X/Open System Verification Suite

VSTH User and Installation Guide

Please note that this manual refers to the complete VSTH
and thus some descriptions are not applicable to VSTHlite

The Open Group

1. FOREWORD

1.1 VSX DOCUMENTATION

The X/Open Verification Suite, known as VSX, enables you to build and execute test programs which assess operating systems for conformance to the standards established in the Single UNIX Specification versions 1 and 2, the X/Open Portability Guide issues 3 and 4, POSIX.1-1990, POSIX.1-1996, FIPS 151-2, and the Common Application Environment Specification — Protocols for X/Open Interworking: XNFS, Issue 1. VSX consists of a number of separate packages. The VSXgen package is combined with one or more add-on test packages to form a test suite which covers some or all of the above specifications.

This guide describes the use of VSXgen together with the following test packages to perform non-XNFS testing:

VSTH 5.2.1 January 2000

Do NOT use this guide if you wish to perform XNFS testing.

VSX uses part of a set of libraries and programs called *TETware*. TETware has its own documentation; this manual contains sufficient information on how to use TETware in conjunction with VSX, but issues pertaining to TETware installation and use of the more extended TETware functionality will require the user to refer to the toolkit manuals. Some test packages can also be used with TET 1.10, a predecessor of TETware. The instructions given in later parts of this document assume the use of TETware. There may be some differences when predecessors of TETware are used.

The VSX User and Installation Guide is in four parts. Part 1 is the VSX User Guide, which gives information about the terminology and structure used in VSX. In addition, the User Guide tells you what resources and facilities you need to use VSX. Part 2 is the VSX Installation Guide, which gives you all the information you need to install and run VSX. It is a good idea to read both parts before you start installing and using VSX. Part 3 gives additional reference information in a series of appendices, including details of VSX support services. Part 4 contains manual pages for various VSX utilities.

The VSX documentation also includes a separate Pseudo-language Specification, which contains details of the test locales that VSX uses.

1.1.1 Part 1: VSX User Guide

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

1.1.2 Part 2: VSX 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 VSX should be carried out by an experienced systems administrator, because of the wide range of implementations which can be verified by VSX.

During the stages of installation, VSX needs detailed information about your system. On a fully compliant system the configuration details can be found from the system header files and the Conformance Statement. On other systems you may need to obtain the information from the personnel who implemented the system.

Contents
Each chapter in the Installation Guide corresponds with a stage of installation and use of VSX. An overview of each chapter is provided in the User Guide. Some chapters contain supplementary information for specific test packages in the sections at the end of the chapter. You will need to refer to these sections when the generic part of the chapter indicates that a particular requirement only applies to some test packages, in order to determine if it applies to any of the test packages you are using. These sections may also describe additional procedures or requirements for each test package.

You can simply read the Installation Guide for detailed information about VSX. When you want to install VSX 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 VSX 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 VSX tests.

When you are familiar with VSX, 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.

1.1.3 Part 3: VSX Appendices

The first appendix is a summary of the Action Points from the Installation Guide, which you can use when you are familiar with installing VSX. Other appendices contain the reference information you may need when you are using VSX, including information about the support services for VSX.

1.1.4 Part 4: Manual Pages

The commands used to install, build and execute the test suite and to produce reports are covered at a basic level in the installation guide. However, many of these commands have additional features which are described only in the manual pages.

X/Open System Verification Suite

Part 1: User Guide

VSXgen1.5 December 1999

2. VSX TERMINOLOGY

2.1 INTRODUCTION

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

2.2 STAGES OF VSX

VSX is used in a series of independent stages. When you have run all of the stages, you can read the VSX 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.

2.2.1 Stage 1: Preparation

In this stage, you check your system has enough file space available, add entries to the user and group databases and extract the VSX software from the distribution files for VSXgen and one or more test packages.

2.2.2 Stage 2: Configuration

Next, the VSX configuration script interrogates your system for information and asks you questions on the screen. From this information, VSX generates configuration parameter files, which are used in the later stages of VSX to find out details about your system. Additionally, for some test packages you must configure character encodings for the VSX test locales, and install the test locales using the tools and file formats for your system.

2.2.3 Stage 3: Installation

In this stage, VSX sets up the programs, libraries, include files and file systems which are needed to build and execute the test suite.

2.2.4 Stage 4: Building

The building stage uses the VSX source files and the configuration parameter files to build and install executable test programs. This stage places results in a journal file, which is used by the reporting stage. You can choose to build the whole suite of test programs to test your system for conformance, or selected parts. Building selected parts is known as a partial build. In this stage you can also undo all or part of a previous building stage, for example, when you want to rebuild using a different compiler.

2.2.5 Stage 5: Execution

When you have completed the preceding stages, you can run the tests to verify your system. VSX executes the test programs 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 building and execution journal files. You can also produce a summary report for management information and a report comparing the results from several test executions.

2.2.7 Stage 7: Interpreting VSX Results

Using the report generated by VSX, 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

VSX 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 four-level hierarchy consisting of the following levels:

2.3.1 Section

A section corresponds with the primary source of the definition of an interface. For example, the VSX4 section `lang.C` corresponds with the C language definition in XPG3 Volume 4, while the VSX4 section `POSIX.hdr` corresponds with the header file definitions relating to POSIX.1.

2.3.2 Area

An area groups together test programs with a common theme, and is a sub-division of a section. For example, in the VSX4 section `POSIX.os` (operating system interfaces relating to POSIX), the area `files` holds all of the test programs for interfaces which deal with files and directories.

2.3.3 Testset

A testset is the subdivision of an area. A testset usually relates directly to an interface definition, with two exceptions. Firstly, where a single definition covers several different interfaces, VSX may use separate testsets for each interface. Secondly, as there is no natural division in the XPG3 description of the C language, a testset is used for each discrete aspect of the C language definition.

2.3.4 Test

A test tests a particular statement in a 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 VSX naming convention numbers the tests within a testset executable sequentially, and, excepting the C language tests, corresponds with the test descriptions in the VSX manual. The naming conventions are as follows;

Section
major-section`.`*sub-section*

Area
area-name

Testset directory
testset-name

Where testset directory names start with an `M`, they contain tests for the macro version of the interface.

Testset executable
`T.`*testset-name*

Testset names which end in `_X` indicate tests for X/Open extensions to POSIX interfaces (or a historical derivation from such extensions).

2.5 JOURNAL FILE

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

3. VSX DIRECTORY STRUCTURE

3.1 TOP LEVEL DIRECTORY STRUCTURE

3.1.1 Introduction

There are the following directories in the top level of the VSX directory structure. This section gives a brief description of the top level, and is followed by sections with a detailed explanation of each directory hierarchy. These are described thus, with the VSX directory name following the descriptive directory name.

3.1.2 Binaries: `BIN`

A common location for all the VSX commands you execute, for example, `vrpt`. You should include this directory in the search path for your shell.

3.1.3 Manual: `MAN`

Manual pages for the testsets and VSX programs, and test description and strategy files. The manual pages use the `man` macros package associated with `[nt]roff` for formatting.

3.1.4 Results: `results`

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

3.1.5 Source: `SRC`

The source files for libraries and utilities used to configure, install and build VSX.

3.1.6 Support: `SUPPORT`

A template for error reports and details of how to submit them.

3.1.7 Testroot: `TESTROOT`

A directory containing the TESTROOT directory structure, used by default as the directory to install the executable testsets. You can change the location when you are configuring the parameter files.

3.1.8 Testset: `tset`

This directory hierarchy contains the source files which are used to build the VSX testsets. The testset directory contains section directories.

Section
The section directories are named using the naming conventions explained in the chapter entitled ‘‘VSX Terminology’’. Each section directory contains area directories, below which are the testsets.

For header and C language sections, each testset directory contains a compacted (`L.`) source file and a makefile. For other sections, each testset directory holds the source programs and makefile for the testset.

The Test Case Controller executes the makefile to install the relevant files in the testroot directory.

3.2 SOURCE DIRECTORY STRUCTURE

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

3.2.1 Common: `common`

The source files compiled during the installation stage, which are used to build the executable program `vrpt` and others used by VSX. In addition, this directory includes the source for the libraries used for building the testsets.

3.2.2 Install: `install`

The scripts and associated files used to configure and install VSX.

3.2.3 Subsets: `subsets`

This directory contains subdirectories relating to each available VSX subset. The files for each subset contain information about the subset and the test package it belongs to, used in the configuration and installation stages of VSX.

3.2.4 Library: `LIB`

An empty directory, named `LIB`, which is used for the libraries compiled from the source in the directory `common`.

3.2.5 Include: `INC`

The unique VSX *include* files, used to build the common software and the testsets.

3.2.6 System Include: `SYSINC`

A copy of the *include* files for the system, which are modified during the installation stage to correct any deficiencies. Note that it is important to ensure that this copy reflects any changes made to the system *include* files.

3.3 MANUAL DIRECTORY STRUCTURE

The manual directory contains the following directories.

3.3.1 Common: `common`

The manual entries for the common software elements, which correspond with the directories under the source directory structure.

3.3.2 Testset: `tset`

The manual entries, test descriptions and strategies for each VSX testset. The directory structure follows the naming conventions explained in the chapter entitled ‘‘VSX Terminology’’. The testset entries are `T.` files containing the manual pages (including test descriptions) for formatting with `[nt]roff -man`, and `L.` files containing test descriptions and strategies used by `vrpt`.

3.4 TESTROOT DIRECTORY STRUCTURE

This structure contains the executable programs for each of the testsets installed. The structure under the `tset` subdirectory follows the naming conventions explained in the chapter entitled ‘‘VSX Terminology’’. Each testset entry is an executable program and creates a matching `d.` directory entry which is used to hold any temporary files created while the testset is executing.

The header file and C language testsets contain compacted file entries copied from the source directory structure, as well as the executable drivers. These do not use a `d.` temporary directory.

The `BIN` subdirectory is where utility programs and other binary files used by the executable testsets are placed, and may also contain scripts which must be edited by the user.

The `INC` subdirectory contains include files for use in header file and C language tests.

4. RESOURCES

4.1 INTRODUCTION

VSX 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 VSX.

4.2 COMPUTER HARDWARE

4.2.1 Disk Space

Installation
VSX uses a considerable amount of disk space. This is described in detail in the chapter entitled ‘‘PREPARATION’’.

Building and Execution
To save disk space, you can build and execute selected sections of VSX. See the chapter entitled ‘‘BUILDING VSX’’ for more information.

4.2.2 Exclusive Use

It is recommended that you have exclusive use of the machine when you execute the tests. In particular, the operating system tests 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 VSX you will need the following devices to be available on your machine. These are only required for some subsets.

1. One or more mountable file systems. VSX uses these for three separate purposes, and so three separate device names can be specified if desired. However, one file system is normally used for all three. One of the file systems is used for testing ENOSPC errors, and will be filled up as part the installation process (if needed by a selected subset).
2. A terminal at which a user is logged on while the tests are executing.
3. Two spare ports to test the terminal interfaces.

4.3 UTILITIES

VSX assumes that the following utilities, which are defined in Volume 1 of XPG3, are available on your system. VSX also assumes that the utilities work in the way described in XPG3.

4.3.1 Bourne Shell

The configuration and installation stages use scripts which are written for the Bourne shell, or a similar shell.

4.3.2 `make`

The installation and building stages use `make` files.

4.3.3 Compiler

VSX requires a C compiler with the `-E` option, and a link editor. VSX assumes that when these utilities execute successfully, they will return an exit value of 0 (zero).

4.3.4 Library Archiver

VSX 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 awk

The reporting stage uses `awk` scripts.

4.3.6 Editors

VSX uses the basic editor `ed` and the stream editor `sed`. The implementation of `sed` can be either in the style of System V or BSD. In addition, the configuration and installation stages use the utility `grep` extensively.

4.3.7 File Utilities

To handle files, VSX uses the basic utilities `cp`, `mv` and `rm`. To handle file modes, VSX uses the utilities `chown`, `chgrp` and `chmod` and, for this reason, you must have access to these during the installation phase of VSX.

VSX also uses a variety of other commands and utilities (as described in Volume 1 of XPG3) during execution and uses the text formatter `nroff` to format the documentation.

4.3.8 Null Device

VSX assumes that the file `/dev/null` is readable and writable by all users and behaves as described in XPG3.

4.4 TIME

For an experienced VSX user, the installation and execution of VSXgen and the full set of test packages should take less than 24 hours. A longer period may be needed on slower processors, as the building stage and the C language and header tests make considerable use of the compiler.

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

4.5 SKILLS

4.5.1 Using VSX

To install, build and execute the tests correctly, you must be able to give detailed information about your system to VSX. VSX is not a product which can be loaded and run without assistance. VSX 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 may need to know whether your system generates the `EBUSY` error if you attempt to remove a busy directory. Without this level of information, you may find difficulty in using VSX.

4.5.2 Interpreting Results

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

X/Open System Verification Suite

Part 2: Installation Guide

VSXgen1.5 December 1999
VSTH 5.2.1 January 2000

5. PREPARATION

5.1 INTRODUCTION

Before you install VSX you must first check there is file space available and then create the user accounts. When you are ready to load the VSX distribution, you must unpack the distribution files and then check that the contents were extracted correctly. Finally, you can optionally remove any unwanted parts of the distribution.

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. The amount of space required by each test package is given in the package-specific sections at the end of this chapter. In addition VSXgen itself requires approximately 5 to 8 Mb free space, of which less than 1 Mb is for the `TESTROOT` directory.

Note the following points:

1. The TESTROOT directory may be on a different file system, but it must be one that allows privileged access (e.g., it cannot be on a remote file system where user ID 0 will be mapped to an 'anonymous' user ID).
2. The disk usage is much greater on RISC systems due to the larger size of object and executable files. Where a range is given the higher figure is for a typical RISC system, but some systems have been known to require up to three times this amount.
3. You can reduce the disk usage by installing VSX in smaller segments, for example, one subset at a time.
4. You will need space to hold the reports. Allow 2Mb minimum.
5. The archiver and compiler may also use some temporary file space.
6. Disk usage in `TESTROOT` will be less if the executables use shared libraries.

Action Points

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

5.2.2 VSX User Accounts

You must add one or more group names and one or more user names to the group and user databases on your system. The precise requirements vary between test packages. Refer to the package-specific sections at the end of this chapter for details.

The only requirements common to all packages are the user name `vsx0` and the group name `vsxg0`. The home directory for user `vsx0` must be a subdirectory of your **TET_ROOT** directory. (If you do not have TETware installed, you must first create a directory to be designated as your **TET_ROOT** directory.)

The home directories of users `vsx1` and `vsx2` (if needed) must differ from that of `vsx0`.

The user `vsx0` must have a login shell. The user ID and group ID values chosen must not exceed the value of **INT_MAX** for the system.

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 user `vsx0`. For example, if the setting of the `LANG` environment variable is such that processes have a locale setting other than the C or POSIX locale on entry to `main()` then this would typically be disabled by adding the line

```
unset LANG
```

to the `.profile` for user `vsx0`.

Action Points

1. Create a distinct group entry for `vsxg0` and (if required by one of the test packages) distinct group entries for `vsxg1` and `vsxg2`; usually in the file `/etc/group`.
2. If you do not already have TETware installed, create a directory you wish to designate as your **TET_ROOT** directory.
3. Create a distinct user entry for `vsx0` in group `vsxg0`, with home directory located under your **TET_ROOT**, and with a login shell; usually in the file `/etc/passwd`.
4. Make sure that the user `vsx0` has write permission in the **TET_ROOT** directory.
5. Add `$HOME/BIN` and `$HOME/../bin` to the command search path for user `vsx0`, and include it in the **PATH** environment variable set in the login script for user `vsx0`.
6. Ensure that any extensions enabled by environment variables that would cause non-compliant behaviour are disabled in the login script for user `vsx0`.
7. For some test packages, if the implementation supports supplementary groups, the user `vsx0` should have the maximum number of supplementary groups associated with it. These supplementary groups must exclude the groups `vsxg1` and `vsxg2`. The group ID values chosen must not exceed the value of **INT_MAX** for the system.
8. If required by one of the test packages, create a distinct user entry for `vsx1` in group `vsxg1`, in the password file. The home directory must differ from that of user `vsx0`.
9. If required by one of the test packages, create a distinct user entry for `vsx2` in group `vsxg2`, in the password file. The home directory must differ from that of user `vsx0`.

5.3 LOADING THE VSX DISTRIBUTION

5.3.1 Unpacking the Distribution Files

The sources for VSXgen and each VSX test package are distributed separately as compressed `cpio` or `tar` archives.

When you unpack the distribution files, the contents are installed in a hierarchy which starts from the current working directory. Make sure you log in as the user `vsx0` and unpack the files in the `vsx0` home directory, to ensure that the access permissions and locations for the files are correct.

The recommended tool to use to unpack POSIX `cpio` and `tar` archives is the POSIX.2 standard **pax** command. If this is not available, it is normally possible to use **tar** or **cpio** commands, but the default format handled by these utilities may not be the POSIX formats on some systems. Refer to the relevant man pages on your system to check whether additional options are needed in order to specify the POSIX formats. Also, some versions of **cpio** are known to create intermediate directories with restrictive permissions - this problem should not occur with **pax**.

Action Points

1. Log in to the test system as the user `vsx0`, who must be the owner of all the loaded files.
2. Ensure you are working in the `vsx0` home directory and that you have write permission in that directory.
3. Unpack the distribution files for VSXgen and the test packages you wish to use, using appropriate commands to decompress each file and extract all files from the resulting POSIX `cpio` or `tar` archive, e.g.:

```
zcat ../VSXgen1.5/source.cpio | pax -r -pp -x cpio
zcat ../VSX4/source.cpio | pax -r -pp -x cpio
zcat ../VSX5/source.cpio | pax -r -pp -x cpio
```

5.3.2 Checking the Contents

When you have finished unpacking the files, the following main directories should be in the `vsx0` home directory:

Directory Name	Summary of contents
`BIN`	VSX user commands
`MAN`	VSX user manuals (on-line)
`results`	a directory tree for journal files
`SRC`	general source tree
`SUPPORT`	VSX error reporting macros
`TESTROOT`	executable testsets tree
`tset`	testset source tree

In addition, a number of other files should be in the `vsx0` home directory. The most important ones to look for are the test package release identification files, called *pkg*`rel`*num*, where *pkg* is the package name (e.g. `VSX4`) and *num* is the release number of the package. This is the last file written to each test package archive. Its presence tells you that all the contents of the archive have been read.

Action Points

1. Change to the `vsx0` home directory (using `cd`) and list the directory. Check the expected subdirectories and the release identification files for each test package are there.

   If they are not, check that there were no read errors while the archives were being read and that there is space available on the file system.
2. Check that the release numbers given in the test-package specific Action Points for this chapter correspond with the release identification files.

5.4 REMOVING UNWANTED VSX DATA (OPTIONAL)

When you want to save space on your system and you do not want to install the tests for some sections, you can remove parts of the VSX distribution. Refer to the package-specific sections at the end of this chapter to identify which parts of each test package you might be able to remove.

Action Points

1. Remove any unwanted sections from the directories `tset` and `MAN/tset`.

5.5 VSTH PREPARATION

This section provides supplemental information for users of VSTH.

5.5.1 File Space Requirements

A VSTH source code distribution requires approximately 5 Mb of free space. This is in addition to any space taken by VSXgen, TET, or other test packages.

A VSTH binary distribution requires approximately 130 Mb of free space on a typical RISC system.

5.5.2 VSTH User Accounts

Action Points

1. The VSTH test package requires configuration of the user `vsx0` and the groups `vsxg0.` User `vsx0` must configured to be a member of the group `vsxg0.` If your implementation supports supplementary group ids, you must also configure these supplementary groups for user vsx0.

5.5.3 TET configuration dependencies

TET must be built with POSIX threads support for VSTH use. Please consult the TET release notes for further information on how to build TET with POSIX threads support.

5.5.4 Loading the VSTH Distribution

Action Points

1. The VSTH test package is distributed as a compressed tar archive. If you have downloaded the gzip'd version of VSTH, execute the following commands as user vsx0 from your home directory:

   ```
   gzip -d vsth-<release_number>.tar.gz
   tar -xvf vsth-<release_number>.tar
   ```

2. If you have downloaded the compressed version of VSTH, execute the following commands as user vsx0 from your home directory:

   ```
   uncompress vsth-<release_number>.tar.Z
   tar -xvf vsth-<release_number>.tar
   ```

3. When you have finished unpacking VSTH, you should find the file VSTHrel*num* in the home directory of user vsx0, where *num* is the release number of VSTH. This file identifies the version of VSTH that you have unpacked and that you have unpacked all the contents of the release.

