does the system consider it to be an error if a nonprivileged process attempts to rename a file or directory using "mv" while that file or directory is being used by the system or another process? [y/n]`

- `Does the implementation's "ln" utility support creating hard links across file systems? [y/n]`

- `What is the value of LINK_MAX on the file hierarchy under which the test suite will execute? [Numeric answer]`

  The answer must be at least 6 (_POSIX_LINK_MAX). You can probably determine the correct answer for your system by typing the command

  ```
  getconf LINK_MAX .
  ```

Don't forget the trailing '.'; you need to supply a pathname argument to `getconf` *when querying LINK_MAX.*

- `What is the file allocation block size in bytes (the number of bytes that are removed from the file free space when a file consisting of a single byte is created)? If the value is not fixed, then enter the minimum number of bytes that the system consumes.  [Numeric answer]`

  The answer to this question is used in testing the `df` *and* `du` *utilities. If your system has a variable granularity for file allocation, enter the minimum value (i.e. the smallest size of disc block that can be allocated).*

7.3 POST CONFIGURATION

When you are running `Build`, immediately after configuration you are prompted for the superuser password. VSC needs to give certain utilities privileges in order to run tests that require privileges. **As soon as you provide the superuser password, your system is insecure.** There is not much that you can do about this. If you do not want to leave an untrusted privileged program on the system, then do not provide the superuser password. However, in that case you will have to provide privileges to three programs in the `$TET_ROOT/vsc/Bin` directory before you can run VSC. These are:

— ExecWithPriv

— ExecAsUser

— Execute

On most systems, you provide these privileges by changing the owner of these files to the superuser and setting the Set-User-ID and Set-Group-ID bits for these files.