5.5.5 Removing Unwanted VSTH Data (Optional)

All parts of VSTH are required to test a thread implementation for conformance with POSIX96 or UNIX98.

Action Points

1. Header file tests are collected into the `PTHR.hdr` section.
2. Tests of pthread functions and related thread-safe functions are grouped under the `PTHR.os` section. Underneath this directory are further subdirectories corresponding to functional subsets of functions under test.
3. We recommend that you edit your scenario file to control compilation and execution of VSTH subsets.

6. CONFIGURING VSX

6.1 INTRODUCTION

When you receive VSX, the source code is written to run even when your system does not yet conform fully to the specification. The construction of VSX enables it to run on a wide range of implementations, by including a configuration stage.

During the configuration stage, VSX finds out the specific details about your system and uses the information to generate parameter files for the system. VSX finds the information both by interrogating your system and by asking you questions.

Before you start, you must establish an installation directory and find out the information which is needed for configuration. Read this chapter and write the information for your system next to each action point. When you have finished configuring VSX, check the parameter files are correct for your system.

For some subsets you must also install several test locales, or 'pseudo-languages'. Further information is given in the section entitled ''INSTALLING THE PSEUDO-LANGUAGES'' later in this chapter.

6.2 INSTALLATION DIRECTORY

The directory you use to install the testsets may be on a different file system from the distribution directory. The file system must have enough space available for the executable testsets. A default installation directory is provided, named `TESTROOT` in the `vsx0` home directory. If you choose a directory that does not already exist, it will be created by the VSX installation procedure.

For some test packages (as indicated in the package-specific sections at the end of this chapter), if you are testing for conformance to FIPS 151, XPG4 or UNIX the installation directory must support the inheritance of parent directory group ID. The method of achieving this (if it is not the default) is implementation dependent, but will typically be either an option when the file system is created, or a mode setting on the individual directories which support the feature. If the method used involves the setting being inherited by subdirectories when they are created and there is an existing directory hierarchy under the installation directory which is not set up to support the inheritance of parent directory group ID, then you must remove the `tset` subdirectory and everything below it.

It is recommended that you set the environment variable `TET_EXECUTE` to the pathname of the installation directory in your login script. For example, if you are using a Bourne-type shell and the default location for the installation directory, include the lines

```
TET_EXECUTE=$HOME/TESTROOT
export TET_EXECUTE
```

in the `.profile` for the `vsx0` login.

Action Points

1. Choose a directory for the installation of testset executables. The default is `TESTROOT` under the `vsx0` home directory.
2. If you are testing for conformance to FIPS 151, XPG4 or UNIX and one of the VSX test packages you are using requires that the installation directory must support the inheritance of parent directory group ID, and your system implements the inheritance of parent directory group ID by means of a directory mode setting

that is inherited by subdirectories (for example by setting the `S_ISGID` bit on the directory), and there is an existing directory hierarchy under the testroot which does not support this feature, then you must remove it using

```
rm -rf TESTROOT/tset
```

3. Set the environment variable `TET_EXECUTE` in the `vsx0` login script to the pathname of the testroot directory.

6.3 PARAMETERS

6.3.1 Introduction

The shell script included with VSX for configuration interrogates your system and asks you questions on the screen. There are VSX default values which you can use, or you can choose to start with the defaults in a parameter file which has already been set up. A list of the parameter files available is given when the configuration script starts. See also the section entitled ‘‘CREATING PARAMETER FILES’’ at the end of the chapter.

6.3.2 Libraries

When VSX interrogates your system, it searches libraries in an order which ensures that the standard C library is checked last. Give the names of other libraries to search in the order you want VSX to use them. Note that if loadable objects with the same name appear in different libraries, problems may occur when you are compiling if the libraries are searched in the wrong order.

6.4 VSX CONFIGURATION SCRIPT

6.4.1 Introduction

The configuration script finds or requests information about the following subjects. When the configuration script asks a question on the screen, you can use the default value, shown in brackets, or type the answer for your system. The sections on the following pages tell you what information VSX is looking for. The package-specific sections at the end of this chapter may indicate additional requirements or restrictions on the answers you give to some of these questions.

Note that the defaults (shown in parentheses) in the text on the following pages are the VSX defaults. When you use a parameter file which has already been set up, the defaults on the screen will be taken from the parameter file you chose. When you re-run the configuration script later, the defaults are taken from the previous run unless you use a parameter file.

6.4.2 General Information

Parameter File
Choose a parameter file from the list shown on the screen. Alternatively, press RETURN to start with values from a previous run if any, otherwise the VSX defaults.

Mode
This determines which test mode you wish to run. Some or all of the following modes will be offered, depending on which test packages are available:

- `UNIX98` — tests compliance to the Single UNIX Specification, version 2
- `UNIX95` — tests compliance to the Single UNIX Specification, version 1
- `XPG4` — tests compliance to X/Open Portability Guide, Issue 4
- `XPG3` — tests compliance to X/Open Portability Guide, Issue 3

- `POSIX96` — tests compliance to POSIX.1-1996
- `POSIX90` — tests compliance to POSIX.1-1990
- `FIPS` — tests compliance to FIPS 151-2

Subset
A list of the subsets that support the chosen test mode from each available test package is displayed. Enter a space-separated list of the subsets which contain tests you wish to run (default: all subsets that support the chosen test mode).

If only one subset supports the chosen test mode, its name is displayed and the question is not asked.

Name
Your name in the way you want it shown on reports.

Agency
Your agency name, to use on reports.

System
The operating system name and the release number, to use on reports.

Installation Directory
The name of the installation directory for the executable testsets (default: `$HOME/TESTROOT`). You can choose any other directory you want to use, to make the best use of the file space available. The installation directory is also known as the testroot directory.

Machine Speed
The speed of your machine, in the range 1–10, where 1 is very fast and 10 is slow (default: 5). For example, the speed of an average workstation is rated 5 on this scale. It is better to underestimate the speed of your machine than to overestimate it.

The speed rating is used to determine how much time to allow a test before timing out, usually in cases where a test has failed. The speed rating does not affect the execution time for VSX significantly.

Include Files
The system's *include* directories in order of searching (default: `/usr/include`).

C Compiler
The name of the C compiler (default: `c89` or `cc`). Note that you **must** use `c89` for UNIX98 registration runs.

C Compiler Special Command Line Options
This question prompts for special command line options for your C compiler. If you want to compile parts of the suite with special options, you can specify them when you build the parts with the Test Case Controller. The code is not usually optimised.

If the list of system *include* directories specified earlier is not the default for the C compiler, then add `-I/`*directory* options as necessary.

C Compiler Special Link Editor Options
The next question prompts for special link editor options for your compiler. The default options are usually adequate. However, it should be noted that the C compiler special command line options are **not** used on the link command line, and thus some of these special options may need to be repeated here.

Libraries
The location of the library maintenance utilities `ar`, `lorder`, `tsort` and `ranlib`. This is requested when they are not in the user's **PATH**.

Files
The location of the commands to change file ownership, file group ownership and file modes; namely `chown`, `chgrp` and `chmod`. This is requested when they are not in the user's **PATH**.

Additional Libraries
Libraries, other than the C library and specific libraries asked for individually, used by your system for some of the routines (for example `-lmalloc`). Give one library name each time the question is asked.

Note that some questions about specific libraries may be asked **after** this question, if they are only needed by certain subsets.

ANSI `vprintf` Function
Whether the ANSI functions `vprintf()` and `vfprintf()` are supported (y/n, default: `y`). This question is only asked in POSIX and FIPS modes.

6.4.3 Compiler Characteristics and Libraries

VSX uses a series of small C programs to test your compiler and libraries.

6.4.4 Subset-specific Information

Information needed by individual subsets is asked at this point. Refer to the package-specific sections at the end of this chapter for details.

6.4.5 Optional Information

The following questions are only asked if the information is needed by one or more of the subsets you have selected. The package-specific sections at the end of this chapter indicate which questions apply to the subsets and test mode you have selected.

C Compiler Threads Command Line Options
This question prompts for compiler options to be used **instead** of the normal options, when compiling thread-safe programs.

C Compiler Pure Executable Flag
The C compiler option to produce a pure executable (shared text) file. VSX needs this information to create a pure executable program to test the `ETXTBSY` error condition. Many compilers create all executable files in this way, while others require you to use an option such as `-n` explicitly.

Maths Library
The location of the archive library which contains your maths library routines if there is one (default: `-lm`).

File System for ENOSPC Tests
A mountable device to be used for `ENOSPC` tests. This will be initialised to a known nearly-full condition during the installation stage. The filling procedure will take less time if the device to be used has a small capacity. If only a large device is available it is advisable to create a small file system on it if possible.

6.4.6 Running the Configuration Script

When you execute the configuration script and answer the questions, messages appear on the screen which tell you when the script is updating the parameter files.

Action Points

1. Read through the configuration script section and write down any information you will need to use which is different from the defaults.

2. Execute the shell script `config.sh` which is in the `BIN` directory. When you have included this directory in your **PATH**, you can execute the command from any location.
3. Answer the questions which the configuration script asks.

6.5 CHECKING THE PARAMETER FILES

When you have run the configuration script and answered the questions, VSX generates two files in the `SRC` directory. One contains configuration parameters and the other is a header file containing *include* file elements.

6.5.1 Configuration Parameters File

The parameters file, named `vsxparams`, contains the configuration constants for your system which VSX uses during the installation and building stages. The installation stage uses the parameters file to modify makefiles to suit your system and to generate the files `tetbuild.cfg`, `tetexec.cfg` and `tetclean.cfg` which are used by the Test Case Controller. The parameters file contains a set of parameters which are defined in a format suitable for inclusion in a Bourne shell script:

parameter–name="*parameter–value*"

Note that you must include the quotation marks. A description of the use and possible values for the parameter precedes each parameter setting. You can change the values of parameters in the file without re-running the configuration script.

Action Points

1. Check the values in the configuration parameters file `SRC/vsxparams` and edit the *parameter–name* lines if necessary.

6.5.2 Configuration Header File

The source files used in the installation and building stages of VSX will not compile unless your header files contain the definitions used by the source code for the subsets you have selected. The configuration script checks your system header files for these definitions and, when some are missing, creates a header file, named `vsxconfig.h`, with a list of the definitions which are missing from your system.

Note that VSX uses `NSIG` in `signal.h`, which is not in the Single UNIX Specification. However, this is required to find the highest signal number. This must be set to the highest number that a signal can take plus one.

When the configuration script adds a definition to this file, a message appears on the screen.

Definitions
The list may contain missing elements of the following types:

1. Where the value of a defined constant is unlikely to vary between systems, the configuration header file uses the most common value.
2. Where the value of a defined constant is likely to vary between systems, the configuration header file uses a value of −1. You can change the value to one which is more suitable for your system, or leave it as −1. However, VSX only functions correctly when all of these values have been changed to the correct values for your system. Some tests will fail when values are left as −1. For example if your `signal.h` file does not include the signal `SIGABRT`, you can map it onto the signal `SIGIOT` in the configuration header file, by adding the

definition

```
#define SIGABRT SIGIOT
```

3. Where defined constants are missing from the file `limits.h`, the configuration header file uses the minimum acceptable value.
4. Where type definitions are missing the configuration header file will contain a dummy `typedef` statement. You should replace the token `<type>` with the correct type for your system. If a type definition appears in more than one header file, it must be protected against redefinition in the same way in all the headers that define it, otherwise it must be protected against multiple inclusions of the header that defines it. You may also need to move the type definition so that it appears before any declarations that use the type name.
5. Where structure definitions are missing the configuration header file will contain a dummy `struct` statement. You should replace the token `<members>` with the correct structure members for your system.
6. Where `extern` declarations are missing the configuration header file will contain the correct declaration.

Action Points

1. Check the `SRC/vsxconfig.h` file to ensure that the values are correct for your system.
2. Check that the values of all the varying defined constants have been changed from –1 to the values for your system.
3. Check that the values of the other defined constants are correct for your system.
4. Check that all dummy statements have been changed to valid ones.
5. If you re-run `config.sh` at a later time, it will overwrite `SRC/vsxconfig.h`, so make sure you copy it first.

6.5.3 IMPORTANT

When VSX creates the configuration header file with a list of definitions, the indication is that your system does not conform to the specification. If you are unable to give the correct values for the definitions in the file, you may find that some of the VSX sources do not compile and that some of the tests fail.

You must check all the values for the definitions added to this file, to ensure that they are suitable for your system. Incorrect values may cause particular tests to function incorrectly. If this happens, it is much more difficult to ascertain the cause of the error. As this information is not available to all users, you may need assistance from the personnel who implemented your system.

6.6 CREATING PARAMETER FILES

You can create a new parameter file by copying the `vsxparams` file to the directory `install/params.data`. If you re-run the configuration script, you can choose to use any of the files in this directory to provide the default values for your system. The default values are taken from the file `SRC/vsxparams` unless you choose a different parameter file.

6.7 TOP LEVEL MAKEFILE

The VSX configuration script generates a `Makefile` in the `vsx0` home directory. This contains commands to be executed in the installation stage. Many of the commands needed are implementation-specific and so must be configured by the user. Only the commands needed for the subsets and test mode you have selected are placed in the template `Makefile`.

The operations that must be performed by the configured commands are described in the following sections. You need only refer to the descriptions of the targets that appear in the template that has been created by `config.sh`.

The `Makefile` is provided for convenience should the installation stage need to be repeated. However, as the configured commands must be executed as a privileged user, you may, if you prefer, choose not to use the `Makefile` and execute the necessary commands by hand instead.

6.7.1 Privilege Check

The `privchk` target checks that `make` has been executed with the necessary privileges. The default commands assume that these privileges are associated with user ID 0 and use the commands `id` and `grep` to check the current user ID value.

6.7.2 Parent Directory Group ID

The `dirgid` target sets up the testroot directory to support the inheritance of parent directory group ID. The default commands assume that this is done by setting the `S_ISGID` bit on the directory.

6.7.3 Execute Install Script as User `vsx0`

The `install` target executes the VSX installation script `install.sh` with the user ID of user `vsx0`. Since the script expects the environment variable **HOME** to contain the `vsx0` home directory, these commands must ensure that it is set appropriately.

6.7.4 Assign Privileges to `chmog` Program

The `chmogpriv` target gives the program `chmog` the appropriate privilege to change the mode, owner and group of any file and to assign privileges to executable files. The default commands make the program setuid `root`.

6.7.5 Set Up File System for ENOSPC Tests

The `filldisc` target initialises the mountable device to be used for `ENOSPC` tests to a known nearly-full condition. This is done by mounting the device, changing directory to the mount point, and executing the `filldisc.sh` script. The default commands also include a `df` to report the free space remaining after `filldisc.sh` has completed.

6.7.6 Additional Subset-specific Targets

The `Makefile` may also contain additional targets that are specific to the subsets you have selected. Refer to the package-specific sections at the end of this chapter for details.

6.7.7 Non-configured Commands

The top level `Makefile` may also perform some of the following operations, using commands that do not need to be configured by the user:

- Creates the testroot directory if it does not already exist.

- Additional subset-specific operations.

Action Points

1. Edit the `Makefile` in the `vsx0` home directory.
2. Configure the implementation-specific installation commands correctly for your system.
3. If you re-run `config.sh` at a later time, it will overwrite `Makefile`, so make sure you copy it first.

6.8 USER-SUPPLIED INTERFACE ROUTINES

6.8.1 Introduction

Depending on the test mode selected, VSX needs to use a variety of functions which are not in the corresponding specification in order to set up the conditions required to execute tests of system interfaces which are. Since this functionality can be defined in any way a system implementor requires, VSX has a file which needs to be edited to define these functions prior to the compilation of the test suite.

For example, VSX needs to obtain *appropriate privileges* in many tests. Since the means of obtaining these privileges is not specified in the Single UNIX Specification, it is configurable in the file `SRC/userintf.c`.

As supplied with VSX, these routines make use of system interfaces commonly found on many systems. The file contains sensible defaults and if, after reviewing these, you decide that they are not appropriate for your system, you should modify the routines as necessary before the VSX test suite is installed.

Only the routines needed for the subsets and test mode you have selected are placed in the template `userintf.c` file. Descriptions of the individual interfaces may be found below. You need only refer to the descriptions of the routines that appear in the template that has been created by `config.sh`.

6.8.2 `setprv()`

`setprv()` provides the current process with *appropriate privileges*. The argument specifies what privilege is being requested from the following set:

`PRV_SETID`	to perform privileged `setuid()` and `setgid()` calls
`PRV_MOUNT`	to mount and unmount file systems
`PRV_LINKDIR`	to create and remove links to directories
`PRV_ACCESS`	to gain unrestricted access to files
`PRV_CHOWN`	to perform privileged `chown()` and `chmod()` calls
`PRV_SETGRPS`	to set supplementary group IDs
`PRV_NEWROOT`	to set the root directory of the process
`PRV_KILL`	to perform privileged `kill()` calls
`PRV_DEVICE`	to access device files
`PRV_ASSIGN`	to assign privileges to an executable file
`PRV_IPC`	to allow unrestricted IPC access

PRV_NICE	to perform privileged nice() calls
PRV_ULIMIT	to perform privileged setlimit() or ulimit() calls
PRV_LIMITS	to perform privileged setrlimit() calls
PRV_MEMLOCK	to obtain memory locking privileges
PRV_SETTIME	to set (real-time) clocks
PRV_SETRTSCHED	to set (real-time) process scheduling parameters
PRV_GETRTSCHED	to get privileged (real-time) process scheduling parameters
PRV_SETTHRSCHED	to set threads scheduling parameters
PRV_GETTHRSCHED	to get privileged threads scheduling parameters

The mapping of this privilege set to the set of privileges on the implementation may not be one-to-one. If distinction between individual privileges is not considered important, then the simplest mapping is to give the calling process all possible privileges on each call to setprv().
setprv() returns 0 for success, –1 for failure.

6.8.3 unsetprv()

unsetprv() removes the specified privilege from the current process. The argument specifies what privilege is to be removed; the values are the same as those used with setprv().
unsetprv() returns 0 for success, –1 for failure. (If the process already does not have the privilege, this is considered success).

6.8.4 prv_assign()

prv_assign() assigns *appropriate privileges* to an executable file. The arguments are the file name and a zero-terminated array of the privileges to be assigned. Privilege values are the same as those used with setprv(). If a process with effective user ID of root automatically has the requested privilege, no action is necessary. When the file is executed it will make calls to setprv() to activate the assigned privileges. If privileges assigned with prv_assign() are automatically active when the file is executed, then setprv() should just check that the requested privilege is in effect. prv_assign() is only called after setprv(PRV_ASSIGN) since assigning privilege is a privileged operation.
prv_assign() returns 0 for success, –1 for failure.

6.8.5 mnt_rw()

mnt_rw() mounts the file system specified by *spec* on to the directory *dir* for reading and writing.
mnt_rw() returns 0 for success, –1 for failure.

6.8.6 mnt_ro()

mnt_ro() mounts the file system specified by *spec* on to the directory *dir* for reading only.
mnt_ro() returns 0 for success, –1 for failure.

6.8.7 unmnt()

`unmnt()` unmounts the file system specified by *spec* from the directory *dir* where it has previously been mounted.
`unmnt()` returns 0 for success, –1 for failure.

6.8.8 openctl()

`openctl()` opens the terminal device *spec* with the specified *flags*. This then becomes the controlling terminal for the current process.
`openctl()` returns the open file descriptor for the terminal, or –1 on failure.

6.8.9 openpty()

`openpty()` opens the master and slave sides of a pseudo-terminal. Both device names are supplied, but if master pseudo-terminals are obtained from a clone device such as `/dev/ptmx` then the slave name is a dummy which must be overwritten with the real device name. An argument specifies whether the slave must be opened as a controlling terminal by calling `openctl()` instead of plain `open()`. Both devices must be opened for reading and writing. On many systems master devices can only be opened once, giving EBUSY for example on subsequent opens. If `openpty()` encounters a busy master device, it must simply open the slave. This routine is only called if pseudo-terminals are being used for terminal testing.
`openpty()` returns 0 for success or –1 for failure.

6.8.10 ptygetattr()

`ptygetattr()` obtains terminal attributes for the slave pseudo-terminal corresponding to the master device addressed by a specified file descriptor. This routine is only called if pseudo-terminals are being used for terminal testing.
`ptygetattr()` returns 0 for success or –1 for failure.

6.8.11 newroot()

`newroot()` causes *path* to become the root directory for the current process. `newroot()` is only called after `setprv(PRV_NEWROOT)` since setting the root directory is usually a privileged operation.
`newroot()` returns 0 for success, –1 for failure.

6.8.12 setlimit()

`setlimit()` is used to reduce the output file size limit for the process. The argument is the value to be set in units of 512 byte blocks, and will always be greater than or equal to the value set for the **VSX_ULIMIT_BLKS** parameter.
`setlimit()` returns the new limit, or –1 for failure.

6.8.13 pathdepth()

`pathdepth()` is used to determine the depth of pathnames beginning with `//` below the root directory of the process. The default code supplied is suitable for systems which use `//` to refer to a directory level above the root; i.e., when the root directory is referred to by `//`*somename.*

6.8.14 Additional Subset-specific Routines

The package-specific sections at the end of this chapter describe any additional routines that may have been added to `userintf.c` for the subsets you have selected.

Action Points

1. Review the file `SRC/userintf.c` and identify if it needs to be modified.
2. Modify the file to meet your system's requirements.
3. If you re-run `config.sh` at a later time, it will overwrite `SRC/userintf.c` with the default version, so make sure you copy it first.

6.9 INSTALLING THE PSEUDO-LANGUAGES

6.9.1 Introduction

The internationalisation elements of VSX make use of a set of 'pseudo-languages' which must be installed, if they are needed by one or more test packages, using the appropriate tools for your system. Refer to the package-specific sections at the end of this chapter to determine whether the pseudo-languages are needed for the subsets and test mode you have selected.

The character encodings for additional characters in these languages must be chosen so as not to conflict with the encodings for the portable character set on your system.

Detailed information about the VSX pseudo-languages is contained in a separate document, called the *VSX Pseudo-language Specification*.

6.9.2 Configuring the Character Encodings

Each pseudo-language contains both the portable character set and the control character set with their normal encodings, plus a number of additional characters. The character encodings for the additional characters are specified in the file `SRC/INC/pslcodes.h`, which you must modify if the default encodings conflict with the encodings for the portable character set or the control character set on your system. The default encodings are shown in the section entitled ''PSEUDO-LANGUAGE DEFINITION'' in the VSX Pseudo-language Specification. They are suitable for systems where the characters of the portable character set and control character set have values in the range 0x0 to 0x7f (as in the ASCII character set).

The character encodings for the control character set are specified in the file `SRC/INC/ctrlcodes.h`. You must modify this file if the default encodings are not correct for your system. The default encodings are those of the control characters in the ASCII character set. The file contains two sets of definitions, one for the C locale and one for the pseudo-languages, in case the C locale definitions are not appropriate for use in the pseudo-languages. Any control characters that are not supported by your system should be defined with the value zero. The C locale definitions for the characters DEL, NAK, ETX, FS, DC1, DC3, EOT, VT and SUB must not be zero, as they are used by the terminal interface tests.

The character encodings specified in these two files will be compiled into the VSX libraries when the libraries are built. Therefore you must configure these files before proceeding to the installation stage, even though the languages themselves will not be used until the execution stage.

6.9.3 Installing the Languages

There are five separate pseudo-languages, with the locale names `VSX4L1`, `VSX4L2`, `VSX4L3`, `VSX4L3@dict` and `VSX4L0`. The definitions of the language elements (shift and classification tables, language information tables and collating sequences) are contained in the section entitled ''PSEUDO-LANGUAGE DEFINITION'' in the VSX Pseudo-language Specification. If you execute tests before installing the pseudo-languages, then tests which require a pseudo-language will give a result of

`unresolved` or `uninitiated`.

If your system supports the `localedef` utility, and has ASCII or EBCDIC character encodings, you should be able to use the files supplied in `SUPPORT/psldefs` with little or no modification to install the pseudo-languages. If your system does not support `localedef` you may find that you can adapt these files to the file format required by your system. Note that the character encodings given in the `CHARMAP` entries must match the encodings you have configured in `SRC/INC/ctrlcodes.h` and `SRC/INC/pslcodes.h`. Also, since EBCDIC character encodings vary, you may need to alter some of the encodings for the portable character set. Comments in the file suggest alternative encodings.

Action Points

1. Check the package-specific action points for this chapter to see whether the pseudo-languages are needed for the subsets and test mode you have selected. If they are not needed, skip the following action points.
2. If your system does not use the ASCII character set, modify the file `SRC/INC/ctrlcodes.h` so that it contains the correct control character encodings for your system. If your system uses an EBCDIC character set, copy the file `SRC/INC/ctrlebcdic.h` to `SRC/INC/ctrlcodes.h` first.
3. Check that the character encodings specified in the file `SRC/INC/pslcodes.h` are suitable for your system, and modify the file if they are not. If your system uses an EBCDIC character set, copy the file `SRC/INC/pslebcdic.h` to `SRC/INC/pslcodes.h` first.
4. Install the VSX pseudo-languages using the appropriate tools for your system. You may be able to make use of the files in `SUPPORT/psldefs` to do this.

6.10 WIDE CHARACTER CONFIGURATION FILE

6.10.1 Introduction

Some test packages contain tests for wide character and multibyte character interfaces, which obtain information about the wide characters and multibyte characters supported on the system from a configuration file, `SRC/wchars.cfg`. Refer to the package-specific sections at the end of this chapter to see if this file is needed for the subsets and test mode you have selected. If it is needed, you must enter the required information in this file before proceeding to the installation stage, as the install script compiles the information into a binary file for use in the tests.

An example file suitable for use on systems which only support single-byte characters is provided with VSX. To use it simply copy `SRC/wc_nosup.cfg` to `SRC/wchars.cfg`.

The wide character configuration file contains information about two locales. One locale is often required to exhibit the opposite behaviour of the other, so while one of the locales can be selected from the locales already present on the system, it will usually be necessary to install the other, as it will have strange properties not found in a normal locale. The locales chosen must support non-state-dependent encodings. Also, VSX tests only multibyte characters of 2, 3 and 4 bytes. The locales chosen must support at least one of these encodings.

The first two parameters in the file contain the names of the two locales. The remaining parameters contain lists of characters with specified attributes. These characters may be entered either as wide character numeric values or as multibyte sequences in quotes.

Numeric values may only be used if conversion to a multibyte sequence using `wctomb()` will not generate a shift sequence. The usual convention of a leading `0x` for hexadecimal and leading `0` for octal are used (otherwise decimal). In quoted strings the usual C language string literal conventions are used, including `"\x"` hexadecimal escapes. Where a multibyte character requires a lead sequence to introduce the encoding used, it must be entered as two adjacent quoted strings so that the byte count of the character itself will be correct.

Comments preceding each parameter specify the required attributes of the characters listed in the parameter. Where no character with those attributes exists, enter either `0` or `""`.

Note that the file does not contain parameters to specify printing and non-printing characters. The tests assume that all graphic and space characters are printing characters, and all control characters are non-printing characters.

6.10.2 Example Wide Character Configuration File

On the following pages is an example wide character configuration file based on a draft of the ISO 10646 character set. This character set supports three and four byte multibyte encodings introduced by one or two SGCI (single graphic character introducer) control characters (`"\x97"`).

Long lines in this example have been folded for formatting purposes, marked by a \ character on the end of the line. These lines must be entered as a single line in the configuration file.

Action Points

1. If the wide character locales are needed for the subsets and test mode you have selected, update the file `SRC/wchars.cfg` with information about wide characters and multibyte characters supported on your system. If your system only supports single-byte characters, simply copy `SRC/wc_nosup.cfg` to `SRC/wchars.cfg`.
2. If you re-run `config.sh` at a later time, it will overwrite `SRC/wchars.cfg`, so make sure you copy it first.
3. Install additional locales if necessary to reflect the information in the file.
4. If you change `SRC/wchars.cfg` at a later stage there is no need to repeat the installation stage or to rebuild any testsets. Simply change directory to `SRC/common/wchars` and type `make`. This will update the binary file used by the tests from the new information in `wchars.cfg`.

```
#  This file is a pro forma wide character configuration parameter file.
#  Wide characters may be entered as a numeric value (with a leading 0
#  indicating octal or a leading 0x indicating hexadecimal, otherwise
#  decimal), or as the equivalent multibyte sequence in double quotes
#  (using 'C' string literal conventions).  If a character needs a lead
#  sequence to introduce it, it must be specified as two double quoted
#  strings ("lead_seq" "character") so that the byte count of the
#  character itself will be correct.  Numeric values may only be used
#  if conversion by wctomb() will not generate a shift sequence.
#
#  Where no wide character with the specified attributes exists, enter 0
#  or "".  White space in parameter values is ignored.
#
#  Wide character entries are separated by commas; groups of wide characters
#  are separated by semi-colons.

#Locale names to which the values in the following parameters apply.
LOCALE1=VSXWCLOC1
LOCALE2=VSXWCLOC2

#Characters for which iswcntrl(), etc. are true (ISWCNTRL, etc.) or
#false (NOTCNTRL, etc.) for LOCALE1, and vice versa for LOCALE2.
#It is assumed that all graphic and space characters are printing
#characters, and all control characters are non-printing characters.
#Each parameter must contain a comma separated list of three wide
#characters which correspond to 2,3 and 4 byte multibyte characters
#respectively.
ISWCNTRL=0, "\x97""  \x81", "\x97\x97""   \x81"
NOTCNTRL=0, "\x97""  \x94", "\x97\x97""   \x94"
ISWGRAPH=0, "\x97""  ~", "\x97\x97""   ~"
NOTGRAPH=0, "\x97""  &", "\x97\x97""   &"
ISWLOWER=0, "\x97""  a", "\x97\x97""   a"
NOTLOWER=0, "\x97""  A", "\x97\x97""   A"
ISWPUNCT=0, "\x97""  !", "\x97\x97""   !"
NOTPUNCT=0, "\x97""  #", "\x97\x97""   #"
ISWSPACE=0, "\x97""  @", "\x97\x97""   @"
NOTSPACE=0, "\x97""   ", "\x97\x97""    "
ISWUPPER=0, "\x97""  A", "\x97\x97""   A"
NOTUPPER=0, "\x97""  a", "\x97\x97""   a"

#Lowercase/uppercase character pairs.  In LOCALE1 the first character
#in each pair is lowercase, the second is the corresponding uppercase
#character.  In LOCALE2 the first is uppercase and the second is
#the corresponding lowercase character.
#This parameter must contain a semi-colon separated list of three pairs
#of comma separated wide characters with the first character in each pair
#corresponding to a 2, 3 and 4 byte multibyte character respectively.
TOWUPPER=0,0; "\x97""  a", "\x97""  A"; "\x97\x97""   a", "\x97\x97""   A"

#Uppercase/lowercase character pairs.  In LOCALE1 the first character
#in each pair is uppercase, the second is the corresponding lowercase
#character.  In LOCALE2 the first is lowercase and the second is
#the corresponding uppercase character.
#This parameter must contain a semi-colon separated list of three pairs
#of comma separated wide characters with the first character in each pair
#corresponding to a 2, 3 and 4 byte multibyte character respectively.
TOWLOWER=0,0; "\x97""  A", "\x97""  a"; "\x97\x97""   A", "\x97\x97""   a"
```

```
#Additional properties for use with wctype().
#This parameter must contain a comma separated list of all property
#names supported by wctype() other than the standard ones
#(alnum, etc.).
PROPERTIES=cyrillic, greek

#Characters for which the above properties are true (ISW<property>)
#or false (NOT<property>) for LOCALE1, and vice versa for LOCALE2.
#One ISW<property> and one NOT<property> parameter should be
#defined for each property name in PROPERTIES, with <property>
#in the parameter names replaced by the real property name.
#Each parameter must contain a comma separated list of three wide
#characters which correspond to 2,3 and 4 byte multibyte characters
#respectively.
ISWCYRILLIC=0, "\x97"" \x28\x60", "\x97\x97""  \x28\x60"
NOTCYRILLIC=0, "\x97"" \x2a\x22", "\x97\x97""  \x2a\x22"
ISWGREEK=0, "\x97"" \x2a\x22", "\x97\x97""  \x2a\x22"
NOTGREEK=0, "\x97"" \x28\x60", "\x97\x97""  \x28\x60"

#Characters with specified printing widths in LOCALE1.  The wide
#characters in WIDTH<x> have a printing width of <x>.
#Each parameter must contain a comma separated list of three wide
#characters which correspond to 2,3 and 4 byte multibyte characters
#respectively.  If the system does not support any printing
#characters of a particular width, enter 0 or "" for all three
#wide characters in the corresponding parameter.
WIDTH0=0, "\x97""  \x82", "\x97\x97""   \x82"
WIDTH1=0, "\x97""  \x83", "\x97\x97""   \x83"
WIDTH2=0, "\x97""  \x84", "\x97\x97""   \x84"
WIDTH3=0, "\x97""  \x85", "\x97\x97""   \x85"

#Decimal point character for LOCALE1.
#This parameter should contain a multibyte character of more than
#one byte, if supported.
DECPOINT1="\x97""  ."

#Decimal point character for LOCALE2.
#This parameter should contain a multibyte character of more than
#one byte, if supported, with a different value to DECPOINT1.
DECPOINT2="\x97""  , "

#A character in LOCALE1 which corresponds to a multibyte sequence
#of length MB_CUR_MAX.
MBCURMAX1="\x97\x97""   a"

#A character in LOCALE2 which corresponds to a multibyte sequence
#of length MB_CUR_MAX.
MBCURMAX2="\x97\x97""   a"

#An invalid multibyte sequence.
#This parameter must contain a string in double quotes.
INVALID_MB="\x97"

#A wide character value which does not correspond to a multibyte
#character.
#This parameter must contain an integer value.
INVALID_WC=999999
```

```
#  The following parameters are not used in any tests if wcscoll() and
#  wcsxfrm() are not supported.  In this case "all zero" values should
#  be used to satisfy the config file compiler.  (They can be copied
#  from wc_nosup.cfg).

#Characters which are ignored when collating strings.
#This parameter must contain a comma separated list of three wide
#characters which correspond to 2,3 and 4 byte multibyte characters
#respectively.
DONTCARE=0, "\x97""  \x86", "\x97\x97""   \x86"

#Character pairs which differ in primary collation order.  In
#each pair the first character collates before the second in
#LOCALE1, and vice versa in LOCALE2.
#This parameter must contain a semi-colon separated list of ten
#pairs of comma separated wide characters which correspond to
#multibyte characters with the following numbers of bytes: 1, 1;
#1, 2; 1, 3; 1, 4; 2, 2; 2, 3; 2, 4; 3, 3; 3, 4; 4, 4.
PRIMARY="\x87", "\x88";0,0; "\x87", "\x97""  \x88"; "\x87", "\x97\x97"" \
\x88";0,0;0,0;0,0; "\x97""  \x87", "\x97""  \x88"; "\x97""  \x87", "\x97\x97"" \
\x88"; "\x97\x97""   \x87", "\x97\x97""   \x88"

#Character pairs which have the same primary collation order
#but differ in secondary collation order.  In each pair the first
#character collates before the second in LOCALE1, and vice versa
#in LOCALE2.
#This parameter must contain a semi-colon separated list of ten
#pairs of comma separated wide characters which correspond to
#multibyte characters with the following numbers of bytes: 1, 1;
#1, 2; 1, 3; 1, 4; 2, 2; 2, 3; 2, 4; 3, 3; 3, 4; 4, 4.
SECONDARY="\x89", "\x8a";0,0; "\x89", "\x97""  \x8a"; "\x89", "\x97\x97"" \
\x8a";0,0;0,0;0,0; "\x97""  \x89", "\x97""  \x8a"; "\x97""  \x89", "\x97\x97"" \
\x8a"; "\x97\x97""   \x89", "\x97\x97""   \x8a"

#Characters which collate as 2-1 mappings, with different primary
#order.  In each set of three characters the first two together
#form a collation element which collates before the third character
#in LOCALE1, and vice versa in LOCALE2.
#This parameter must contain a semi-colon separated list of six
#triplets of comma separated wide characters which correspond to
#multibyte characters with the following numbers of bytes: 1, 1, 2;
#2, 3, 1; 2, 1, 3; 4, 2, 3; 1, 3, 3; 3, 4, 4.
PRIMARY2_1=0,0,0;0,0,0;0,0,0;0,0,0; "\x8b", "\x97""  \x8c", "\x97""  \x8d"; \
"\x97""  \x8b", "\x97\x97""   \x8c", "\x97\x97""   \x8d"

#Characters which collate as 2-1 mappings, with the same primary
#collation order, but different secondary order.  In each set of
#three characters the first two together form a collation element
#which collates before the third character in LOCALE1, and vice
#versa in LOCALE2.
#This parameter must contain a comma separated list of six
#triplets of wide characters which correspond to multibyte
#characters with the following numbers of bytes: 1, 1, 2;
#2, 3, 1; 2, 1, 3; 4, 2, 3; 1, 3, 3; 3, 4, 4.
SECONDARY2_1=0,0,0;0,0,0;0,0,0;0,0,0; "\x8e", "\x97""  \x8f", "\x97""  \x90"; \
"\x97""  \x8e", "\x97\x97""   \x8f", "\x97\x97""   \x90"
```

```
#Characters which collate equally as 1-2 mappings.  In each set of
#three characters the first two are collation elements which
#collate as a pair equally with the third character.
#This parameter must contain a comma separated list of six
#triplets of wide characters which correspond to multibyte
#characters with the following numbers of bytes: 1, 1, 2;
#2, 3, 1; 2, 1, 3; 4, 2, 3; 1, 3, 3; 3, 4, 4.
EQUAL1_2=0,0,0;0,0,0;0,0,0;0,0,0; "\x91", "\x97""  \x92", "\x97""  \x93"; \
"\x97""  \x91", "\x97\x97""   \x92", "\x97\x97""   \x93"

#  The following parameters are not used in any tests if wcsftime() is not
#  supported.  In this case dummy values should be entered to satisfy the
#  config file compiler.  (They can be copied from wc_nosup.cfg).

#Time values returned by wcsftime() for LOCALE1.
#MONTH1_1 contains a comma separated list of the characters in
#the full name for January, MONTH2_1 for February, and so on.
#Likewise ABMON1_1, etc. contain the abbreviated month names,
#DAY1_1, etc. contain the seven full day names, starting with
#Sunday, and ABDAY1_1, etc. contain the abbreviated day names.
#AM_STR1 contains the AM string and PM_STR1 contains the PM string.
#These parameters must together contain 2, 3 and 4 byte multibyte
#characters (at least one of each size that is supported).
MONTH1_1="J", "a", "\x97\x97""   n", "\x97""  u", "a", "r", "y"
MONTH2_1="F", "e", "b", "r", "\x97""  u", "a", "r", "y"
MONTH3_1="M", "a", "r", "c", "h"
MONTH4_1="A", "p", "r", "i", "l"
MONTH5_1="M", "a", "y"
MONTH6_1="J", "\x97""  u", "\x97\x97""   n", "e"
MONTH7_1="J", "\x97""  u", "l", "y"
MONTH8_1="A", "\x97""  u", "g", "\x97""  u", "s", "t"
MONTH9_1="S", "e", "p", "t", "e", "m", "b", "e", "r"
MONTH10_1="O", "c", "t", "o", "b", "e", "r"
MONTH11_1="N", "o", "v", "e", "m", "b", "e", "r"
MONTH12_1="D", "e", "c", "e", "m", "b", "e", "r"
ABMON1_1="J", "a", "\x97\x97""   n"
ABMON2_1="F", "e", "b"
ABMON3_1="M", "a", "r"
ABMON4_1="A", "p", "r"
ABMON5_1="M", "a", "y"
ABMON6_1="J", "\x97""  u", "\x97\x97""   n"
ABMON7_1="J", "\x97""  u", "l"
ABMON8_1="A", "\x97""  u", "g"
ABMON9_1="S", "e", "p"
ABMON10_1="O", "c", "t"
ABMON11_1="N", "o", "v"
ABMON12_1="D", "e", "c"
DAY1_1="S", "\x97""  u", "\x97\x97""   n", "d", "a", "y"
DAY2_1="M", "o", "\x97\x97""   n", "d", "a", "y"
DAY3_1="T", "\x97""  u", "e", "s", "d", "a", "y"
DAY4_1="W", "e", "d", "\x97\x97""   n", "e", "s", "d", "a", "y"
DAY5_1="T", "h", "\x97""  u", "r", "s", "d", "a", "y"
DAY6_1="F", "r", "i", "d", "a", "y"
DAY7_1="S", "a", "t", "\x97""  u", "r", "d", "a", "y"
ABDAY1_1="S", "\x97""  u", "\x97\x97""   n"
ABDAY2_1="M", "o", "\x97\x97""   n"
ABDAY3_1="T", "\x97""  u", "e"
ABDAY4_1="W", "e", "d"
```

```
ABDAY5_1="T", "h", "\x97""  u"
ABDAY6_1="F", "r", "i"
ABDAY7_1="S", "a", "t"
AM_STR1="A", "M"
PM_STR1="P", "M"

#Time values returned by wcsftime() for LOCALE2, specified in the
#same manner as for LOCALE1.  The values in these parameters must
#differ from the values in the corresponding LOCALE1 parameters.
MONTH1_2="j", "a", "\x97\x97""   n", "v", "i", "e", "r"
MONTH2_2="f", "E", "v", "r", "i", "e", "r"
MONTH3_2="m", "a", "r", "s"
MONTH4_2="a", "v", "r", "i", "l"
MONTH5_2="m", "a", "i"
MONTH6_2="j", "\x97""  u", "i", "\x97\x97""   n"
MONTH7_2="j", "\x97""  u", "i", "l", "l", "e", "t"
MONTH8_2="a", "o", "C", "t"
MONTH9_2="s", "e", "p", "t", "e", "m", "b", "r", "e"
MONTH10_2="o", "c", "t", "o", "b", "r", "e"
MONTH11_2="\x97\x97""   n", "o", "v", "e", "m", "b", "r", "e"
MONTH12_2="d", "E", "c", "e", "m", "b", "r", "e"
ABMON1_2="j", "a", "\x97\x97""   n", "v"
ABMON2_2="f", "E", "v", "r"
ABMON3_2="m", "a", "r", "s"
ABMON4_2="a", "v", "r"
ABMON5_2="m", "a", "i"
ABMON6_2="j", "\x97""  u", "i", "\x97\x97""   n"
ABMON7_2="j", "\x97""  u", "i", "l"
ABMON8_2="a", "o", "C", "t"
ABMON9_2="s", "e", "p", "t"
ABMON10_2="o", "c", "t"
ABMON11_2="\x97\x97""   n", "o", "v"
ABMON12_2="d", "E", "c"
DAY1_2="d", "i", "m", "a", "\x97\x97""   n", "c", "h", "e"
DAY2_2="l", "\x97""  u", "\x97\x97""   n", "d", "i"
DAY3_2="m", "a", "r", "d", "i"
DAY4_2="m", "e", "r", "c", "r", "e", "d", "i"
DAY5_2="j", "e", "\x97""  u", "d", "i"
DAY6_2="v", "e", "\x97\x97""   n", "d", "r", "e", "d", "i"
DAY7_2="s", "a", "m", "e", "d", "i"
ABDAY1_2="d", "i", "m"
ABDAY2_2="l", "\x97""  u", "\x97\x97""   n"
ABDAY3_2="m", "a", "r"
ABDAY4_2="m", "e", "r"
ABDAY5_2="j", "e", "\x97""  u"
ABDAY6_2="v", "e", "\x97\x97""   n"
ABDAY7_2="s", "a", "m"
AM_STR2="a", "m"
PM_STR2="p", "m"
```