7.4 SAMPLE tetexec.cfg FILE

Here is a sample *tetexec.cfg* file as produced by running `Build`. **Note that not all the values in this example are applicable to VSC5-lite .**

```
###########################################################################
#
# VSC5 configured on Wed Apr 17 11:01:07 PDT 1997 by vsc0
#
###########################################################################

########################################################################
#
# Copyright X/Open Company Ltd. 1993-1997
#
#    Written by Mindcraft, Inc.
#
#    @(#)tetexec.cfg.orig 5.0.0
#

########################################################################
#
# Do not change the next line:
TET_OUTPUT_CAPTURE=False

########################################################################
#
```

```
# TET_EXEC_TOOL should not be set for a conformance or branding run.
# For debugging and testing runs you may wish to set it to the absolute
# pathname of a shell that is not invoked under the name "sh" but that
# has all or most of the semantics of a POSIX.2 shell.
#
#TET_EXEC_TOOL=/usr/bin/ksh

########################################################################
#
# The PCTS_* variables are test suite constants.  Their existence is
# required by POSIX.3.2 (IEEE Draft Std 2003.2).  The values chosen
# here are NOT configurable; they should be the same for all instances
# of a given release of VSC.
PCTS_NAME_MAX=511
PCTS_PATH_MAX=2047
PCTS_LINK_MAX=255

########################################################################
#
# VSC_WINK, VSC_NAP and VSC_SNOOZE are convenient delay times.
# They are 2, 10 and 30 second respectively, by default, but you can
# edit this file to change them as you like for your system.  The
# configure script does not prompt for them.
VSC_WINK=2
VSC_NAP=10
VSC_SNOOZE=30

########################################################################
#
# VSC_EXPECT_SUPPORT is TRUE by default.  If your system is unable to
# support "expect", then change this to FALSE.  This will result in
# many UNRESOLVED test results.
VSC_EXPECT_SUPPORT=TRUE


########################################################################
#
# The following symbols define ASCII (ISO/IEC 646:1991 IRV) values for
# various control characters.  Modify them if you use another character
# set.  Use a three-digit octal sequence, without backslashes.  Note: the
# use of these symbols within the test suite is not complete.
#
# Escape (Esc)
VSC_ESC=033
# EOF (ctrl-D)
VSC_END_OF_FILE=004
# Backspace (ctrl-H)
VSC_BS=010
# Kill line (ctrl-U)
VSC_KILL=025
# newline (ctrl-J)
VSC_NL=012
# literal-next (ctrl-V)
VSC_LNEXT=026
```

```
# Stop (ctrl-Z)
VSC_STOP=032
# Redraw (ctrl-L)
VSC_REDRAW=014
# Interrupt (ctrl-C)
VSC_INTR=003
# Word-advance (ctrl-W)
VSC_CTLW=027
# ctrl-P (used only in Shell emacs test)
VSC_CTLP=020

########################################################################
#
# The VSC_Largefile_OPTS variable defines any additional options to be
# added to the Largefile utility command line.  The Largefile utility
# creates large files.  If real large file testing is to be done then
# set VSC_Largefile_OPTS to -f to cause a file to be filled to the
# offset point rather than seeking to the offset point.  By default,
# sparse files are created to save space, and the variable is set to
# null.
VSC_Largefile_OPTS=

########################################################################
#
# The VSC_PAX_BIG_FILE_SUPPORT defines whether or not pax tests related
# to specifying large file operands up to 8GB should be tested or not.
# This should be set to TRUE if supported or undefined if not supported.
VSC_PAX_BIG_FILE_SUPPORT=TRUE

########################################################################
#
# VSC_CONFIG_TYPE is the type of configuration that has just been
# performed; it is XPG4, POSIX2 or FIPS189
VSC_CONFIG_TYPE=XPG5

########################################################################
#
# VSC_OPER is the name of the person who is actually running the tests.
# This name appears on the VSC output report produced by vrpt.
VSC_OPER=vsc0

########################################################################
#
# VSC_ORG is the name of the organization that is running the tests.
# This name appears on the VSC output report produced by vrpt.
VSC_ORG=SuperTesters Inc

########################################################################
#
# VSC_ADDR[1-5]is the address of the organization that is running the
# tests.  This address name appears on the VSC output report produced
# by vrpt.
VSC_ADDR1=1969 Armstrong Lane
VSC_ADDR2=Mare Tranquillitatis
```

```
VSC_ADDR3=Luna
VSC_ADDR4=
VSC_ADDR5=

#########################################################################
#
# VSC_SYS is a decription of the system (hardware and software) under
# test.  This appears on the VSC output report produced by vrpt.
VSC_SYS=HAL 9000, OS/2001

#########################################################################
#
# VSC_TESTER is the name of the user configured to run VSC
# VSC_TESTER_UID is the numeric UID of that user.
VSC_TESTER=vsc0
VSC_TESTER_UID=802

#########################################################################
#
# VSC_GRP_NAME_1 and VSC_GRP_NAME_2 are the names of groups
# to which $VSC_TESTER must belong.
VSC_GRP_NAME_1=vsc0