6.11 CONFIGURING VSTH

6.11.1 Introduction

This section provides supplemental information for users of VSTH.

6.11.2 Installation Directory

VSTH will require approximately 130 MB for TESTROOT on a typical RISC system.

6.11.3 Parameters

6.11.4 VSX Configuration Script

VSTH supports POSIX96 and UNIX98 execution modes only.

Action Points

1. VSTH uses the VSTH_PTHREAD_LIB and the VSTH_SYSLIBS parameters to identify libraries against which to link VSTH tests.

   `VSTH_PTHREAD_LIB` specifies the system library containing POSIX threads interfaces and must be set to -lpthread when performing conformance testing with VSTH.

   `VSTH_SYSLIBS` specifies any other system libraries needed to link VSTH and is often set to `-lrt`-laio when configuring VSTH.

6.11.5 Checking the Parameter Files

6.11.6 Creating Parameter Files

6.11.7 Top Level Makefile

VSTH does not require inheritance of group ids. It does require exectution of the filldisc target in the top-level makefile to configure the filesystem used for ENOSPC testing.

Action Points

1. Implement the ENOSPC filesystem.

6.11.8 User-Supplied Interface Routines

The following VSTH tests require `PRV_SETTHRSCHED` and `PRV_GETTHRSCHED` privileges. Paths are relative to TESTROOT/tset/PTHR.os .

```
thread/pthread_create/T.pthread_create
mutex/pthread_mutex_init/T.pthread_mutex_init
mutex_rt/pthread_mutex_getprioceiling/T.pthread_mutex_getprioceiling
mutex_rt/pthread_mutex_setprioceiling/T.pthread_mutex_setprioceiling
mutexattr_rt/pthread_mutexattr_getprioceiling/T.pthread_mutexattr_getprioceiling
mutexattr_rt/pthread_mutexattr_setprioceiling/T.pthread_mutexattr_setprioceiling
mutexattr_rt/pthread_mutexattr_getprotocol/T.pthread_mutexattr_getprotocol
mutexattr_rt/pthread_mutexattr_setprotocol/T.pthread_mutexattr_setprotocol
concurrency/pthread_setconcurrency/T.pthread_setconcurrency
concurrency/pthread_getconcurrency/T.pthread_getconcurrency
sched_rt/pthread_setschedparam/T.pthread_setschedparam
sched_rt/pthread_getschedparam/T.pthread_getschedparam
rwlock/pthread_rwlock_init/T.pthread_rwlock_init
sched/pthread_attr_setschedparam/T.pthread_attr_setschedparam
```

```
sched_rt/pthread_attr_setschedpolicy/T.pthread_attr_setschedpolicy
cond/pthread_cond_init/T.pthread_cond_init
condattr/pthread_condattr_setpshared/T.pthread_condattr_setpshared
mutexattr/pthread_mutexattr_init/T.pthread_mutexattr_init
mutexattr/pthread_mutexattr_setpshared/T.pthread_mutexattr_setpshared
```

The following VSTH test requires the PRV_MOUNT privilege. Paths are relative to TESTROOT/tset/PTHR.os .

```
ioprim/pwrite/pwrite_13
```

VSTH uses the vsth_setconcurrency() user-supplied routine found in SRC/userintf.c. The purpose of vsth_setconcurrency() is to ensure that significant thread contention occurs during VSTH execution.

If you are running VSTH in UNIX98 mode and have the pthread_setconcurrency() function available, simply remove the comments and supply a value for level which provides the maximum concurrency without requiring appropriate privilege.

If you are running VSTH in POSIX96 mode and don't have the pthread_setconcurrency() function, consult your system documentation to identify an appropriate implementation of vsth_setconcurrency().

Action Points

1. Implement setprv() for the arguments PRV_SETTHRSCHED, PRV_GETTHRSCHED and PRV_MOUNT.
2. Implement vsth_setconcurrency().

6.11.9 Installing The Pseudo-Languages

VSTH does not require installation of the pseudo-languages.

6.11.10 Wide Character Configuration File

VSTH does not require configuration of the wide character locales.

7. INSTALLING VSX

7.1 INTRODUCTION

When you have configured VSX for your system, you can proceed to the VSX installation stage. This is where the installation commands configured in the top level `Makefile` are executed and the VSX utilities and libraries are built.

The major part of the installation procedure is performed by a script which is executed with the user ID of user `vsx0` from the top level `Makefile`. The installation script applies the parameters and definitions in the files `vsxparams` and `vsxconfig.h` to the VSX source files. The script generates a report with details of the success of each step in a journal file in the `results` directory. The journal files from successive runs of the installation script are numbered sequentially.

The installation script includes the following steps:

7.1.1 VSX Header Files

The file `SRC/INC/std.h` is updated with values from `vsxparams` which are needed in C compilations.

7.1.2 Include Files

The set of your system include files is copied into `SRC/SYSINC` and each file is updated with any extra definitions required, from the file `vsxconfig.h`. The VSX include files are used to install and build the VSX software, but not in the execution of header file tests.

7.1.3 Directory Routines

The installation script checks the directory routines `opendir()` and `readdir()` are working. When the directory routines are not functioning correctly, the script gives a warning, both on the screen and in the install journal.

7.1.4 Variable Argument Routines

The installation script checks whether variable argument functions work correctly, using either `<varargs.h>` or `<stdarg.h>` depending on the test mode selected and whether the compiler defines the symbol `__STDC__`. Note that the contents of these headers is not checked during the configuration stage, when the other system headers are checked. If the header file contents are found to be incorrect, and you do not wish to alter the file in the system include directory, you can copy it in to `SRC/SYSINC` and correct the problem there.

7.1.5 Testroot Initialisation

The install script creates the testroot directory structure under the testroot directory.

7.1.6 Configuration Files

The building and cleaning configuration files, `tetbuild.cfg` and `tetclean.cfg` are created in the home directory, using the information from the configuration stage. The execution configuration file, `tetexec.cfg` is created in the testroot directory with some of the parameters set up with the information from the configuration stage. However, you must add the values for most of the parameters in the execution file manually. See the chapter entitled ‘‘EXECUTING VSX’’ for details.

These files contain the parameters which are used during the building, execution and cleanup stages respectively.

If these configuration files already exist, they are first renamed to `oldbuild.cfg`, `oldclean.cfg` and `oldexec.cfg` before being created. Parameter values from

the old `tetexec.cfg` are copied to the new file before it is updated with information from the configuration stage.

7.1.7 Scenario Files

The install script creates the scenario files `scen.bld` and `scen.exec` in the home directory. These are used by TETware to determine which tests are built and executed, respectively.

7.1.8 Update Common Software Files

Firstly, `userintf.c` is copied into `SRC/common/vport`. Then the `Makefile`s in the various sub-directories of `SRC/common` are updated with information from the configuration stage.

7.1.9 Subset-specific Install Scripts

Any additional procedures needed for the subsets and test mode you have selected are performed at this point. For example, additional subset-specific values may be added to `SRC/INC/std.h`.

7.1.10 Build Common Software

The installation script builds the VSX libraries and utility programs. The install journal gives a success/failure indication for each directory in which `make` is executed. You must investigate and correct any failures which occur during building before continuing any further.

Action Points

1. Obtain the necessary privileges for execution of the installation commands you have configured in the top level `Makefile` and execute `make` in the `vsx0` home directory. E.g.

   ```
   su root -c make
   ```

2. When `make` has completed, check the installation log in the `results` directory to ensure that no errors have occurred. If `make` encountered any errors, or there are errors in the log, you must correct them and re-run `make`.

7.2 INSTALLING VSTH

7.2.1 Introduction

This section provides supplemental information for users of VSTH.

7.2.2 VSX Header Files

7.2.3 Include Files

7.2.4 Directory Routines

7.2.5 Variable Argument Routines

7.2.6 Testroot Initialisation

7.2.7 Configuration Files

7.2.8 Scenario Files

7.2.9 Update Common Software Files

7.2.10 Subset-specific Install Scripts

There are no additional VSTH specific installation scripts.

7.2.11 Build Common Software

VSTH builds three libraries during the installation phase. They are `libvsth.a`, `libvsth_cp.a` and `libvsth_rt.a`. If you observe errors when attempting to build libvsth.a, you must correct them before you can build VSTH, as virtually all the tests in `tset/PTHR.os` link with libvsth.a.

If you observe errors when attempting to build libvsth_cp.a but libvsth.a builds successfully, you can still link all VSTH tests with the exception of `tset/PTHR.os/cancel/pthread_cancel/T.pthread_cancel`.

If you observe errors when attempting to build libvsth_rt.a but libvsth.a builds successfully, you can still link all VSTH tests with the exception of `tset/PTHR.os/general/General/T.General`.

VSTH test package libraries have been organized in this way to facilitate use by incomplete implementations.

Action Points

1. Ensure that libvsth.a builds cleanly prior to any use of VSTH. Failure to do so will cause the subsequent build failure of all tests in the `tset/PTHR.os` area.
2. Ensure that libvsth_cp.a builds cleanly prior to use of VSTH as a conformance test. Failure to do so will cause the subsequent build failure of the `tset/PTHR.os/cancel/pthread_cancel/T.pthread_cancel` testset.
3. Ensure that libvsth_rt.a builds cleanly prior to use of VSTH as a conformance test. Failure to do so will cause the subsequent build failure of the `tset/PTHR.os/general/General/T.General` testset.

8. BUILDING VSX

8.1 INTRODUCTION

When you have configured and installed VSX, the next stage builds a series of executable programs in your `TESTROOT` directory. These programs make up the VSX test suite, which you run in the execution stage to verify your system. The building stage also copies the source files for any header file and C language tests into the correct directories.

You can choose to build all the testsets you have configured. Alternately, you can choose the section or area you want to build and, optionally, build single testsets. When building the tests, you can specify options to use an alternative configuration file and to modify parameters on the command line, among others. In addition, you can use the output from the building stage in a VSX journal file for the VSX reporting stage.

You may also need to install a loopback lead for the terminal interface testing. See the section entitled ‘‘TERMINAL INTERFACE TESTING’’ later in this chapter for information.

8.2 BUILDING ALL REQUIRED TESTSETS

8.2.1 Introduction

Invoking the Test Case Controller with the command

```
tcc -b -s scen.bld
```

will cause all the testsets for the options selected during the configuration stage to be built. If you have not set the environment variable **TET_EXECUTE** to the pathname of your testroot directory, you must specify it on the `tcc` command line. For example, to specify the default testroot location, append

```
-a TESTROOT
```

to the command given above. The remaining example commands in this chapter assume that **TET_EXECUTE** is set in the environment.

The build parameters are found in the `tetbuild.cfg` file in the vsx0 home directory. This file is created during the installation phase.

Journal File
The results from the building stage are placed in a journal file under the `results` directory. The name of this file is output by the `tcc` on startup. VSX provides utilities to produce reports from these files — see the chapter entitled ‘‘REPORTING’’ for further details.

8.3 BUILDING SELECTED TESTSETS (OPTIONAL)

8.3.1 Sections and Areas

The package-specific sections at the end of this chapter give details about the parts of the test suite you can build. To build individual testsets and use other options for the `tcc` command, see the sections marked ‘‘OPTIONAL’’ later in the chapter.

8.3.2 Building Selected Parts of a Scenario

To build selected parts of the test suite, you can use the `-y` and `-n` options of the `tcc` to select which lines of the file `scen.bld` you wish to include (`-y`) or exclude (`-n`). You can use as many of these options as you like in one command. If a scenario line matches both a `-y` string and a `-n` string it will be excluded.

For example, to build just the `POSIX.os` section, use the command:

```
tcc -b -s scen.bld -y POSIX.os
```

or to build everything except the header tests, use the command:

```
tcc -b -s scen.bld -n .hdr
```

or to build the `streamio` areas in all the sections that have one, but excluding tests of `fprintf`, `printf`, `sprintf`, `vfprintf`, `vprintf` and `vsprintf`, use the command:

```
tcc -b -s scen.bld -y streamio -n printf
```

The journal file for a partial execution is handled correctly by the report writer.

8.3.3 Building Individual Testsets

If the list of testsets you wish to build is too varied to be specified easily using `-y` and `-n` options, simply edit a copy of the scenario file `scen.bld` to reflect the testsets you wish to build.

Examples
If the file `myscen.bld` contains an edited copy of `scen.bld` then use the command

```
tcc -b -s myscen.bld
```

to build just the testsets contained in the file.

8.3.4 Additional Options

The `tcc` manual page gives full details of the additional options you can use with the `tcc` command. The following options are some of the most useful.

1. To see a running progress report, include the `-p` option. The `tcc` will then output a line to the terminal as it starts building each testset.
2. To use an alternative configuration file to `tetbuild.cfg`, include the `-g` *filename* option. The file must be in the same format as `tetbuild.cfg`.
3. To override a parameter in the build parameters file, include the `-v` option. For example, to use the operator name ‘‘A N Other’’ rather than the value defined in `tetbuild.cfg`, use the command

   `tcc -b -v VSX_OPER="A N Other"` *rest–of–command*
4. In order to build testsets which failed to build in a previous build, use

   `tcc -b -s scen.bld -r FAIL` *old–journal–file*

   where *old–journal–file* is the journal file from which the codes are extracted.

Action Points

1. Log in as the user `vsx0`.
2. Give the command

   ```
   tcc -b -s scen.bld
   ```

   with the other options you want to use. Use the command

   ```
   ../bin/tcc -b -s scen.bld
   ```

   from the `vsx0` home directory if `$HOME/../bin` is not in your **PATH**.

8.4 REMOVING BUILT TESTSETS

When you want to remove all of the object files and executable programs for the whole test suite or part of it, you can use the `-c` option of the `tcc` command. This option works in exactly the same way as the `-b` option except that, instead of building the test suite, this option returns the system to the state it was in before you started the building stage.

Use this option when you want to re-build VSX using a different compiler.

8.5 REPORTING

You can produce reports from the journal files output by `tcc` by using the procedures described in the chapter entitled ‘‘REPORTING’’. These reports show you where any compilation failures have occurred.

8.6 TERMINAL INTERFACE TESTING

8.6.1 Introduction

For tests that need to use a terminal interface, two ports on the same machine are connected with each other. Then one port simulates the presence of a terminal providing for error-free and repeatable testing. Where implementations do not have at least two terminal ports, the tests can instead be performed using pseudo-terminals. In this case a user-supplied routine is called to open the pseudo-terminals. See the section entitled ‘‘USER-SUPPLIED INTERFACE ROUTINES’’ for a description of this routine.

The terminal loop-back is only needed by some test packages. Refer to the package-specific sections at the end of this chapter for details.

If the system does not provide any devices which support the general terminal interface, then tests which need to use a terminal interface will give `unsupported` results.

Certain tests require modem control and will be reported `unsupported` if this is not available.

8.6.2 Cable Wiring

If the system has at least two ports, preparation for the tests begins by connecting two terminal or modem ports back-to-back (sometimes called *loop-back* or *closed-loop*) using a cable which fits these sockets. For example, a null modem cable is suitable for RS232-type DTE ports. A typical cable wiring would be:

Null Modem Cable (DTE-DTE)				
Function	**Pins**	**Data Flow**	**Pins**	**Function**
Protective Ground	1	⇐⇒	1	Protective Ground
Transmit Data	2	⇒	3	Receive Data
Receive Data	3	⇐	2	Transmit Data
Request to Send	4	⇒	8	Data Carrier Detect
Clear to Send	5	⇐	8	Data Carrier Detect
Data Set Ready	6	⇐	20	Data Terminal Ready
Signal Ground	7	⇐⇒	7	Signal Ground
Data Carrier Detect	8	⇐	4	Request to Send
Data Carrier Detect	8	⇒	5	Clear to Send
Data Terminal Ready	20	⇒	6	Data Set Ready

An equivalent ‘‘null terminal’’ cable suitable for RS232-type DCE ports could be wired as follows:

Null Terminal Cable (DCE-DCE)				
Function	**Pins**	**Data Flow**	**Pins**	**Function**
Protective Ground	1	⇐⇒	1	Protective Ground
Transmit Data	2	⇐	3	Receive Data
Receive Data	3	⇒	2	Transmit Data
Request to Send	4	⇐	6	Data Set Ready
Data Set Ready	6	⇒	4	Request to Send
Data Set Ready	6	⇒	20	Data Terminal Ready
Signal Ground	7	⇐⇒	7	Signal Ground
Data Terminal Ready	20	⇐	6	Data Set Ready

Action Points

1. Check the package-specific action points for this chapter to see whether the terminal loop-back is needed for the subsets and test mode you have selected. If it is not needed, skip the following action points.
2. If the system has at least two terminal ports, wire and connect two ports to provide a loop-back.
3. Make the two ports readable and writable by user `vsx0`.
4. There must not be any processes attached to these ports. (For example, you may have to change `/etc/inittab` or `/etc/ttys` to remove login processes at this point.)

8.7 TROUBLESHOOTING

You may encounter some of the following problems when you build tests with `tcc`. This section lists common problems and gives notes explaining how to overcome them.

1. `tcc` cannot create lock files.