VSC_GRP_NAME_2=testers

#########################################################################
#
# VSC_NON_MEMBER_GRP and VSC_NON_MEMBER_GID are the name and GID
# respectively of a configured group to which the tester does not belong.
VSC_NON_MEMBER_GRP=sysadm
VSC_NON_MEMBER_GID=170

#########################################################################
#
# VSC_NAMELESS_UID is a user ID that is lexically valid (i.e. is
# a number in the range accepted by the system) but that does not
# correspond to any name in the user database.
VSC_NAMELESS_UID=9999

#########################################################################
#
# VSC_NAMELESS_GID is a group ID that is lexically valid (i.e. is
# a number in the range accepted by the system) but that does not
# correspond to any name in the group database.
VSC_NAMELESS_GID=9999

#########################################################################
#
# VSC_GRP_ID_1 and VSC_GRP_ID_2 are the numerical group IDs
# of VSC_GRP_NAME_1 and VSC_GRP_NAME_2 respectively.
VSC_GRP_ID_1=200
VSC_GRP_ID_2=160

#########################################################################
#
```

```
# POSIX1_SUPPORT should be set to "TRUE" if the system claims support
# for the C language interfaces of ISO/IEC IS 9945-1:1990, and to
# "undefined" otherwise.
POSIX1_SUPPORT=TRUE

########################################################################
#
# XPG4_SW_DEV should be set to "TRUE" if the system claims support
# for the XPG4 Development utilities, and to "undefined" otherwise.
# NOTE: this version specific variable is here for compatibility.
# Use VSC_XPG_SW_DEV in the future along with VSC_CONFIG_TYPE
# This variable has precedence in XPG4 mode
XPG4_SW_DEV=TRUE

########################################################################
#
# VSC_XPG_SW_DEV should be set to "TRUE" if the system claims support
# for the XPG4/XPG5 Development utilities, and to "undefined" otherwise.
VSC_XPG_SW_DEV=undefined

########################################################################
#
# If the implementation supports XPG5:
# VSC_XBS5_ILP32_OFF32, VSC_XBS5_ILP32_OFFBIG VSC_XBS5_LP64_OFF64 and
# VSC_XBS5_LPBIG_OFFBIG should be set to TRUE or undefined depending on
# whteher the implementation supports the associated c89 programming
# mode. At least one of these modes must be TRUE.
VSC_XBS5_ILP32_OFF32_SUPPORT=TRUE
VSC_XBS5_ILP32_OFFBIG_SUPPORT=undefined
VSC_XBS5_LP64_OFF64_SUPPORT=undefined
VSC_XBS5_LPBIG_OFFBIG_SUPPORT=undefined

########################################################################
#
# If the implementation supports XPG5:
# VSC_DEFAULT_C89_PROG_ENV should be set to the default c89 programming
# environment; either XBS5_ILP32_OFF32, XBS5_ILP32_OFFBIG, XBS5_LP64_OFF64
# or XBS5_LPBIG_OFFBIG.
VSC_DEFAULT_C89_PROG_ENV=XBS5_ILP32_OFF32

########################################################################
#
# VSC_XOPEN_RT should be set to "TRUE" if the system claims support
# for the XPG Realtime option, and to "undefined" otherwise.
VSC_XOPEN_RT=TRUE

########################################################################
#
# VSC_XOPEN_RTT should be set to "TRUE" if the system claims support
# for the XPG Realtime threads option, and to "undefined" otherwise.
VSC_XOPEN_RTT=TRUE

########################################################################
#
```

```
# VSC_XPG_FORT should be set to "TRUE" if the system claims support
# for the XPG4 FORTRAN option, and to "undefined" otherwise.
VSC_XPG_FORT=TRUE

########################################################################
#
# VSC_XPG_LEGACY should be set to "TRUE" if the system claims support
# for the XPG5 Legacy option, and to "undefined" otherwise.
VSC_XPG_LEGACY=undefined

########################################################################
#
# VSC_DATE_CAN_SET should be set to "TRUE" if the system permits a
# privileged process to use the "date" utility to set the data and time
# as specified in the XCU4 Issue 2 specification, and to "undefined"
# otherwise.
VSC_DATE_CAN_SET=TRUE

########################################################################
#
# VSC_CHARMAP_FILE and VSC_LOCALEDEF_FILE should be the pathnames
# of files suitable for use with the -f and -i options to the localedef
# utility, respectively, to define the locale named $VSC_LOCALE_NAME.
# VSC_LOCALEDEF_OUTPUT_FTYPE should be "file" if the locale created by
# localedef and referred to by the "name" argument is a regular file,
# "dir" if the locale is a directory, and "other" if the locale is anything
# else.
#
# If the system does not support the localedef utility then the values
# of these four variables should be set to the string "undefined".
VSC_LOCALEDEF_OUTPUT_FTYPE=file
VSC_LOCALE_NAME=lunatic
VSC_CHARMAP_FILE=/usr/src/GA/vsc/Lib/Data/POSIX/lunatext
VSC_LOCALEDEF_FILE=/usr/src/GA/vsc/Lib/Data/POSIX/lunatext.src

########################################################################
#
# VSC_TRAP_NUMERIC_SIGS should be set to "TRUE" if the system claims
# support for the use of numeric values for signals in the shell’s
# "trap" command, and to "undefined" otherwise.
VSC_TRAP_NUMERIC_SIGS=TRUE

########################################################################