   There may be lock files left over from a previous run. Whenever possible `tcc` always removes its lock files, however if it is terminated by an uncatchable signal or the system crashes then lock files may be left behind. Check for files named `tet_lock` in both the source and testroot directory hierarchies at the testset level. The `tet_lock` file may itself be the lock file, or it may be a directory containing lock files.

2. Some header tests fail to build, due to a program called **hdrdefs** trying to write files larger than the file size limit, or running out of disk space.

   Normally **hdrdefs** creates a copy of each header it processes, with all nested **#include** lines expanded out. On some systems, if certain key headers are included by many others, the nesting complexity can cause these headers to be copied so many times into the expanded file that the file becomes huge.

   The **hdrdefs** program can be prevented from creating the expanded-out files by setting the environment variable HDRDEFS_NOEXPAND. However, this also causes **hdrdefs** to treat 'missing' nested include files differently. Instead of indicating that a nested header could not be opened by placing a special **#define** in the expanded-out file, it will report an error immediately and exit. If you try to build the header tests with HDRDEFS_NOEXPAND set, and the build fails due to some missing nested includes, you should first check that the reason they are not being found is because they do not exist on your system, rather than because their location has been omitted from the INCDIRS parameter in `SRC/vsxparams`. You can stop **hdrdefs** reporting missing nested include files by creating dummy versions of the missing files. Each file must contain a line which will cause an error to be reported should the file be compiled during one of the header tests. For example:

   ```
   #error Dummy version of sys/thisfile.h was used
   ```

   The location of the dummy include files should be added to the INCDIRS parameter in `SRC/vsxparams` so that **hdrdefs** will be able to find them.

3. Some header tests fail to build, with syntax errors reported by a program called **hdranal**.

   If your system headers make use of type names that are built-in to the compiler, they will not be recognised by **hdranal** unless they are defined in the file `SRC/INC/builtins.h`. This file may be edited by the user. As distributed, the file contains all the definitions needed for systems that VSX has been run on to date. If you find you need to add new definitions to the file, please inform the VSXgen support team (see the appendix entitled ''SUPPORT SERVICES'') so that the additions can be included in the next VSXgen release.

8.8 BUILDING VSTH

8.8.1 Introduction

This section provides supplemental information for users of VSTH.

8.8.2 Building All Required Testsets

The commands described in the generic part of this guide can be followed to build all the required testsets.

8.8.3 Building Selected Testsets (Optional)

To build selected parts of the test suite, you can use the `-y` and `-n` options of the `tcc` command. For example, to build just the tests related to reader-writer locks use the command:

```
tcc -p -b -s scen.bld -y /rwlock
```

or to build everything except the Realtime Threads tests, use the command:

```
tcc -p -b -s scen.bld -n _rt
```

8.8.4 Removing Built Testsets

8.8.5 Reporting

VSTH uses the default report writer provided with VSXgen.

8.8.6 Terminal Interface Testing

VSTH does not currently require configuration of the terminal loop-back.

8.8.7 Troubleshooting

Action Points

1. None.

9. EXECUTING VSX

9.1 INTRODUCTION

Before you run the testsets you have built, you must set up the file containing execution parameters. When you are ready to execute the VSX testsets, you can choose to run the entire selection of testsets you have configured or, optionally, execute sections, areas, single testsets and individual invocable components. In addition, you can use options to change parameters on the command line and to re-execute failed testsets from old journal files, among others. You can use the output from the execution stage in a journal file for the VSX reporting stage.

9.2 THE EXECUTION PARAMETERS FILE

9.2.1 Introduction

During the testset execution stage, the VSX testsets find information about your system from the execution parameters file. This file contains lines which define the parameters and give their values. If the testset cannot find the information, it either uses a default value or reports that the parameter is not set in the test results for the tests that use that parameter.

9.2.2 Setting the Execution Parameters

VSX looks for the execution parameters file, `tetexec.cfg`, in the testroot directory when you execute the Test Case Controller. During the installation stage, VSX generates an execution parameters file in the testroot directory, but some of the values in the file are INCORRECT for your system.

Format
Each line in the execution parameters file is either a comment line, beginning with the hash character (#), or a parameter line. Parameter lines use the following format:

parameter–name=*parameter–value*

The contents of the generated file vary according to the subsets you have selected. The section below lists the parameter names that are always present. Additional parameters required only for one subset are listed in the package-specific sections at the end of this chapter.

The value given to each parameter may be any sequence of characters which is valid for the associated parameter. When you leave the value blank after the equals sign (=), the parameter is set to its default value, if it has one.

Action Points

1. Check the execution parameters file `tetexec.cfg` before you start the VSX execution stage and edit any values which are not correct for your system.

9.3 EXECUTION PARAMETER NAMES

Check or set the values for the following parameters, and any additional subset-specific parameters, in the execution parameters file before you start the VSX execution stage. Note that the order of these parameters is the same as they are in the file where you will edit them.

9.3.1 General Parameters

TEST_MODE

The testing mode selected when `config.sh` was run. The value is set automatically by the installation procedure and should not be altered.

TEST_PACKAGES

A list of the test packages being used. The value is set automatically by the installation procedure and should not be altered.

VSXDIR

The source directory for the VSX source software.

Where Used
`vbuild` (`vprog`)

Default Value
None

VSX_DBUG_FLAGS

The debugging flags used to determine the level of debugging information generated upon execution of tests.

Where Used
All testsets

Default Value
If this parameter is not set, no debugging output is produced.

VSX_DBUG_FILE

The default destination for debug and path tracing output. This file is used if none is specified in the debug flags in **VSX_DBUG_FLAGS**. Output is appended to the file on each run, so an existing file should be saved or deleted before running `tcc` with debugging enabled. Use of relative path names is not recommended, as the directory in which test programs are executed varies. This parameter is usually set to *your–testroot–directory*`/dbug.out`.

Where Used
All testsets

Default Value
If this parameter is not set, debug output is sent to the standard error stream. Note that this often causes incorrect test results in cases where the interface being tested uses `stderr`. For this reason, it is advisable to direct debugging output to a file.

VSX_NAME

The test run name; that is, what this particular test run will be called in the final `vrpt` and/or `prpt` output.

Where Used
`vrpt` (`vprog`)
`prpt` (`vprog`)

Default Value
No default value is assigned.

VSX_OPER

The name of the operator for this test run.

Where Used

`vrpt` (`vprog`)
`prpt` (`vprog`)

Default Value

No default value is assigned.

VSX_ORG

Name of the agency running the tests/for whom the tests are being run.

Where Used

`vrpt` (`vprog`)
`prpt` (`vprog`)

Default Value

No default value is assigned.

VSX_PATH

Used to set the **PATH** environment variable during execution of testsets in the C language and header file sections. This will be used to locate the C compiler and the executable object produced by it. This parameter should always include the current directory.

Where Used

`driver.hdr` (`drivers`)
`driver.C` (`drivers`)

Default Value

`:/bin:/usr/bin`; that is: your current directory, then `/bin`, then `/usr/bin`.

VSX_SYS

The test system name; that is, the name of the system being tested.

Where Used

`vrpt` (`vprog`)
`prpt` (`vprog`)

Default Value

No default value is assigned.

VSX_UID0, VSX_UID1, VSX_UID2

These are the user IDs associated with the users `vsx0`, `vsx1` and `vsx2` respectively.
These must not be privileged users.

Where Used

`uids` (`genlib`)

Default Value

If not defined, each of the tests that uses these parameters is reported as `unresolved` or `uninitiated`.

VSX_GID0, VSX_GID1, VSX_GID2

These are the group IDs associated with the groups `vsxg0`, `vsxg1` and `vsxg2` respectively.
These must not be privileged groups.

Where Used
`uids` (`genlib`)

Default Value
If not defined, each of the tests that uses these parameters is reported as `unresolved` or `uninitiated`.

TET_SIG_IGN

A list of the signal numbers that are to be ignored during testing. This should be a comma-separated list of (non-POSIX) signal numbers. Many systems will need to include the signal number for SIGSYS.

Where Used
TETware API

Default Value
No default value is assigned.

TET_SIG_LEAVE

A list of the signal numbers that are to be left alone during testing. These are most often signals which cause problems both if they are set to be caught and if they are ignored via **TET_SIG_IGN**. This should be a comma-separated list of (non-POSIX) signal numbers.

Where Used
`setsigs` (`vlib`)
TETware API

Default Value
No default value is assigned.

9.3.2 Compiler Characteristics

These parameters are only required if one or more selected subsets contain C language tests, or header tests which use the generic driver `driver.hdr`. Some test packages may build alternative header test drivers, which use different parameters.

VSX_CC

The full path name of the C compiler to be used in header and C language tests. This is normally set to the same value as **CC** in `SRC/vsxparams`. The file named by **VSX_CC** may be a shell script or an executable file.

Where Used
`driver.hdr` (`drivers`)
`driver.C` (`drivers`)

Default Value
`/bin/cc`

VSX_CFLAGS

The flags to be passed to the C compiler (**VSX_CC**). This is normally set to the same value as **COPTS** in `SRC/vsxparams`.

These flags must not define any of the feature test macros `_XOPEN_SOURCE`, `_XOPEN_SOURCE_EXTENDED`, `_POSIX_SOURCE`

or `_POSIX_C_SOURCE`.

Where Used
`driver.hdr` (`drivers`)
`driver.C` (`drivers`)

Default Value
NULL; that is, no string.

VSX_LIBS
Libraries and linker flags to be passed to the C compiler (**VSX_CC**).

These will usually include any subset-specific libraries named individually in parameters in `SRC/vsxparams`, any libraries specified in **SYSLIBS** in `SRC/vsxparams`, and any link editor command line options specified in **LDFLAGS** in `SRC/vsxparams`.

Where Used
`driver.hdr` (`drivers`)
`driver.C` (`drivers`)

Default Value
`-lm`

9.3.3 Operating System Characteristics Common To Multiple Subsets

The following parameters are in the generic part of `tetexec.cfg` because they may be needed by more than one subset. This avoids the need to configure the same information multiple times under different parameter names. However, if you have not selected any of the subsets which use a particular parameter, you do not need to set a value for that parameter. Refer to the package-specific sections at the end of this chapter to see which of these parameters are needed, and where they are used.

VSX_BLKDEV_FILE
The full path name of a block special device which exists on your system. If block special files are not supported, this parameter should be set to `unsup`.

Default Value
If not defined, each of the tests that uses **VSX_BLKDEV_FILE** is marked as `unresolved` or `uninitiated`.

VSX_CHRDEV_FILE
The path name of a character special device which exists on your system. If character special files are not supported, this parameter should be set to `unsup`.

Default Value
If not defined, each of the tests that uses **VSX_CHRDEV_FILE** is reported as `unresolved` or `uninitiated`.

VSX_FCNTL_EDEADLK
Does `fcntl()` detect `EDEADLK`? (`Y`–yes or `N`–no)

Default Value
None. In the case no value is specified, tests using this parameter will be reported `uninitiated` or `unresolved`.

VSX_FCNTL_MAXLOCK

The maximum number of locks that can be set using `fcntl()`. This number does not have to be exact, just sufficiently large to obtain an `ENOLCK` error. A value of −1 indicates there is no practical limit (e.g., if limited only by available memory).

Default Value

None. In the case no value is specified, tests using this parameter will be reported `uninitiated` or `unresolved`.

VSX_INVALID_FCNTL_CMD

An invalid `cmd` value for the argument to `fcntl()`.

Default Value

−1

VSX_INVALID_GID

Out of range group ID. If all *gid_t* values are valid group IDs, this parameter should be set to `unsup`, so tests using this parameter will be reported `unsupported`.

Default Value

None. In the case no value is specified, tests using this parameter will be reported `uninitiated` or `unresolved`.

VSX_INVALID_GNAME

Group name not in the group database.

Default Value

`nogroup`

VSX_INVALID_PNAME

User name not in the user database.

Default Value

`nouser`

VSX_INVALID_UID

Out of range user ID. If all `uid_t` values are valid user IDs, this parameter should be set to `unsup`, so tests using this parameter will be reported `unsupported`.

Default Value

None. In the case no value is specified, tests using this parameter will be reported `uninitiated` or `unresolved`.

VSX_INVALID_WHENCE

An invalid `whence` value for the argument to `lseek()` and `fseek()`, and for the `l_whence` structure member passed to `fcntl()`.

Default Value

−1

VSX_INVAL_SIG

An illegal signal number.

Default Value

If not defined, the default is set to −1.

VSX_MOUNT_DEV

The full path name of a block special file which can be mounted. The value may be the same as that given for the **VSX_ROFS** and/or **VSX_NOSPC_DEV** parameters.

Default Value

If not defined, each of the tests that uses **VSX_MOUNT_DEV** will be reported as `uninitiated` or `unresolved`.

VSX_NOSPC_DEV

The full path name of the block special file for the mountable device which was filled during the installation stage. The value may be the same as that given for the **VSX_MOUNT_DEV** and/or **VSX_ROFS** parameters.

Default Value

If not defined, each of the tests that uses **VSX_NOSPC_DEV** will be reported as `untested`.

VSX_PURE_FILE

The full path name to an executable pure procedure (shared text) file, this is normally set to *your–testroot–directory*`/BIN/purefile`. If shared executables are not supported, this parameter should be set to `unsup`. If the system supports shared executables but does not produce the `ETXTBSY` error condition, this parameter should **not** be set to `unsup`.

Each test takes its own copy of the file before performing the test using it. This is done so that if, for example, `creat()` on a busy pure procedure file succeeds and truncates the file, other tests that depend on the existence of `purefile` will not be affected.

Default Value

If not defined, each of the tests that uses this parameter will be reported as `unresolved` or `uninitiated`.

VSX_READDIR_EBADF

Does `readdir()` detect `EBADF`? (`Y`–yes or `N`–no)

Default Value

None. In the case no value is specified, tests using this parameter will be reported `uninitiated` or `unresolved`.

VSX_ROFS

The full path name of a block special file which can be mounted read-only. If read-only file systems are not supported, this parameter should be set to `unsup`. The value may be the same as that given for the **VSX_MOUNT_DEV** and/or **VSX_NOSPC_DEV** parameters.

Default Value

If not defined, each of the tests that uses **VSX_ROFS** will be reported `unresolved`.

VSX_SIGSET_EINVAL

Can `sigaddset()` and `sigdelset()` give `EINVAL`? (`Y`–yes or `N`–no)

Default Value

None. In the case no value is specified, tests using this parameter will be reported `uninitiated` or `unresolved`.

VSX_SYS_OPEN_MAX

The maximum number of files that can be open on the system at any time. This number does not have to be exact, just sufficiently large to obtain an `ENFILE` error. A value of –1 indicates that there is no practical limit (e.g., if limited only by available memory). In the latter case the tests using this parameter will be reported as `untested`.

Default Value

None. In the case no value is specified, tests using this parameter will be reported `uninitiated` or `unresolved`.

VSX_TTYNAME

Device name of the terminal on which the `tcc` is running.

Default Value

None.

VSX_TTYUSER

The login name of the user logged on at the terminal specified by **VSX_TTYNAME**.

Default Value

None.

VSX_ULIMIT_BLKS

The minimum multiple of blocks for which a file size limit can be set (using the `setlimit()` user-supplied function). Most implementations only support settings which correspond to the underlying file system block size. If the system does not support setting a file size limit this parameter should be set to –1.

Default Value

None.

VSX_UNLOCKABLE_FILE

The name of a file which cannot be locked. The file must have read and write permission for user `vsx0` or group `vsxg0`. If no unlockable files exist or can be created on the system, then this parameter should be set to `unsup`.

Default Value

None. If not defined, each of the tests that uses **VSX_UNLOCKABLE_FILE** will be reported as `unresolved` or `uninitiated`.

VSX_UNUSED_GID

An unused (but valid) group ID. If all unused group IDs are invalid, this parameter should be set to `unsup`.

Default Value
None.

VSX_UNUSED_UID
An unused (but valid) user ID. If all unused user IDs are invalid, this parameter should be set to `unsup`.

Default Value
None.

9.3.4 Terminal Interface Parameters Common To Multiple Subsets

The parameters in this part of `tetexec.cfg` are only needed by some test packages. Refer to the package-specific sections at the end of this chapter to see whether they are needed, and if so where they are used.

If the system does not provide any devices which support the general terminal interface, the **VSX_TERMIOS_TTY** and **VSX_TERMIOS_LOOP** parameters should be set to `unsup`. In this case the remaining parameters are not used.

If the system does provide devices which support the general terminal interface, but does not have two ports which can be used to form a hardware loopback, then where possible testing should be performed using pseudo-terminals and the software loopback facility in VSX. If the pseudo-terminal devices do not support the general terminal interface fully, they should still be used, and waivers requested for the tests which use unsupported features (typically parity generation/detection and character size). If testing with neither a hardware nor software loopback is possible, set the **VSX_TERMIOS_TTY** and **VSX_TERMIOS_LOOP** parameters to `unsup`. In this case it will be necessary to request a waiver for the terminal interface tests which give UNSUPPORTED results as they will disagree with the conformance statement.

VSX_TERMIOS_TTY
This is the terminal device to be used as controlling terminal for the tests. This parameter should be set to `unsup` if the general terminal interface is not supported. When using pseudo-terminals for terminal interface testing, if pseudo-terminal master devices are obtained from a clone device such as `/dev/ptmx` then **VSX_TERMIOS_TTY** must be set to a dummy slave device name long enough to be overwritten with the real name when it is obtained; for example `/dev/pts/XXX`.

Default Value
None.

VSX_TERMIOS_LOOP
This is the terminal device connected to **VSX_TERMIOS_TTY** by loopback. This parameter should be set to `unsup` if the general terminal interface is not supported. When using pseudo-terminals for terminal interface testing, if pseudo-terminal master devices are obtained from a clone device such as `/dev/ptmx` then **VSX_TERMIOS_LOOP** must be set to a dummy slave device name long enough to be overwritten with the real name when it is obtained.

Default Value
None.

VSX_MASTER_TTY

When using pseudo-terminals for terminal interface testing, this identifies the master side of the pseudo-terminal pair for which **VSX_TERMIOS_TTY** is the slave. This parameter should be left unset if **VSX_TERMIOS_TTY** is not a pseudo-terminal.

Default Value

If not defined, the tests which perform terminal testing do not provide a software loopback between **VSX_TERMIOS_TTY** and **VSX_TERMIOS_LOOP**.

VSX_MASTER_LOOP

When using pseudo-terminals for terminal interface testing, this identifies the master side of the pseudo-terminal pair for which **VSX_TERMIOS_LOOP** is the slave. This parameter should be left unset if **VSX_TERMIOS_LOOP** is not a pseudo-terminal.

Default Value

If not defined, the tests which perform terminal testing do not provide a software loopback between **VSX_TERMIOS_TTY** and **VSX_TERMIOS_LOOP**.

VSX_TERMIOS_ASYNC

Are **VSX_TERMIOS_TTY** and **VSX_TERMIOS_LOOP** asynchronous serial terminals? (`Y`–yes or `N`–no)

Default Value

If not defined, each of the tests in the above testsets that uses **VSX_TERMIOS_ASYNC** will be reported as `uninitiated` or `unresolved`.

VSX_TERMIOS_BUFFERED

Do **VSX_TERMIOS_TTY** and **VSX_TERMIOS_LOOP** have buffered output queues? (`Y`–yes or `N`–no)

Default Value

If not defined, each of the tests in the above testsets that uses this parameter will be reported as `uninitiated` or `unresolved`.

VSX_TERMIOS_SPEED

This is the normal speed setting for terminal tests (e.g., `B9600`). If split baud rates are supported, this parameter should be set to an **output** baud rate which may be used with a different input baud rate.

Default Value

None.

VSX_MODEM_CONTROL

Is modem control supported? (`Y`–yes or `N`–no)

Default Value

None. If not defined, each of the tests in the above testsets that uses **VSX_MODEM_CONTROL** will be reported as `uninitiated` or `unresolved`.

VSX_START_STOP_CHNG
Can the `START` and `STOP` characters be changed? (`Y`–yes or `N`–no)

Default Value
None. If not defined, each of the tests in the above testsets that uses **VSX_START_STOP_CHNG** will be reported as `uninitiated` or `unresolved`.

VSX_TCGETPGRP_SUPPORTED
Is `tcgetpgrp()` supported? (`Y`–yes or `N`–no)

Default Value
None. If not defined, each of the tests in the above testsets that uses **VSX_TCGETPGRP_SUPPORTED** will be reported as `uninitiated` or `unresolved`.