#
# VSC_IPC_KEY should be set to the base of a range of 16 key values
# that may be used when creating semaphores, shared memory segments
# or message queues. This value may be specified in octal (0####),
# decimal, or hex (0x####) format.
VSC_IPC_KEY=0xbeefbeef

########################################################################
#
# The VSC Test suite requires three numeric user IDs and three
# numeric group IDs that are not associated with any privileges.
# These IDs should not be the user or group IDs of the tester or
# any of the tester's supplementary groups.  The first pair of
# ID's is used under two different names.
VSC_OTHER_UID=8991
VSC_OTHER_GID=891
VSC_UID0=8991
VSC_UID1=8992
VSC_UID2=8993
VSC_GID0=891
VSC_GID1=892
VSC_GID2=893

########################################################################
#
# The VSC Test suite requires the name and UID of another user:
VSC_OTHER_USER_NAME=dave
VSC_OTHER_USER_UID=310

########################################################################
#
# The symbol VSC_ASA_1 is the string that your system's "asa" writes
# when it encounters a line that starts with a '1'.  The default value
# is a single ASCII Formfeed ( 14).  If necessary, edit the following
# line to change the default.
VSC_ASA_1=^L

########################################################################
#
# The symbol VSC_ASA_P is the string that your system's "asa" writes
# when it encounters a line that starts with a '+'.  The default value
# is a single ASCII Backspace ( 10).  If necessary, edit the following
# line to change the default.
VSC_ASA_P=^H

########################################################################
#
# The VSC_DOT_O_FILE names a file that is an unstripped object file.
# If tcm.o exists in $TET_ROOT/src/posix_c/api, then this should be used.
# Otherwise, you must supply a filename if you support either the
# POSIX.2 Software Development option or the XPG4 Development utilities.
# If you support neither, you can leave this blank.
VSC_DOT_O_FILE=/usr/src/GA/vsc/Lib/Data/POSIX/tcm.o

########################################################################
#
# The VSC_EXE_FILE names a file that is an unstripped executable
# binary. If tcc exists in $TET_ROOT/bin, then this should be used.
# Otherwise, you must supply a filename if you support either the
# POSIX.2 Software Development option or the XPG4 Development utilities.
# If you support neither, you can leave this blank.
VSC_EXE_FILE=/usr/src/GA/vsc/Lib/Data/POSIX/tcc

########################################################################
#
```

```
# If the implementation supports the fort77 utility (determined by
# calling "getconf POSIX2_FORT_DEV"), then VSC_FORT77_DLD is the
# location (absolute pathname) of the directory in which the default
# libraries used by fort77 are searched.  If there is no such directory
# or if fort77 is not supported, the value should be "none".
VSC_FORT77_DLD=/usr/lib

########################################################################
#
# VSC_TALKNAME names another user that can be used for the "talk"
# test case.  It must be a user configured in the user database; that
# is the only requirement.
VSC_TALKNAME=dave

########################################################################
#
# VSC_PS_SUPPORT indicates whether or not your system supports
# the "ps" utility.  If you support the POSIX.2 UPE, then VSC_PS_SUPPORT
# must be set to "TRUE".  Otherwise it should be set to either "TRUE"
# or "undefined", depending on whether or not you have "ps".
VSC_PS_SUPPORT=TRUE

########################################################################
#
# VSC_PRINTF_FLOAT_SUPPORT indicates whether or not your system's
# printf utility supports floating point conversions such as %e, %f and %g.
# It should be set to either "TRUE" or "undefined", depending on whether
# or not your printf supports these conversions.
VSC_PRINTF_FLOAT_SUPPORT=TRUE

########################################################################
#
# VSC_TERMIOS_SUPPORT indicates whether or not your system has a line
# (in the broadest sense, that is, including pseudoterminals) that is
# configured to support the POSIX.1 termios interface.  If the system
# does support such a line, then the value of VSC_TERMIOS_SUPPORT
# should be set to TRUE and the test suite should be run from a login
# session over such a line.  Otherwise the value should be set to
# "undefined".  This value is currently used only in the "stty" tests.
VSC_TERMIOS_SUPPORT=TRUE

########################################################################
#
# VSC_BYTE_SIZE is the number of bits in one byte on the system
# under test
VSC_BYTE_SIZE=8

########################################################################
#
# VSC_LPDEST and VSC_LPNODEST are respectively valid and invalid
# destinations for the "lp" command.
VSC_LPDEST=lp
VSC_LPNODEST=nosuchlp
```

```
########################################################################
#
# The "lpstat" command, the "cancel" command, and the -m, -o, -t and
# -w options to "lp" are marked as Possibly Unsupportable in XPG4.
# So are the "uulog", "uuname", "uupick" and "uuto" utilities, and
# the -j option to the uux utility.
#
# The corresponding variables should be set to "TRUE" if the utility
# or option is supported, and to "undefined" (for the utilities) or
# "unsupported" (for the options) otherwise.
#
VSC_LPSTAT=TRUE
VSC_CANCEL=TRUE
VSC_UULOG=TRUE
VSC_UUNAME=TRUE
VSC_UUPICK=TRUE
VSC_UUTO=TRUE
VSC_LP_MFLAG_SUPPORT=TRUE
VSC_LP_OFLAG_SUPPORT=TRUE
VSC_LP_TFLAG_SUPPORT=TRUE
VSC_LP_WFLAG_SUPPORT=TRUE
VSC_UUX_JFLAG_SUPPORT=TRUE

########################################################################
#
# VSC_MAIL_USER is the local address of $VSC_TESTER.  Almost always,
# these are identical.
VSC_MAIL_USER=vsc0
VSC_MAIL_FILE=vsc0

########################################################################
#
# VSC_CHRDEV and VSC_BLKDEV are YES if the system supports character
# and block terminals respectively, and "undefined" otherwise.
VSC_CHRDEV=YES
VSC_BLKDEV=undefined

########################################################################
#
# VSC_TEST_SUITE_FS and VSC_TEST_SUITE_MOUNT_POINT are the full pathsnames
# of the block device for the filesystem which contains the VSC test
# suite hierarchy and the full pathname of the directory where this
# filesystem is mounted.
VSC_TEST_SUITE_FS=/usr/vsc5
VSC_TEST_SUITE_MOUNT_POINT=/dev/root

########################################################################
#
# VSC_OTHER_FS and VSC_RO_FS are the names of an other file system and
# another, read-only, file system respectively.  They may be "undefined" if
# the implementation under test does not support or provide such objects.
# They should be mounted before running VSC.
VSC_OTHER_FS=/spare1
VSC_RO_FS=/spare2
```

```
#########################################################################
#
# VSC_LFS_MAX_FILE_SIZE defines the implementation maximum file size
# VSC_LFS_FS defines a directory where files of the implementation
# maximum file size can be created.
VSC_LFS_MAX_FILE_SIZE=2147483647
VSC_LFS_FS=/lfs_fs

#########################################################################
#
# VSC_BLK_DEV_FILE and VSC_CHR_DEV_FILE are the names of block
# and character special files.
VSC_CHR_DEV_FILE=/dev/rfd0
VSC_BLK_DEV_FILE=/dev/fd0

#########################################################################
#
# VSC_STREAM_DEV is the name of a STREAMS device.
# A pesudo-terminal device might be a good choice
VSC_STREAM_DEV=/dev/ptmx

#########################################################################
#
# VSC_SXAF is "Y" if ^ and $ ancho in RE subexpressions, and "N"
# otherwise.
VSC_SXAF=Y

#########################################################################
#
# VSC_MV_X_FS and VSC_LN_X_FS are "TRUE if the system can "mv"
# and (hard-link) "ln" files across file systems, respectively, and
# "undefined" otherwise.
VSC_MV_X_FS=TRUE
VSC_LN_X_FS=undefined

#########################################################################
#
# VSC_LINK_DIR_SUPPORT is "TRUE" if the system can link directories
# and "undefined" otherwise.
VSC_LINK_DIR_SUPPORT=TRUE

#########################################################################
#
# VSC_WPERM_TO_MV_DIR is "TRUE" if the system requires write permission
# to "mv" (rename) an existing directory, and "undefined" otherwise.
VSC_WPERM_TO_MV_DIR=undefined

#########################################################################
#
# VSC_NO_MV_FILE_IN_USE is "TRUE" if the system considers it an error
# for a nonprivileged process to "mv" a file or directory that is in
# use by the system or another process, and "undefined" otherwise.
VSC_NO_MV_FILE_IN_USE=undefined
```

```
########################################################################
#
# VSC_LINK_MAX is the correct value of LINK_MAX in the file
# hierarchy under which the tests will be run.
VSC_LINK_MAX=32767

########################################################################
#
# VSC_DU_SZ is the number of bytes consumed by the creation of the
# smallest possible (i.e. 1-byte) nonempty file.  It should be a
# multiple of 512.
VSC_DU_SZ=512

########################################################################
#
# VSC_ICONV_FROM and VSC_ICONV_TO are names of codesets suppported
# by the implementation’s "iconv" utility.  VSC_ICONV_FROMFILE is an
# existing readable file that uses the VSC_ICONV_FROM codeset.
VSC_ICONV_FROM=lunatext
VSC_ICONV_TO=ISO8859-1
VSC_ICONV_FROMFILE=/usr/src/GA/vsc/Lib/Data/POSIX/lunatics


########################################################################
#
# The following are miscellaneous variables used by the UUCP tests.
#
_L_SYS=clarke
_R_SYS=jupiter
_R_SYS2=clarke
_L_PUBDIR=/usr/spool/uucppublic
_R_PUBDIR=/usr/spool/uucppublic
VSC_UUCP_XFER_DELAY=75
VSC_UUCP_NOSUCH_SYS=nOsUcH_sYsTeM
M_SPOOL_FILE=/usr/spool/mail/vsc0
VSC_UUCP_JFLAG_SUPPORT=undefined

########################################################################
#
# For branding or certification purposes, the VSC_SPACE variable should
# be set to "SPACE".  However, while you are debugging a system in which
# some utilities do not allow options and their arguments to be separated
# by a space, you can set VSC_SPACE to "NOSPACE".  This will cause the
# tests to omit the space between certain options and their arguments.
# This feature is used only by some of the tests for SCCS utilities.
VSC_SPACE=SPACE
```