VSX_TCSETPGRP_SUPPORTED
Is `tcsetpgrp()` supported? (`Y`–yes or `N`–no)

Default Value
None. If not defined, each of the tests in the above testsets that uses **VSX_TCSETPGRP_SUPPORTED** will be reported as `uninitiated` or `unresolved`.

VSX_UNSUPPORTED_CFLAG
An unsupported `c_cflag` value or speed. The value can be specified using `c_cflag` symbols from `<termios.h>`, as a speed with leading `B`, or as a numeric value (with leading `0` for octal, `0x` for hexadecimal, otherwise decimal). If a `c_cflag` symbol or numeric value is prefixed with `~` the relevant bits will be cleared instead of set in `c_cflag`. This parameter should be set to `none` if all possible `c_cflag` values are supported.

Default Value
None. If not defined, each of the tests in the above testsets that uses this parameter will be reported as `uninitiated` or `unresolved`.

VSX_SUPPORTED_CFLAG
A `c_cflag` value which is not the default value in effect when the terminal is opened, or a speed other than that specified for **VSX_TERMIOS_SPEED**. The value must be unrelated to **VSX_UNSUPPORTED_CFLAG** (e.g., they cannot both be speeds or both be one of `CS5`, `CS6`, `CS7` and `CS8`). The value is specified in the same form as for **VSX_UNSUPPORTED_CFLAG**. This parameter should be set to `none` if no settings other than the defaults are supported.

Default Value
None. If not defined, each of the tests in the above testsets that uses this parameter will be reported as `uninitiated` or `unresolved`.

PCTS_ECHOE
Erase sequence echoed when `ECHOE` and `ECHO` are set.

Default Value
None.

PCTS_ECHOK

Kill sequence echoed when `ECHOK` and `ECHO` are set, and a `'\025'` kill character is used to kill an input line containing seven characters.

Default Value
None.

9.4 EXECUTING THE VSX TEST SUITE

9.4.1 Introduction

The TETware test case controller, `tcc`, controls the execution of the test suite. The driver executes all the testsets or those for the part you have requested. The results from the execution stage are placed in a journal file under the `results` directory. The name of this file is output by the `tcc` on startup.

The sections below give details about the parts of the test suite you can execute and how to execute them. To execute individual testsets and use other options for the test suite driver, see the sections marked ‘‘OPTIONAL’’ later in this chapter.

Note that `tcc` cannot be run using `nohup` as this would break the association with the login terminal specified in the **VSX_TTYNAME** parameter. If you wish to leave `tcc` to run unattended but do not want the terminal to be left logged in when it finishes you can use the shell’s `exec` command to execute `tcc` in place of the login shell. This will cause the terminal to be logged out when `tcc` exits.

9.4.2 Executing All Required Tests

To execute all the tests for the options selected during the configuration stage, invoke the test case controller with the command:

```
tcc -e -s scen.exec
```

If you have not set the environment variable **TET_EXECUTE** to the pathname of your testroot directory, you must specify it on the `tcc` command line. For example, to specify the default testroot location, append

```
-a TESTROOT
```

to the command given above. The remaining example commands in this chapter assume that **TET_EXECUTE** is set in the environment.

The following sections contain details on executing selected parts of the test suite. Note that if you make changes to your system to correct the faults diagnosed by VSX, it is not sufficient just to re-build and re-run the tests that failed and see that they now pass. The whole of VSX must be re-built and re-run to ensure that the changes have not had an adverse effect on any other tests.

Action Points

1. Log in as the user `vsx0`.
2. Give the command

   ```
   tcc -e -s scen.exec
   ```

   with the other options you want to use. Use the command

   ```
   ../bin/tcc -e -s scen.exec
   ```

   from the home directory if `$HOME/../bin` is not in your **PATH**.

9.4.3 Executing Selected Parts of a Scenario (OPTIONAL)

To execute selected parts of the test suite, you can use the `-y` and `-n` options of the `tcc` to select which lines of the file `scen.exec` you wish to include (`-y`) or exclude (`-n`). You can use as many of these options as you like in one command. If a scenario line matches both a `-y` string and a `-n` string it will be excluded.

For example, to execute just the `POSIX.os` section, use the command:

```
tcc -e -s scen.exec -y POSIX.os
```

or to execute everything except the header tests, use the command:

```
tcc -e -s scen.exec -n .hdr
```

or to execute the `streamio` areas in all the sections that have one, but excluding the macro versions of the interfaces, use the command:

```
tcc -e -s scen.exec -y streamio -n streamio/M
```

The journal file for a partial execution is handled correctly by the report writer.

9.4.4 Executing Individual Testsets (OPTIONAL)

If the list of testsets you wish to execute is too varied to be specified easily using `-y` and `-n` options, simply edit a copy of the scenario file `scen.exec` to reflect the testsets you wish to build.

Examples
If the file `myscen.exec` contains an edited copy of `scen.exec` then use the command

```
tcc -e -s myscen.exec
```

to execute just the testsets contained in the file.

9.4.5 Executing Individual Tests (OPTIONAL)

The lowest level of granularity in VSX allows you to execute individual tests. Most tests can be executed in this way, but some are dependent upon execution of earlier tests in the testset, in which case only groups of dependent tests may be executed as a single unit. Also, the header and C language tests can only be executed as whole testsets.

The mechanism for executing individual tests is the TET *invocable component* (IC) facility. Where no dependencies between tests exist the IC numbers are the same as the test numbers. Where dependencies exist, the IC number for a group of dependent tests is the same as the test number of the first test in the group. For example if a testset contains four tests and test 3 is dependent on test 2, the IC numbers will be as follows:

Test number	IC number
1	1
2	2
3	2
4	4

If IC number 2 is requested, then tests 2 and 3 will both be executed.

Note that even where no explicit dependency has been identified, some tests may behave differently when executed individually than when executed in the normal testset sequence. For this reason, it is always advisable to re-execute the whole testset once individual testing has been completed.

To execute selected invocable components from one or more testsets add a comma-separated list of the IC numbers in curly braces on the end of the associated scenario

lines. For example, the scenario file

```
all
        /tset/POSIX.os/ioprim/write/T.write{3,7,8}
```

will execute only IC numbers 3,7 and 8 in the `write()` testset.

Alternatively, a one-off execution of selected IC numbers from a single testset can be performed using the `-l` option of `tcc`. For example, the above execution could also be achieved by the command

```
tcc -e -l /tset/POSIX.os/ioprim/write/T.write{3,7,8}
```

Some shells may require the braces to be quoted.

Multiple `-l` options may be specified to execute more than one testset.

9.4.6 Additional Options (OPTIONAL)

The `tcc` manual page gives full details of the additional options you can use with the `tcc` command. The following options are some of the most useful:

1. To see a running progress report, include the `-p` option. The `tcc` will then output a line to the terminal as it starts executing each testset.
2. To use an alternative configuration file to `tetexec.cfg`, include the `-x` *filename* option. The file must be in the same format as `tetexec.cfg`.
3. To override a parameter in the execution parameters file, include the `-v` option. For example, to use the run name `XYZ123` rather than the value defined in `tetexec.cfg`, use the command

   `tcc -e -v VSX_NAME=XYZ123` *rest–of–command*
4. In order to execute testsets which failed during a previous run, use

   `tcc -e -r` *code–list other–options old–journal–file*

   where *code–list* is a comma-separated list of result codes to be re-executed, *other–options* are the other `tcc` options (e.g., `-y` or `-n`) and *old–journal–file* is the journal file from which the codes are extracted. For example, to re-execute all the tests that failed with `FAIL`, `UNRESOLVED` and `UNINITIATED` codes from journal file `results/0002e/journal`, use the following command:[3]

   ```
   cd results
   tcc -e -r FAIL,UNRESOLVED,UNINITIATED \
           -s ../scen.exec 0002e/journal
   ```

9.4.7 Path Tracing (OPTIONAL)

Most tests within the VSX test suite are coded for path tracing. Path tracing information may be generated when a test is run by enabling the debugging mechanism. The following VSX parameters, described earlier in this chapter, control VSX debugging and path tracing:

3. The long line in this example has been folded at the `\` character for formatting purposes. The command can be typed all on one line, in which case the `\` character must be omitted.

VSX_DBUG_FLAGS

This determines the level of debugging, and what useful information is generated with the output; e.g., line numbers, function names, etc.

VSX_DBUG_FILE

This determines the output file for the debugging information.

These parameters can be set to generate debugging information either on the command line of the `tcc` by specifying `-v VSX_DBUG_FILE=`*filename* and `-v VSX_DBUG_FLAGS=`*dbug–flags* or by modifying the relevant `tetexec.cfg` file lines.

9.4.8 Debugging Options (OPTIONAL)

This section summarises the currently available debugging options and the flag characters which enable them. Argument lists enclosed in square brackets are optional. Options are separated by colons.

`d[,`*keywords*`]`

Enable output from macros including the specified *keywords*. A null list of keywords implies that all keywords are selected.

`f[,`*functions*`]`

Limit debugger actions to the specified list of *functions* and the functions which they invoke. A null list of functions implies that all functions are selected.

`F`

Mark each debugger output line with the name of the source file containing the macro causing the output.

`l[,`*n*`]`

Specify level of debugging output required, defaults to 1. Use level 2 to get debugging output from library routines.

`L`

Mark each debugger output line with the source file line number of the macro causing the output.

`n`

Mark each debugger output line with the current function nesting depth.

`o[,`*file*`]`

Re-direct the debugger output stream to the specified *file*. The default output file is specified by the configuration variable **VSX_DBUG_FILE**. If this is not set or cannot be opened, `stderr` is used. A null argument list or the file name `-` causes output to be sent to `stderr`.

`p[,`*processes*`]`

Limit debugger actions to the specified *processes*. A null list implies all processes. This is useful for processes which run child processes.

`P`

Mark each debugger output line with the name and PID of the current process. Most useful when used with a process which runs child processes that are also being debugged.

`t[,`*n*`]`

Enable function control flow tracing. The maximum nesting depth is specified by *n* and defaults to 200.

Useful Debugging Setting

A useful setting for path tracing purposes is

```
VSX_DBUG_FLAGS=d,trace:F:L
```

which prints the file name and line number as well as the check point value, and therefore can provide a useful aid to locating the source of a path tracing error.

9.4.9 Debugging Output and the Path Tracing Code

This section describes the correlation between the debugging output and the source code functions.

PATH_TRACE

This is the basic macro to checkpoint program progress. Output associated with a call to `PATH_TRACE` will be in the form:

```
trace: path counter 1
```

Where the test is running correctly, one would expect a single output line for each occurrence of the `PATH_TRACE` macro within the code, and the path counter to increment.

PATH_FUNC_TRACE

`PATH_FUNC_TRACE` calls generate identical style of debugging information to `PATH_TRACE` when a function is diagnosed as having path traced successfully; however, if a function failed its path trace, output of the form

```
trace: path check failed in function call
```

will be produced by the `PATH_FUNC_TRACE` call.

PATH_XS_RPT

This call is made when the test has completed, and wishes to signal a **PASS** result, subject to the path check being a given number. If the path check was successful, then the success will be reported. However, if the path check was unsuccessful, an `unresolved` result will be reported, together with the following diagnostic appearing in both the journal and the debugging output:

`path trace error: path counter` *X*`, expected` *Y*

9.4.10 Executing Tests Directly (OPTIONAL)

When debugging tests it is sometimes useful to execute them directly instead of under the control of `tcc`. When tests are executed in this way the current directory must be the location of the testset executable file. Also the variables **TET_CONFIG** and **TET_CODE** must be set in the environment. Once the tests have been executed the results are found in a file called `tet_xres`.

For example to execute the tests for `write()` directly you would use the commands:

```
TET_CONFIG=$TET_EXECUTE/tetexec.cfg
TET_CODE=$HOME/tet_code
export TET_CONFIG TET_CODE

cd $TET_EXECUTE/tset/POSIX.os/ioprim/write
./T.write
more tet_xres
```

This will execute all the tests in the testset. If you want to execute only specified ICs, give the IC list as an argument:

```
./T.write 1-3,7
```

9.5 TROUBLESHOOTING

You may encounter some of the following problems when you execute tests with `tcc`. This section lists common problems and gives notes explaining how to overcome them.

1. `tcc` refuses to find testsets.

   Ensure you have either set the environment variable **TET_EXECUTE** to the full pathname of the testroot directory, or used the `-a` option of the `tcc` to specify the testroot directory.

2. Tests appear to hang for long periods.

   Some tests do require a long time, as they must wait for timeouts. If the test is not simply waiting but is using processor time, it may be receiving a signal repeatedly. If you interrupt the `tcc` program with a `SIGINT` (e.g., by typing DEL or CTRL-C on the terminal where `tcc` is running), it will terminate the current testset and start the next one. You can also do this by sending a `SIGTERM` signal to the stuck process.

3. The message *IC number not defined for this test case* in journal files.

   There may be a dependency. See journal from whole testset execution to identify IC numbers.

4. Most header and C language tests give FIP results.

   If your C compiler produces output on successful compilations (e.g. an identification message) then these tests will give *Further Information Provided* (FIP) results, meaning the output must be checked manually before it can be considered a pass. It is impractical to check hundreds of FIP results, so to avoid producing them you should set the **VSX_CC** parameter to the name of a shell script which invokes the C compiler and discards the identification message. It must discard only the exact message, not all output. A suitable script to discard a one line message would be:

   ```
   c89 "$@" 2> tmp.err
   code=$?
   grep -v '^Text of message$' tmp.err >&2
   rm -f tmp.err
   exit $code
   ```

5. Some header tests do not compile because the compiler cannot cope with a huge *header*.h.D file.

   You can prevent these huge files from being created by building the header tests with the variable HDRDEFS_NOEXPAND set in the environment. However, this will produce build failures as there will be some nested include files that the

hdrdefs program cannot open. Refer to the TROUBLESHOOTING section in the chapter entitled ‘‘BUILDING VSX’’ for information on how to deal with these build failures.

9.6 EXECUTING VSTH

9.6.1 General Parameters

All of the paramters in this section of `tetexec.cfg` are needed.

9.6.2 Compiler Characteristics

All of the parameters in this section of `tetexec.cfg` are needed.

Many implementions will need to include a `-lpthread` in VSX_LIBS so that the header tests in `PTHR.hdr/misc/` link successfully. Note that the XSH5 requirement that -lpthread makes visibile all symbols with external linkage defined by pthread.h is tested separately in `PTHR.os/general/General/T.General.`

9.6.3 Operating System Characteristics Common to Multiple Subsets

The following execution parameters are used by VSTH. The full descriptions of these parameters are in the generic part of this chapter.

VSX_UID0

VSX_GID0

VSX_INVALID_GID

VSX_INVALID_GNAME

VSX_INVALID_PNAME

VSX_INVALID_UID

VSX_NOSPC_DEV

VSX_TTYNAME

9.6.4 Subset Specific Execution Parameters

VSTH tests will report an unresolved result if any required execution parameter is not set. There are no defaults. The following sections list the parameters, their descriptions and the files within VSTH that use these parameters. The testset code is located under `/tset/PTHR.os`.

VSTH_CFLAGS

VSTH performs header file testing by compiling and, if needed, executing individual test programs extracted from the tset/PTHR.hdr/misc/<testname>/L.<testname> vsxgen archive files at VSTH execution time. This parameter defines the set of compiler flags used by the VSTH header test driver when compiling these test programs.

Where Used

`All`tests

VSTH_LIBS

This parameter defines the set of libraries with which VSTH header file tests are linked at VSTH execution time.

Where Used

`All`tests

VSTH_ADS

This parameter defines the allocation domain size of the implementation. If VSTH_ADS is not set to 1, then tests of features in the priority scheduling feature group or tests which rely on priority scheduling behavior will report an UNTESTED result.

Where Used

```
sched_rt/pthread_attr_setschedpolicy/pthread_attr_setschedpolicy.c:3-
7
cond/pthread_cond_broadcast/pthread_cond_broadcast.c:2
cond/pthread_cond_signal/pthread_cond_signal.c:2
mutex/mutex_init/mutex_init.c:2
mutex/mutex_lock/mutex_lock.c:1
mutexattr_rt/pthread_mutexattr_setprioceiling/pthread_mutexattr_setprioceiling.c:2
mutexattr_rt/pthread_mutexattr_setprotocol/pthread_mutexattr_setprotocol.c:6,9,11
sched/pthread_attr_setschedparam/pthread_attr_setschedparam.c:2,3
```

VSTH_CV_EBUSY

If the implementation supports detection of busy condition variables, for example by pthread_cond_destroy() and pthread_cond_init(), then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
cond/pthread_cond_destroy.c:4
cond/pthread_cond_init.c:4
```

VSTH_DETACH_EINVAL

If the implementation detects thread ids which can not be joined in calls to pthread_detach() set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
thread/pthread_detach/pthread_detach.c:3
```

VSTH_DIR_PREAD

If the implementation allows directories to be read with pread(), then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
ioprim/pread/pread.c:12
```

VSTH_EDEADLK

If the implementation detects pthread_join() deadlocks, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
`thread/pthread_join/pthread_join.c:6`

VSTH_ENXIO_DEV

This parameter identifies a non-existent device used for pread ENXIO testing. If your implementation of pread does not detect ENXIO errors, then set this to 'N' or 'n', otherwise set it to the full pathname of a non-existent device.

Where Used
`ioprim/pread/pread.c:14`
`ioprim/pwrite/pwrite.c:14`

VSTH_FIFO_MIN_PRIORITY

VSTH_FIFO_MAX_PRIORITY

These parameters define the valid priority range for the SCHED_FIFO policy

They exist because in 9945-1:1996, the sched_get_priority_min() and sched_get_priority_max() functions are under control of the _POSIX_PRIORITY_SCHEDULING feature test macro and the functions we're testing are under control of the _POSIX_THREADS_PRIORITY_SCHEDULING feature test macro.

If your implementation defines _POSIX_PRIORITY_SCHEDULING, then the sched_get_priority_min() and sched_get_priority_max() functions are used instead of these execution variables.

Where Used
`SRC/common/vsthlib/vsth_gen.c`

VSTH_RR_MIN_PRIORITY

VSTH_RR_MAX_PRIORITY

These parameters define the valid priority range for the SCHED_RR policy

They exist because in 9945-1:1996, the sched_get_priority_min() and sched_get_priority_max() functions are under control of the _POSIX_PRIORITY_SCHEDULING feature test macro and the functions we're testing are under control of the _POSIX_THREADS_PRIORITY_SCHEDULING feature test macro.

If your implementation defines _POSIX_PRIORITY_SCHEDULING, then the sched_get_priority_min() and sched_get_priority_max() functions are used instead of these execution variables.

Where Used
`SRC/common/vsthlib/vsth_gen.c`

VSTH_OTHER_MIN_PRIORITY

VSTH_OTHER_MAX_PRIORITY

These parameters define the valid priority range for the SCHED_OTHER policy

They exist because in 9945-1:1996, the sched_get_priority_min() and sched_get_priority_max() functions are under control of the _POSIX_PRIORITY_SCHEDULING feature test macro and the functions we're testing are under control of the _POSIX_THREADS_PRIORITY_SCHEDULING feature test macro.

If your implementation defines _POSIX_PRIORITY_SCHEDULING, then the sched_get_priority_min() and sched_get_priority_max() functions are used instead of these execution variables.

Where Used
`SRC/common/vsthlib/vsth_gen.c`

VSTH_GETLOGIN_R_ERANGE

If the implementation supports detection of ERANGE errors by getlogin_r, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
`procenv/getlogin_r/getlogin_r.c:2`

VSTH_INVALID_CANCELSTATE_DETECTED

If the implementation of pthread_setcancelstate() detects invalid values, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
`cancel/pthread_setcancelstate/pthread_setcancelstate.c:3`

VSTH_INVALID_CANCELTYPE_DETECTED

If the implementation of pthread_setcancelstate() detects invalid values, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
`cancel/pthread_setcanceltype/pthread_setcanceltype.c:3`

VSTH_INVALID_GUARDSIZE

This parameter defines an invalid guardsize for the pthread_attr_setguardsize function. If the implementation regards all guardsize values as valid, then set the parameter to 'N' or 'n', otherwise set it to an invalid guardsize value.

Where Used
`threadattr/pthread_attr_setguardsize/pthread_attr_setguardsize.c:3`

VSTH_INVALID_KEY_DETECTED

If the implementation of pthread_setspecific() detects invalid keys, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
key/pthread_setspecific/pthread_setspecific.c:4
```

VSTH_INVALID_POLICY

If the implementation of pthread_attr_setschedpolicy() and pthread_setschedparam() detect invalid scheduling policies, set this variable to the value of an invalid policy, otherwise set it to 'N' or 'n'.