8. BUILDING VSC

8.1 BUILDING THE VSC TEST SCRIPTS

1. Log in as the VSC Test User.
2. Change directory to `$TET_ROOT`.
3. Issue the command

   ```
   . ./SOURCE_ME
   ```

   to set the necessary shell variables.
4. Issue the command

   ```
   tcc -b vsc all
   ```

   This builds the VSC test scripts.

Note that this "build" consists simply of making a new link for each test script, with a different extension (`.ex` instead of `.sh`). The entire build should take less than one hour, and on fast systems will take only a few minutes.

9. EXECUTING VSC

9.1 INTRODUCTION

This chapter describes the steps that must be taken to run a test or group of tests. The *tcc* is used to run any of the scenarios defined in the *tet_scen* file. Additionally, the *vscrun* tool can be used to execute a single test case or a set of invocable components within a single test case. The *BuildScen* and *RunCustom* tools can be used to create and execute custom scenarios. Debugging output can be enabled via the *TST_TRACE* environment variable.

9.2 RUNNING ALL TESTS

This section is a list of steps to executing VSC tests. These are the actions that will be performed during a branding run of the VSC. Steps 1 and 2 may be skipped if you have just configured the test suite and are still logged in as the tester.

1. Log in to the system as a nonprivileged user. This should be the same user ID that was used to install and configure the test suite. If you have created a special user account for this purpose, then you should log in to the system using that account.
2. Change the current directory to the location where TET was installed. Type

   ```
   unset TET_ROOT
   . ./SOURCE_ME
   ```

   if the variable TET_NSIG is not set in the current environment, then you will be prompted for its value. This variable should be set to one greater than the highest numbered signal supported by your system.
3. Ensure that your PATH variable is set to a value that includes the directories in which the POSIX.2, XPG4, and/or XCU5 utilities can be found.
4. Change directories to $VSC_ROOT.
5. To run the entire test suite, type:

   ```
   tcc -bec vsc all
   ```

   to build, execute and clean each test case in succession. This is the required command for a branding run.

   The test suite is now running. If a test case hangs then then test suite will not timeout. Instead, you will have to kill the hanging test case by hand. A journal of the test suite run will be left in a directory under the *$VSC_ROOT/results* directory. The various report and analysis tools can then be used to process the journal file. See chapter 10 for information on these tools.

9.3 RUNNING INDIVIDUAL TESTS OR SETS OF TESTS

9.3.1 INTRODUCTION

Although VSC is principally intended as a conformance test suite, it also functions as a development tool. During development it is often necessary to run individual test cases, groups of test cases, or even groups of test assertions. This section describes a number of ways to accomplish this:

1. A scenario to run any number of assertion tests or test cases can be defined in the scenario file *tet_scen*. This special scenario can then be run with the *tcc* utility.
2. The vscrun utility enables the user to easily run a single test case or a set of assertions within a test case.
3. The BuildScen and RunCustom tools can be used to automatically create and execute a custom scenario.

9.3.2 TCC

The scenario file `tet_scen` supplied with the test suite contains a number of different scenarios that can be executed. The "all" scenario is used to run the entire test suite for conformance branding. During development or debugging you may wish to run smaller sets of tests. To do this you would type

```
tcc -bec -y <pattern> vsc
```

where <*pattern*> matches a test name name found in the *$VSC_ROOT/tet_scen* file. For example, to run the chmod test, type:

```
tcc -bec -y chmod vsc
```

See the TET documentation for a further description of scenario files.

9.3.3 VSCRUN

The vscrun tool is useful for running and debugging individual tests. It's syntax has changed somewhat as of VSC 4.1.5. See the vscrun man page for more information.

9.4 DEBUGGING

The VSC tests are designed to provide some debugging output during execution. This feature is not used during conformance branding, but is helpful when the test suite is used as a development tool. Each VSC test purpose starts by executing the command $TST_TRACE. By default this is an empty string. If you want to turn on full debugging you can type the commands

```
TST_TRACE="set -x"
export TST_TRACE
```

and a complete trace of the code executed by the shell will be written to standard error. This is very voluminous, and should only be done when you are running a few individual test cases. You can also track the execution of the test by setting TST_TRACE to other commands. For example,

```
export TST_TRACE='eval echo $VSC_COMMAND_UNDER_TEST: $tet_thistest >&2'
```

will display to standard error the name of the test case, a colon and the name of the test purpose each time a new test purpose is started.

10. REPORTING

10.1 INTRODUCTION

You can use the VSC reporting program to format reports from the results of the building and execution stages. VSC provides a report format similar to that produced by the NIST PCTS, and a report facility similar to that produced by VSX4.

10.2 THE vrpt REPORTING PROGRAM

VSC uses a variation of the VSX reporting program, `vrpt`, to format the results in the VSC journal files generated by the building and execution stages. When you use `vrpt`, the environment variable **PATH** must be correctly set so that commands can be executed. The reporting program and its subsidiary programs are located in the directory `Bin` below the `$TET_ROOT/vsc` directory. Include this directory in your `PATH`.

See the vrpt man page for information on running vrpt.

11. INTERPRETING VSC RESULTS

11.1 INTRODUCTION

To interpret the results of the VSC tests, you must review each test and the test results from your system. To review the test results, you must generate a report from the VSC journal file using `vrpt`, as explained in the chapter entitled ‘‘REPORTING’’. All the test output from the journal is included in reports generated with `vrpt` for tests whose result was anything other than PASS.

11.2 TEST RESULTS

11.2.1 Failed

The test shell script for failed shell and utility tests is located in the appropriate testset directory, in the directory hierarchy starting from `tset`. To analyse the results of these tests fully, you must be able to examine the test source code to understand the test strategy and identify the conditions which led to the test failure. This level of expertise requires the skills of a specialist; non-specialist staff should not attempt to interpret these results.

11.2.2 Uninitiated or Unresolved

Uninitiated means that the particular test in question did not start to execute.

Unresolved means that the test started but did not reach the point where the test was able to report success or failure.

When a test is reported as `uninitiated` or `unresolved` you must identify the reason why the test was not completed. Examination of the journal file or the output of `vrpt` may indicate the cause.

11.2.3 Unsupported

Unsupported means that an optional feature within the XPG is not available or supported in the implementation under test.

For example, the Development Utilities are optional. VSX will recognise that they are unsupported on a particular system and report this.

11.2.4 Untested

This occurs because there is no test written to check a particular feature, or an optional facility needed to perform a test is not available on the system.

For example, it is not possible to check that word expansion in a shell occurs in the same environment as that in which the shell is executed.

11.2.5 Inspect

These are tests for which it is not possible or practical to verify the output automatically. The relevant output is in the journal file, and in both the summary and detail output of the vrpt report.

11.2.6 Unapproved Assertion

These tests were not performed because the behavior required by the assertion does not appear to be supported by the specification. All such tests will eventually be removed when a final assertion specification is available.

11.2.7 Not Implemented

These resultd describe assertions that are currently untested. In VSC, there are two reasons for such a result.

1. This release of VSC does not test localization.
2. Certain utility tests in this release of VSC are incomplete.

11.2.8 Not in Use

The assertion number is not being used. These assertions will be removed from a future relase of VSC.

11.2.9 Succeeded

This means the test has been executed correctly and to completion without any kind of problem. The test did not detect any non-compliance with the definitions in XCU5, XPG4 or POSIX.2.

12. SUPPORT and INTERPRETATION REQUESTS

12.1 INTRODUCTION

The Open Group offers a support service for this distribution. Please contact your nearest Open Group sales office for information on licensing support.

12.2 SUPPORT REQUESTS

Support Requests (SRs) are the mechanism used to

- report suspected VSC problems
- suggest VSC enhancements
- request general information about VSC

Procedures and forms for submitting SRs are available at the The Open Group World Wide Web site via the URL

```
http://www.opengroup.org/testing/support/
```

12.3 INTERPRETATION REQUESTS

Interpretation Requests (IRs) are the mechanism used to officially apply for

- permanent interpretations (PINs) related to the XCU specification VSC is based on
- temporary interpretations (TINs) related to the XCU specification VSC is based on
- waivers for test failures caused by VSC test suite deficiencies (TSDs)
- waivers for test failures caused by minor system faults (MSFs) in the application being tested

Procedures and forms for submitting IRs are available at the The Open Group World Wide Web site using the URL

```
http://www.opengroup.org/interps/interpform.html
```

A database of all VSC IRs where there are rulings to date is available on the World Wide Web through the following URL

```
http://www.opengroup.org/interps
```

CONTENTS

1. FOREWORD 1
  1.1 VSC DOCUMENTATION 1
    1.1.1 Part 1: VSC User Guide 1
    1.1.2 Part 2: VSC Installation Guide 2

2. VSC TERMINOLOGY 5
  2.1 INTRODUCTION 5
  2.2 STAGES OF VSC 5
    2.2.1 Stage 1: Preparation 5
    2.2.2 Stage 2: Installation 5
    2.2.3 Stage 3: Configuration 5
    2.2.4 Stage 4: Building 5
    2.2.5 Stage 5: Execution 5
    2.2.6 Stage 6: Reporting 5
    2.2.7 Stage 7: Interpreting VSC Results 5
  2.3 STRUCTURE 6
    2.3.1 Section 6
    2.3.2 Testset 6
    2.3.3 Test 6
  2.4 NAMING CONVENTIONS 6
  2.5 ASSERTION NUMBERING CONVENTIONS 6
  2.6 JOURNAL FILE 7