Where Used

```
sched_rt/pthread_setschedparam/pthread_setschedparam.c:3
```

VSTH_INVALID_SIG

Provide a value for an invalid signal number.

Where Used

```
procprim/pthread_kill/pthread_kill.c:4
procprim/sigwait/sigwait.c:8
```

VSTH_INVALID_STACKSIZE

Provide a value for an invalid stacksize attribute of a pthread_attr_t attributes objects.

Where Used

```
threadattr/pthread_attr_setstacksize/pthread_attr_setstacksize.c:3
```

VSTH_KEYS_USED

This parameter defines the number of thread specific data keys used by the test scaffold. It is used to adjust the number of available keys when testing PTHREAD_KEYS_MAX.

Where Used

```
key/pthread_key_create/pthread_key_create.c:5
```

VSTH_KEY_EINVAL

If the implementation detects invalid TSD keys, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
key/pthread_key_delete/pthread_key_delete.c:5
```

VSTH_MXAO_EINVAL

If the implementation detects invalid mutex attribute objects, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
mutex/pthread_mutex_init/pthread_mutex_init.c:8
mutexattr/pthread_mutexattr_destroy/pthread_mutexattr_destroy.c:2
mutexattr_rt/pthread_mutexattr_getprioceiling/pthread_mutexattr_getprioceiling.c:3
mutexattr_rt/pthread_mutexattr_setprioceiling/pthread_mutexattr_setprioceiling.c:4
mutexattr_rt/pthread_mutexattr_getprotocol/pthread_mutexattr_getprotocol.c:2
```

```
mutexattr_rt/pthread_mutexattr_setprotocol/pthread_mutexattr_setprotocol.c:13
mutexattr/pthread_mutexattr_getpshared/pthread_mutexattr_getpshared.c:4
mutexattr/pthread_mutexattr_setpshared/pthread_mutexattr_setpshared.c:4
mutexattr/pthread_mutexattr_gettype/pthread_mutexattr_gettype.c:2
mutexattr/pthread_mutexattr_settype/pthread_mutexattr_settype.c:3
```

VSTH_MX_DEADLK

If the implementation detects mutex deadlocks, set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
mutex/pthread_mutex_lock/pthread_mutex_lock.c:8
```

VSTH_MX_EBUSY

If the implementation detects busy mutex objects, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
mutex/pthread_mutex_destroy/pthread_mutex_destroy.c:2
```

VSTH_MX_EINVAL

If the implementation detects invalid mutex objects, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
mutex/pthread_mutex_destroy/pthread_mutex_destroy.c:3
mutex/pthread_mutex_lock/pthread_mutex_lock.c:10
mutex_rt/pthread_mutex_setprioceiling/pthread_mutex_setprioceiling.c:4
mutex/pthread_mutex_trylock/pthread_mutex_trylock.c:3
mutex/pthread_mutex_unlock/pthread_mutex_unlock.c:7
cond/pthread_cond_timedwait/pthread_cond_timedwait.c:8
cond/pthread_cond_wait/pthread_cond_wait.c:6
```

VSTH_MX_EPERM

If the implementation of pthread_mutex_unlock() supports a return value of EPERM for attempts to unlock a mutex not owned by the caller, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
mutex/pthread_mutex_unlock/pthread_mutex_unlock.c:8
```

VSTH_NOSEEK_DEV

This parameter identifies a device which does not support seeking.

Where Used

```
ioprim/pread/pread.c:3
```

VSTH_NTS_EINVAL

If the implementation recognizes struct timespecs with negative values for tv_sec or tv_nsec as invalid, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
```
cond/pthread_cond_timedwait/pthread_cond_timedwait.c:9
```

VSTH_PAGSI_SUPPORTED

If the implementation does not support the _POSIX_THREAD_PRIORITY_SCHEDULING or _XOPEN_REALTIME_THREADS feature groups but implements the pthread_attr_getinheritsched() and pthread_attr_setinheritsched() functions then set this variable to 'Y' to cause the ENOSYS test to report an unsupported result, otherwise leave it unset.

Where Used
```
sched_rt/pthread_attr_getinheritsched/pthread_attr_getinheritsched.c:2
sched_rt/pthread_attr_setinheritsched/pthread_attr_setinheritsched.c:4
```

VSTH_PAGSP_SUPPORTED

If the implementation does not support the _POSIX_THREAD_PRIORITY_SCHEDULING or _XOPEN_REALTIME_THREADS feature groups but implements the pthread_attr_getschedpolicy() and pthread_attr_setschedpolicy() functions then setting this variable to 'Y' or 'y' will cause the ENOSYS test to report an unsupported result, otherwise leave it unset.

Where Used
```
sched_rt/pthread_attr_getschedpolicy/pthread_attr_getschedpolicy.c:2
sched_rt/pthread_attr_setschedpolicy/pthread_attr_setschedpolicy.c:10
```

VSTH_PAGSS_SUPPORTED

If the implementation does not support the _POSIX_THREAD_PRIORITY_SCHEDULING or _XOPEN_REALTIME_THREADS feature groups but implements the pthread_attr_getscope() and pthread_attr_setscope() functions then setting this variable to 'Y' or 'y' will cuase the ENOSYS test to report an unsupported result, otherwise leave it unset.

Where Used
```
sched_rt/pthread_attr_getscope/pthread_attr_getscope.c:2
sched_rt/pthread_attr_setscope/pthread_attr_setscope.c:4
```

VSTH_PGSP_SUPPORTED

If the implementation does not support the _POSIX_THREAD_PRIORITY_SCHEDULING or _XOPEN_REALTIME_THREADS feature groups but implements the pthread_getschedparam() and pthread_setschedparam() functions then set this variable to 'Y' to cause the ENOSYS test to report an unsupported result, otherwise leave it unset.

Where Used
```
sched_rt/pthread_getschedparam/pthread_getschedparam.c:3
sched_rt/pthread_setschedparam/pthread_setschedparam.c:2
```

VSTH_PRIO_EINVAL

If the implementation detects invalid prioceiling attribute values, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

`mutex_rt/pthread_mutex_setprioceiling/pthread_mutex_setprioceiling.c:3`

VSTH_PROTO_EPERM

If the implementation requires appropriate privilege to set the scheduling protocol, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

`mutexattr_rt/pthread_mutexattr_setprotocol/pthread_mutexattr_setprotocol.c:14`

VSTH_PROTO_UNSUP

This parameter defines a value for an unsupported scheduling protocol.

Where Used

`mutexattr_rt/pthread_mutexattr_setprotocol/pthread_mutexattr_setprotocol.c:12`

VSTH_PSA_EINVAL

If the implementation detects invalid values for process-shared attributes, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

`condattr/pthread_condattr_setpshared/pthread_condattr_setpshared.c:5`

VSTH_PSHARED_EINVAL

This parameter defines an invalid value for the process shared attribute. If the implementation under test does not detect invalid rwlockattr pshared values, then leave this variable unitialized.

Where Used

`rwlockattr/pthread_rwlockattr_setpshared/pthread_rwlockattr_setpshared.c:5`

VSTH_PS_EINVAL

If the implementation supports detection of invalid process-shared attribute values, set this variable to the value of an unsupported process-shared attribute value.

Where Used

`mutexattr/pthread_mutexattr_setpshared/pthread_mutexattr_setpshared.c:5`

VSTH_READDIR_R_EBADF

If the implementation of readdir_r() detects that dirp does not refer to an open directory stream, then this parameter should be set to 'Y' or

Where Used

`files/readdir_r/readdir_r.c:5`

VSTH_RWLAO_EINVAL

If the implementation detects invalid read-write lock attribute objects, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
rwlock/pthread_rwlock_init/pthread_rwlock_init.c:9
rwlockattr/pthread_rwlockattr_destroy/pthread_rwlockattr_destroy.c:2
rwlockattr/pthread_rwlockattr_getpshared/pthread_rwlockattr_getpshared.c:2
rwlockattr/pthread_rwlockattr_setpshared/pthread_rwlockattr_setpshared.c:5
```

VSTH_RWL_DEADLK

If the implementation detects read-write lock deadlocks, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
rwlock/pthread_rwlock_rdlock/pthread_rwlock_rdlock.c:6
rwlock/pthread_rwlock_tryrdlock/pthread_rwlock_tryrdlock.c:6
rwlock/pthread_rwlock_trywrlock/pthread_rwlock_trywrlock.c:5
rwlock/pthread_rwlock_wrlock/pthread_rwlock_wrlock.c:4
```

VSTH_RWL_EBUSY

If the implementation detects busy read-write locks, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
rwlock/pthread_rwlock_destroy/pthread_rwlock_destroy.c:2
rwlock/pthread_rwlock_init/pthread_rwlock_init.c:8
```

VSTH_RWL_EINVAL

If the implementation detects invalid read-write locks, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
rwlock/pthread_rwlock_rdlock/pthread_rwlock_rdlock.c:5
rwlock/pthread_rwlock_tryrdlock/pthread_rwlock_tryrdlock.c:5
rwlock/pthread_rwlock_trywrlock/pthread_rwlock_trywrlock.c:4
rwlock/pthread_rwlock_unlock/pthread_rwlock_unlock.c:5
rwlock/pthread_rwlock_wrlock/pthread_rwlock_wrlock.c:3
```

VSTH_RWL_EPERM

If the implementation detects EPERM errors for operations on rwlock objects, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
rwlock/pthread_rwlock_unlock/pthread_rwlock_unlock.c:6
```

VSTH_RWL_MAX

This parameter defines the maximum number of read locks allowed by the implementation on a single rwlock object. If the implementation does not impose a deterministic limit, leave the variable uninitialized.

Where Used
rwlock/pthread_rwlock_rdlock/pthread_rwlock_rdlock.c:7
rwlock/pthread_rwlock_tryrdlock/pthread_rwlock_tryrdlock.c:7
rwlock/pthread_rwlock_unlock/pthread_rwlock_unlock.c:1

VSTH_SCHED_EPERM

If the implementation requires appropriate privilege to set the scheduling priority of a thread to the maximum supported priority for the SCHED_FIFO scheduling policy, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
mutex_rt/pthread_mutex_getprioceiling/pthread_mutex_getprioceiling.c:4
mutex/pthread_mutex_init/pthread_mutex_init.c:9
mutex_rt/pthread_mutex_setprioceiling/pthread_mutex_setprioceiling.c:6
sched_rt/pthread_setschedparam/pthread_setschedparam.c:5

VSTH_SIG_EINVAL

If the implementation detects invalid signals, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
procprim/sigwait/sigwait.c:8

VSTH_TID_EINVAL

If the implementation detects invalid thread ids, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
sched_rt/pthread_setschedparam/pthread_setschedparam.c:7

VSTH_TID_ESRCH

If the implementation detects non-existent thread ids, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used
cancel/pthread_cancel/pthread_cancel.c:17
thread/pthread_detach/pthread_detach.c:4
sched_rt/pthread_getschedparam/pthread_getschedparam.c:4

VSTH_TYPE_EINVAL

This parameter defines an invalid value for the mutex type attribute.

Where Used
mutexattr/pthread_mutexattr_settype/pthread_mutexattr_settype.c:2

VSTH_UCVAO_EINVAL

If the implementation recognizes unitialized condition variable attributes objects as invalid, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
cond/pthread_cond_init/pthread_cond_init.c:3
condattr/pthread_condattr_destroy/pthread_condattr_destroy.c:2
condattr/pthread_condattr_getpshared/pthread_condattr_getpshared.c:3
```

VSTH_UCV_EINVAL

If the implementation recognizes unitialized condition variables as invalid, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
cond/pthread_cond_broadcast/pthread_cond_broadcast.c:4
cond/pthread_cond_destroy/pthread_cond_destroy.c:5
cond/pthread_cond_signal/pthread_cond_signal.c:3
cond/pthread_cond_timedwait/pthread_cond_timedwait.c:7
cond/pthread_cond_wait/pthread_cond_wait.c:5
```

VSTH_ULIMIT_BLKS

This parameter defines the lower limit on the process file size limit in blocks of 512 bytes. It may be the same as VSX_ULIMIT_BLKS.

Where Used

```
ioprim/pwrite/pwrite_10.c:16
ioprim/pwrite/pwrite_3.c:16
```

VSTH_UPAO_EINVAL

If the implementation recognizes an unitialized pthread_attr_t as an invalid attributes object, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
threadattr/pthread_attr_getguardsize/pthread_attr_getguardsize.c:3
threadattr/pthread_attr_setguardsize/pthread_attr_setguardsize.c:2
sched_rt/pthread_attr_setinheritsched/pthread_attr_setinheritsched.c:2
```

VSTH_VALID_CONCURRENCY_LEVEL

This parameter defines a valid concurrency level for pthread_setconcurrency().

Where Used

```
concurrency/pthread_getconcurrency/pthread_getconcurrency.c:1
concurrency/pthread_setconcurrency/pthread_setconcurrency.c:1,4
```

VSTH_VALID_GUARDSIZE

This parameter defines a valid value for the guardsize attribute of a pthread_attr_t attributes object

Where Used

```
threadattr/pthread_attr_getguardsize/pthread_attr_getguardsize.c:1
threadattr/pthread_attr_setguardsize/pthread_attr_setguardsize.c:1
```

VSTH_VALID_STACKSIZE

This parameter defines a valid value for the stacksize attribute of a pthread_attr_t attributes objects

Where Used

```
threadattr/pthread_attr_getstackaddr/pthread_attr_getstackaddr.c:1
threadattr/pthread_attr_getstacksize/pthread_attr_getstacksize.c:1
threadattr/pthread_attr_setstackaddr/pthread_attr_setstackaddr.c:1
threadattr/pthread_attr_setstackaddr/pthread_attr_setstackaddr.c:2
threadattr/pthread_attr_setstackaddr/pthread_attr_setstackaddr.c:3
threadattr/pthread_attr_setstacksize/pthread_attr_setstacksize.c:1
threadattr/pthread_attr_setstacksize/pthread_attr_setstacksize.c:4
threadattr/pthread_attr_setstacksize/pthread_attr_setstacksize.c:5
```

VSTH_RTHITS_MIN

This parameter defines the minimum number of times we attempt to achieve actual reentrant conditions during MT safety testing. Testing is stopped when the number of times we successfully set up reentrant conditions equals this value.

Where Used

SRC/common/vsthlib/vsth_fw.c
general/General/General.c:4

VSTH_RTTRIES_MAX

This parameter defines the maximum number of attempts to achieve VSTH_RTHITS_MIN number of successful reentrant conditions. Testing is stopped when the number of tries equals this value, whether or not VSTH achieves VSTH_RTHITS_MIN actual reentrant tests. Implementations of the vsth_setconcurrency() function in SRC/userintf.c should be designed to ensure that the number of times reentrant conditions are achieved approaches VSTH_RTHITS_MIN.

Where Used

SRC/common/vsthlib/vsth_fw.c
general/General/General.c:4

VSTH_SIG_IGN

Some notable threads implementations install signal handlers for standard signals such as SIGUSR1 and communicate with thread manager processes using these signals. This tends to upset TETware. If the implementation under test does so, set this variable to a comma separated list of signal numbers used. Failure to do so will result in the tcm reporting unexpected signals and pre-mature test case termination.

Where Used

SRC/common/vsthlib/vsth_gen.c

VSTH_READDIR_R_EBADF

If the implementation supports the return of EBADF errors from readdir_r, then set this parameter to 'Y', otherwise set it to 'N'.

Where Used

```
files/readdir_r/readdir_r:5
```

VSTH_UNSUP_INHERITSCHED_VAL

If the implementation detects unsupported values for the inheritsched attribute, define an unsupported value for this attribute.

Where Used

```
sched_rt/pthread_attr_setinheritsched/pthread_attr_setinheritsched:3
```

VSTH_UNSUP_CSCOPE_VAL

If the implementation detects unsupported values for the contentionscope attribute, define an unsupported value for this attribute.

Where Used

```
sched_rt/pthread_attr_setscope/pthread_attr_setscope:3
```

VSTH_RR_INTERVAL_SEC

VSTH_RR_INTERVAL_NSEC

If the implementation does not support the _POSIX_PRIORITY_SCHEDULING feature group, but does support the _POSIX_THREAD_PRIORITY_SCHEDULING feature group, provide values for the SCHED_RR scheduling interval.

Where Used

```
sched_rt/pthread_attr_setschedpolicy/pthread_attr_setschedpolicy:7
```

VSTH_RWL_INIT_PRIV

If the implementation requires appropriate privilege to set the process shared attribute of a read-write lock to PTHREAD_PROCESS_SHARED, then set this variable to 'Y' or 'N'. Otherwise set it to 'N' or 'n'.

Where Used

```
rwlock/pthread_rwlock_init/pthread_rwlock_init:7
```

VSTH_INVALID_POLICY=

If the implementation of pthread_attr_setschedpolicy() detects invalid policy values, then set the parameter to the value of an invalid policy, otherwise set it to 'N' or 'n'.

Where Used

```
sched_rt/pthread_attr_setschedpolicy/pthread_attr_setschedpolicy.c:8
sched_rt/pthread_setschedparam/pthread_setschedparam.c:3
```

VSTH_UNSUPPORTED_POLICY=

If the implementation of pthread_attr_setschedpolicy() detects unsupported policy values, then set the parameter to the value of an unsupported policy, otherwise set it to 'N' or 'n'.

Where Used

```
sched_rt/pthread_attr_setschedpolicy/pthread_attr_setschedpolicy.c:9
sched_rt/pthread_setschedparam/pthread_setschedparam.c:4
```

VSTH_INVALID_INHERITSCHED_VAL

If the implementation of pthread_attr_setinheritsched() detects invalid values for the inheritsched attribute, then set this variable to the an invalid value, otherwise set it to 'N' or 'n'. Expected range: integer|N|n

Where Used

`sched_rt/pthread_attr_setinheritsched/pthread_attr_setinheritsched.c:2`

VSTH_SELF_EDEADLK_DETECTED

If the implementation of pthread_join() detects deadlocks when the target thread is the calling thread, then set this variable to 'Y' or 'y', otherwise set it to 'N' or 'n'.

Where Used

`thread/pthread_join/pthread_join.c:6`

VSTH_REFERENTIAL_EDEADLK_DETECTED

If the implementation of pthread_join() detects deadly embrace or circular deadlocks, then set this variable to 'Y' or 'y', otherwise set it to 'N' or 'n'.

Where Used

`thread/pthread_join/pthread_join.c:6`

VSTH_SPC_EPERM

If the implementation requires appropriate privilege to set the prioceiling attribute of a mutex attributes object, then set this variable to 'Y' or 'y', otherwise set it to 'N' or 'n'. Expected range: Y|y|N|n

Where Used

`mutexattr_rt/pthread_mutexattr_setprioceiling/pthread_mutexattr_setprioceiling.c:5`

VSTH_GPC_EPERM

If the implementation requires appropriate privilege to get the prioceiling attribute of a mutex attributes object, then set this variable to 'Y' or 'y', otherwise set it to 'N' or 'n'. Expected range: Y|y|N|n

Where Used

`mutexattr_rt/pthread_mutexattr_getprioceiling/pthread_mutexattr_getprioceiling.c:4`

VSTH_ICONV_CODESET1

VSTH_ICONV_CODESET2

Identify two codesets supported by your implementation of iconv_open()

Where Used

`cancel/pthread_cancel/pthread_cancel.c:9`

VSTH_SUPP_GID

If the implementation supports supplementary group ids, identify a supplementary gid to which user vsxg0 belongs.

Where Used

`cancel/pthread_cancel/pthread_cancel.c:9`

Action Points

1. Edit TESTROOT/tetexec.cfg and supply values for all VSTH execution parameters.

9.7 Executing the VSX Test Suite

9.7.1 Executing Selected Testsets (Optional)

To execute selected parts of the test suite, you can use the `-y` and `-n` options of the `tcc` command. For example, to execute just the tests related to reader-writer locks use the command:

```
tcc -p -e -s scen.exec -y /rwlock
```

or to execute only the header file tests, use the command:

```
tcc -p -e -s scen.exec -y PTHR.hdr -n PTHR.os
```

or to execute everything except the Realtime Threads tests, use the command:

```
tcc -p -e -s scen.exec -n _rt
```

9.8 Troubleshooting

VSTH_DEBUG

This execution variable causes additional diagnostic information to be produced in the journal. It should be left unset when executing VSTH as a formal conformance test.