3. VSC DIRECTORY STRUCTURE 8
  3.1 DIRECTORY STRUCTURE 8
    3.1.1 Introduction 8
    3.1.2 TET Directories 8
    3.1.3 VSC Build Files 8
  3.2 THE vsc DIRECTORY 9
    3.2.1 Binaries: *Bin* 9
    3.2.2 Results: *results* 9
    3.2.3 Source: *Src* 9
    3.2.4 Headers: *Inc* 9
    3.2.5 Library: *Lib* 9
    3.2.6 Scenario Files: *ScenDir* 9
    3.2.7 Shell Functions: *Shlib* 9
    3.2.8 Reporting Tools: *Tools* 9
    3.2.9 Testset: *tset* 9
  3.3 SOURCE DIRECTORY STRUCTURE 9
    3.3.1 Test Suite API: *API* 10
    3.3.2 General Assertion Test Routines: *GA* 10
    3.3.3 Implementation-specific Routines: *ImplSpec* 10
    3.3.4 Interactive Testing Tools: *Interact* 10
    3.3.5 Shell Scripts and Functions: *Shell* 10

4. RESOURCES 11
  4.1 INTRODUCTION 11
  4.2 COMPUTER HARDWARE 11
    4.2.1 Disk Space 11
    4.2.2 Exclusive Use 11

4.2.3 Devices 11
4.3 UTILITIES 11
4.3.1 POSIX Shell 11
4.3.2 `make` 12
4.3.3 Compiler 12
4.3.4 Library Archiver 12
4.3.5 Miscellaneous Utilities 12
4.4 TIME 12
4.5 SKILLS 12
4.5.1 Using VSC 12
4.5.2 Experience With the TET 12
4.5.3 Interpreting Results 13

5. PREPARATION 17
5.1 INTRODUCTION 17
5.2 PREPARING YOUR SYSTEM 17
5.2.1 File Space Requirements 17
5.2.2 VSC User and Group Accounts 17
5.3 LOADING THE VSC DISTRIBUTION 18
5.3.1 Unpacking the Distribution File 18

6. INSTALLING VSC 19
6.1 INTRODUCTION 19
6.2 BEFORE RUNNING Build 19
6.2.1 Set Shell Variables 19
6.2.2 Configure the TET. 19
6.2.3 If Necessary, Modify Implementation-Specific Source Files 20
6.3 THE Build DIALOGUE 20
6.3.1 Set Shell Variables 20
6.3.2 Run ‘Build’ 20
6.3.3 Building the VSC Tools and Utilities 21
6.4 CLEANING VSC 21

7. CONFIGURING VSC 22
7.1 INTRODUCTION 22
7.2 CONFIGURATION 22
7.2.1 Introduction 22
7.2.2 Configuration Questions 23
7.3 POST CONFIGURATION 26
7.4 SAMPLE tetexec.cfg FILE 26

8. BUILDING VSC 37
8.1 BUILDING THE VSC TEST SCRIPTS 37

9. EXECUTING VSC 38
9.1 INTRODUCTION 38
9.2 RUNNING ALL TESTS 38
9.3 RUNNING INDIVIDUAL TESTS OR SETS OF TESTS 39
9.3.1 INTRODUCTION 39
9.3.2 TCC 39
9.3.3 VSCRUN 39
9.4 DEBUGGING 39

10. REPORTING 40
10.1 INTRODUCTION 40
10.2 THE vrpt REPORTING PROGRAM 40

11. INTERPRETING VSC RESULTS 41
11.1 INTRODUCTION 41
11.2 TEST RESULTS 41
11.2.1 Failed 41
11.2.2 Uninitiated or Unresolved 41
11.2.3 Unsupported 41
11.2.4 Untested 41
11.2.5 Inspect 41
11.2.6 Unapproved Assertion 41
11.2.7 Not Implemented 42
11.2.8 Not in Use 42
11.2.9 Succeeded 42

12. SUPPORT and INTERPRETATION REQUESTS 43
12.1 INTRODUCTION 43
12.2 SUPPORT REQUESTS 43
12.3 INTERPRETATION REQUESTS 43