Where Used

`SRC/common/vsthlib/vsth_gen.c`

VSTH_TIMEOUT

The TET `tet_pthread_create()` interface accepts a timeout parameter. TET will terminate the execution of a thread created with `tet_pthread_create()` after this timeout period expires. The default timeout period is defined in SRC/INC/vsth/vsth.h. It may be changed by assigning a new value to VSTH_TIMEOUT. The timeout value is specified in seconds. Note that some tests partition VSTH_TIMEOUT into smaller intervals to perform staging and therefore VSTH_TIMEOUT should not be set to a value less than 10 seconds.

Where Used

`SRC/common/vsthlib/vsth_gen.c`

10. REPORTING

10.1 INTRODUCTION

You can use the VSX reporting program to format reports from the results of the building and execution stages. You can generate reports from a complete journal file or from the results for the part you want to use. In addition, you can generate summary reports which summarise the results for testsets in a given section, area or testset. When you use the reporting program, you can use other options to control the length and width of the text on the page.

There is a separate reporting program for producing POSIX conformance reports.

When you want to compare the results in several journal files, you can use the comparative reporting program, explained at the end of this chapter.

10.2 THE REPORTING PROGRAMS

The VSX reporting program, `vrpt`, formats the results in the VSX 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 `vsx0` home directory. Include this directory in your `PATH`.

The POSIX reporting program, `prpt`, produces the definitive POSIX conformance report from journal files created by execution in POSIX modes. It should only be used with test packages that provide POSIX assertion numbers in their test descriptions. Usually this will be the case only for test packages containing tests that can be run in `POSIX90` mode.

NB. `prpt` does not provide the detailed diagnostic information that `vrpt` provides, and it gives results against POSIX assertion numbers, not the VSX testset names and test numbers that would be needed to re-run failed tests. For these reasons, it is recommended that POSIX mode users run `vrpt` until they are happy with the results of the tests; `prpt` may then be run to produce final documentation of the test results.

10.3 REPORTING PROGRAM USAGE SUMMARY

`vrpt [-l`*level*`] [-r`*coverage*`] [-f`*file*`] [-v] [-H] [-p] [-P] [-L`*len*`] [-W`*wid*`] [`*file* `...]`

`prpt [-H] [-L`*len*`] [-W`*wid*`] [`*file* `...]`

10.4 REPORTING PROGRAM OPTIONS

10.4.1 Reporting on the Entire Journal

To generate a report on an entire journal file, use the command:

`vrpt` *journal–file*

10.4.2 Reporting on a Section or Area (OPTIONAL)

The names of the sections and areas are the same as those listed in the building chapter.

By default, the reporting program generates a report from the complete journal file. To produce a report from results for part of the test suite, use the `-r` option of `vrpt`, followed by the name of the section or area you want to use. For example, to report on the `POSIX.os` testset results in the latest journal file, use the command:

`vrpt -r POSIX.os` *journal–file*

To report on the `streamio` area within the `ANSI.os` section, use the command:

```
vrpt -r ANSI.os/streamio journal–file
```

Note that the Conformance Summary produced as part of the cover pages on validation test reports always contains the complete results for the journal file(s) being processed. Only the body of the report is affected by the `-r` option.

10.4.3 Reporting on Individual Testsets (OPTIONAL)

You can also use the `-r` option for `vrpt` to report on the results of individual testsets, or a range of testsets. The `-P` option is useful here to stop the cover pages being produced. For example, to report on the results of a single testset for the system interface `write()`, use the command:

```
vrpt -r POSIX.os/ioprim/write -P journal–file
```

To report on the results from all the testsets between the system calls `read()` and `write()`, use the following command:[5]

```
vrpt -r POSIX.os/ioprim/read:POSIX.os/ioprim/write \
        -P journal–file
```

10.4.4 Summary Reports (OPTIONAL)

To generate a report which summarises the testset results by section or area, use the `-l` option. The area summary report, which is useful as a management summary, is given in tabular format. For example, to generate a summary report at section level, use the command:

```
vrpt -l sect journal–file
```

For area level reports, use the command:

```
vrpt -l area journal–file
```

10.4.5 Varying the Text Format (OPTIONAL)

You can use the `-L` page length and `-W` page width options to format the text in reports according to your paper size. When you reduce the page width, long output lines are automatically wrapped onto the next line of the report. Note that the Conformance Summary produced as part of the cover pages contains a wide table which does not get wrapped, so if you are using a page width of less than the default 80 characters, you will probably want to disable the cover pages by using the `-P` option. For example, to format the text using a page length of 50 lines and width of 64 characters, use the command:

```
vrpt -L50 -W64 -P journal–file
```

10.4.6 Additional Options (OPTIONAL)

The `vrpt` user manual, in part 4 of this guide, gives full details of the additional options you can use with the `vrpt` reporting program.

5. The long line in this example has been folded at the `\` character for formatting purposes. The command can be typed all on one line, in which case the `\` character must be omitted.

Action Points

1. Log in as the user `vsx0`.
2. Check that the environment variable PATH is set correctly.
3. Change to the directory `results` if required.
4. Give the command `vrpt` with the options you want to use, on a journal file generated from the results of running the `tcc`.

10.5 POSIX CONFORMANCE REPORTING

A separate reporting program `prpt` produces POSIX conformance reports from execution journal files. These reports consist only of the result codes for each POSIX assertion; failure information produced by the tests is not included. Use the standard reporting program, `vrpt`, to generate reports with the details of test failures.

Options
You can use the `-W` and `-L` options, for page width and page length, with `prpt`.

Action Points

1. Log in as the user `vsx0`.
2. Check that the environment variable PATH is set correctly.
3. Change to the directory `results` if required.
4. Give the command `prpt` with the options you want to use, on journal files generated from the results of running the `tcc`.

10.6 COMPARATIVE REPORTING

You can use an alternative reporting program to compare the results in a number of different journal files. The reporting program `vrptm` enables you to compare the results from tests on a range of machines, or from a series of execution runs on the same machine with different software releases.

The comparative reporting program handles results from up to five journal files on the default page width of 80 columns (more on wider pages). The successes and failures are printed in tables, without any extra information about the reasons for tests failing. Use the standard reporting program `vrpt` to generate reports with the details of test failures.

Options
You can use the `-W` and `-L` options, for page width and page length, with `vrptm`.

Action Points

1. Log in as the user `vsx0`.
2. Check that the environment variable PATH is set correctly.
3. Change to the directory `results` if required.
4. Give the command `vrptm` with the options you want to use, on journal files generated from the results of running the `tcc`.

10.7 SAMPLE REPORT OUTPUT

10.7.1 `vrpt` Sample Output

Conformance Summary Information

```
                        X/OPEN Verification Suite
Test-Set Summary                                          Test-Set Summary


CONFORMANCE Summary

         This is to certify that this system when tested for conformance
         to POSIX.1-1990 achieved the results below.
```

Section	TOTALS Expect	Actual	Succeeded	Failed	Warnings	FIP	Unresolved	Uninitiated	Unsupported	Untested	NotInUse
ANSI.hdr	586	586	260	18	0	0	0	0	307	0	1
ANSI.os F	1676	1676	1638	6	2	0	0	0	3	0	27
ANSI.os M	1676	1676	97	0	0	0	0	0	0	0	1579
POSIX.hdr	450	450	233	13	0	0	0	0	198	0	6
POSIX.os F	1434	1434	1330	9	0	3	0	0	57	1	34
POSIX.os M	1434	1434	4	0	0	0	0	0	0	0	1430
TOTAL	7256	7256	3562	46	2	3	0	0	565	1	3077

```
         Number of amendments _____________

                                                  ____________________
                                                  Signature/Date



Test Agency: UniSoft                                  System Tested: oursys
Test Date:   Apr 11, 1997                                        Page 4
```

Test Results

```
Test-Set Summary                                              Test-Set Summary

         Test-Set Results:
         ----------------

         Test-Set Started: 19:55:04

         Test-Set Ended:   19:55:06

         Test-Set Results Summary:
         ------------------------

            1  Tests Executed
            1  Tests Succeeded


         Test-Set Name: /tset/ANSI.os/charhandle/Miscntrl/T.iscntrl
         ----------------------------------------------------------

         Test-Set Results:
         ----------------

         Test-Set Started: 19:55:07

         Test-Set Ended:   19:55:08

         Test-Set Results Summary:
         ------------------------

            2  Tests Executed
            2  Tests Succeeded


         Test-Set Name: /tset/ANSI.os/charhandle/Miscntrl_X/T.iscntrl_X
         --------------------------------------------------------------

         Test-Set Results:
         ----------------

         Test-Set Started: 19:55:09

         Test-Set Ended:   19:55:11

         Test-Set Results Summary:
         ------------------------

            1  Tests Executed
            1  Tests Succeeded


         Test-Set Name: /tset/ANSI.os/charhandle/Misdigit/T.isdigit
         ----------------------------------------------------------

         Test-Set Results:
         ----------------

Test Agency: UniSoft                                  System Tested: oursys
Test Date:   Apr 11, 1997                                            Page 6
```

Summary Information

```
                          X/OPEN Verification Suite
Test-Set Summary                                                Test-Set Summary


         Section Name: ANSI.os
         ---------------------

         Section Started: 19:54:56


         Section Ended:   22:35:59

         Section Results Summary:
         ------------------------

                  10       Areas Containing   296  Test-Sets Completed

                  3352  Tests Executed
                  1735  Tests Succeeded
                     6  Tests Failed
                     2  Tests Warning
                     3  Tests Unsupported
                  1606  Tests Not In Use
```

```
Test Agency: UniSoft                                  System Tested: oursys
Test Date:   Apr 11, 1997                                        Page 119
```

```
                          X/OPEN Verification Suite
Test-Set Summary                                                Test-Set Summary

Test Parameters:
         TET_OUTPUT_CAPTURE   =     False
         TET_RESCODES_FILE    =     tet_code
         TET_VERSION          =     1.10
         TEST_MODE            =     POSIX90
         TEST_PACKAGES        =     VSX4.4.1
         VSXDIR               =     /user4/TET/vsx4/SRC
         VSX_DBUG_FLAGS       =
         VSX_DBUG_FILE        =     /user4/TET/vsx4/TESTROOT/dbug.out
         VSX_NAME             =
         VSX_OPER             =     Joe Programmer
         VSX_ORG              =     UniSoft
         VSX_PATH             =
         VSX_SYS              =     oursys
         VSX_UID0             =     146
         VSX_UID1             =     147
         VSX_UID2             =     149
         VSX_GID0             =     200
         VSX_GID1             =     201
         VSX_GID2             =     202
         TET_SIG_IGN          =     12
         TET_SIG_LEAVE        =
         VSX_AL_ACCURACY      =
         VSX_BLKDEV_FILE      =     /dev/mt/1m
         VSX_CHRDEV_FILE      =     /dev/rmt/1m
         VSX_CLOCK_ERR        =
         VSX_CLOSEDIR_EBADF   =     Y
         VSX_FCNTL_MAXLOCK    =     400
         VSX_FP_SOFTWARE      =
         VSX_INVALID_GID      =
         VSX_INVALID_GNAME    =
         VSX_INVALID_PC       =
         VSX_INVALID_PNAME    =
         VSX_INVALID_SC       =
         VSX_INVALID_UID      =
         VSX_INVAL_SIG        =
         VSX_LINK_DIR_SUPP    =     Y
         VSX_LINK_FILESYS_SUPP =    N
         VSX_MOUNT_DEV        =     /dev/dsk/c1d0s6
         VSX_NONEXEC_FILE     =     .
         VSX_NOSPC_DEV        =     /dev/dsk/c1d0s6
         VSX_PRIV_ACCESS_SUPP =     Y
         VSX_PRIV_CHOWN_SUPP  =     Y
         VSX_READDIR_EBADF    =     Y
         VSX_REMOVE_DIR_EBUSY =     S
         VSX_RENAME_DIR_EBUSY =     S
         VSX_RENAME_DIR_WPERM_REQD =         N
         VSX_ROFS             =     /dev/dsk/c1d0s6
         VSX_SET_ID_MODES_SUPP =    Y
         VSX_SETPGID_SUPPORTED =    Y
         VSX_SIGSET_EINVAL    =     Y
         VSX_SYS_OPEN_MAX     =     600
         VSX_TTYNAME          =     /dev/tty0p3
         VSX_TTYUSER          =     vsx0

Test Agency: UniSoft                                  System Tested: oursys
Test Date:   Apr 11, 1997                                        Page 120
```

Test Failure Information

```
                        X/OPEN Verification Suite
Test-Set Summary                                              Test-Set Summary

         Test-Set Name: /tset/POSIX.os/procprim/sigaddset/T.sigaddset
         ------------------------------------------------------------

         Test-Set Results:
         ----------------

         Test-Set Started: 02:18:15

         Test-Set Ended:   02:18:16

         Test-Set Results Summary:
         ------------------------

            2  Tests Executed
            2  Tests Succeeded


         Test-Set Name: /tset/POSIX.os/procprim/sigconcept/T.sigconcept
         --------------------------------------------------------------

         Test-Set Results:
         ----------------

         Test-Set Started: 02:18:16

         Test Results:

         ************************************************************************
         /tset/POSIX.os/procprim/sigconcept/T.sigconcept 22    Failed

                  Test Description:
         If _POSIX_JOB_CONTROL is defined, setting a signal action to SIG_DFL
         for a SIGCHLD signal that is pending shall cause the pending signal to
         be discarded.
         Posix Ref: Component Signal Concepts Assertion 3.3.1.3-29(C)

                  Test Strategy:
         FORK a child process
         CHILD process:
           SET the SIGCHLD signal action to signal catching function
           BLOCK the SIGCHLD signal and send to itself
           VERIFY the SIGCHLD signal is not received
           VERIFY the SIGCHLD signal is pending
           SET the SIGCHLD signal action to SIG_DFL
           VERIFY the SIGCHLD signal is not pending
           UNBLOCK the SIGCHLD signal
           EXIT with the exit code set to the number of any caught signal,
           otherwise 0
         PARENT process:
           VERIFY the SIGCHLD signal was discarded

                  Test Information:

Test Agency: UniSoft                                  System Tested: oursys
Test Date:   Apr 11, 1997                                          Page 23
```

```
                          X/OPEN Verification Suite
Test-Set Summary                                                Test-Set Summary


         signal 18 (SIGCHLD) still pending after sigaction()
         deletion reason: waitsync() failed, errno 4
         ************************************************************************

         Test-Set Ended:   03:16:35

         Test-Set Results Summary:
         -------------------------

           37  Tests Executed
           36  Tests Succeeded
            1  Tests Failed
```

10.7.2 `prpt` Sample Output

```
POSIX Summary                                                    POSIX Summary


Journal File:              /user4/TET/vsx4/results/0002e/journal
Test Date:                 Fri Apr 11 19:26:25 1997
Test Agency:               UniSoft
Test System:               oursys
Test Operator:             Joe Programmer
Test Parameters:
         TET_OUTPUT_CAPTURE   =     False
         TET_RESCODES_FILE    =     tet_code
         TET_VERSION          =     1.10
         TEST_MODE            =     POSIX90
         TEST_PACKAGES        =     VSX4.4.1
         VSXDIR               =     /user4/TET/vsx4/SRC
         VSX_DBUG_FLAGS       =
         VSX_DBUG_FILE        =     /user4/TET/vsx4/TESTROOT/dbug.out
         VSX_NAME             =
         VSX_OPER             =     Joe Programmer
         VSX_ORG              =     UniSoft
         VSX_PATH             =
         VSX_SYS              =     oursys
         VSX_UID0             =     146
         VSX_UID1             =     147
         VSX_UID2             =     149
         VSX_GID0             =     200
         VSX_GID1             =     201
         VSX_GID2             =     202
         TET_SIG_IGN          =     12
         TET_SIG_LEAVE        =
         VSX_AL_ACCURACY      =
         VSX_BLKDEV_FILE      =     /dev/mt/1m
         VSX_CHRDEV_FILE      =     /dev/rmt/1m
         VSX_CLOCK_ERR        =
         VSX_CLOSEDIR_EBADF   =     Y
         VSX_FCNTL_MAXLOCK    =     400

                           X/OPEN Verification Suite
POSIX Summary                                                    POSIX Summary


         GETENV               4.6.1.1-01(A)      PASS
         GETENV               4.6.1.1-02(C)      UNSUPPORTED
         GETENV               4.6.1.1-03(C)      UNSUPPORTED
         GETENV               4.6.1.2-04(A)      PASS
         GETENV               4.6.1.2-05(A)      PASS
         GETENV               4.6.1.2-06(A)      PASS
         GETENV               4.6.1.2-07(A)      PASS
         GETENV               4.6.1.3-08(D)      UNTESTED


                                                                 Page 23
```

10.8 TROUBLESHOOTING

This section lists known problems, and gives notes on how to overcome them. You may encounter the following problem when you run `vrpt`.

1. `vrpt` gives the error message ‘‘received SIGPIPE’’.

   If the `vrpt` output was not being piped to another process, e.g. a pager, which exited before reading all the output, then this may be due to `awk` becoming overwhelmed and dumping core. Try using the `-t` option to truncate test failure information to a manageable number of lines. If `awk` still dumps core, replacing `awk` with `nawk` or `gawk` may cure the problem.

11. INTERPRETING VSX RESULTS

11.1 INTRODUCTION

To interpret the results of the VSX 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 VSX journal file, as explained in the chapter entitled ‘‘REPORTING’’. Test descriptions and strategy are include in reports generated with `vrpt` for failed tests. Test descriptions are also provided as a VSX manual page, to be found under the `MAN/tset` directory, which you can print out using the utility `[nt]roff -man`.

11.2 TEST RESULTS

11.2.1 Failed

C Language
The reported information for failed C language tests includes the C source code which was used in the test. You can extract the code from the VSX journal file to use in further testing.

Operating System
The test source code for failed operating system 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 an operating system 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. These may be because of incorrect parameters, preceding failures or external events, which are described in the following paragraphs.

Incorrect Parameters
Most tests reported this way cannot be run because a parameter is not set correctly in the execution parameters file `tetexec.cfg`. The test report always identifies the tests which cannot run because of incorrect parameters. For some tests, you can correct the parameter and re-run the tests. For others, you may not be able to correct the parameter because the resources required are not available on your system.

Incorrect Entries in `userintf.c` (Implementation Specific Routines)
Tests may be reported as `unresolved` or `uninitiated` because of incorrect entries in `SRC/userintf.c`. If failures of user-supplied functions are reported, you will need to check this file. See ‘‘User-supplied Interface Routines’’ in the chapter entitled ‘‘CONFIGURING VSX’’ for more details.

Preceding Failures
When earlier tests have failed, some tests cannot be performed. Before you can re-run the tests, you must resolve the problem in the preceding failed tests.

External Events
When an external event occurs unexpectedly, tests may not be performed. Investigate the reason the test has not been run as for a failed test.

11.2.3 Unreported

When a test is marked as `unreported` a major error has occurred during the testset execution. VSX tries to avoid such errors as far as possible. However, if you terminate a testset with the signal `SIGTERM`, tests will be `unreported`. Investigate the cause of the major error as for a failed test.

11.2.4 Warning

Whenever a warning is given, the functionality is acceptable, but you should be aware that later revisions of the relevant standards or specifications may change the requirements in this area. See the appendix entitled ‘‘TESTS GIVING WARNINGS’’ for a list of the tests which may give warnings and the reasons for them.

11.2.5 FIP (Further Information Provided)

When a test has succeeded, additional information may sometimes be given which needs to be inspected. Some examples are given below:

C Language
The additional information is output by the compiler, usually compiler warnings, although the compilation and execution of the test has been successful.

Checking
Information which cannot be checked automatically by a test is given for you to validate. For example, the system name and node name are given by the VSX4 `uname` testset.

11.2.6 Unsupported

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

For example, in some modes the job control features are optional. VSX will recognise that they are unsupported on a particular system and report this.

11.2.7 Not In Use

Where no macro version of an interface exists, or separate macro and function testing is not required, the macro version of the testset will report all tests as not in use. Also, some tests within a testset may not be required in a particular test mode. For example, tests for POSIX.1-1996 functionality when running in POSIX90 mode. These are not failures and require no further work.

11.2.8 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 session IDs are inherited across a `fork()` when job control is not available.

These are generally listed on the manual pages under ‘‘Untestable Aspects’’.

11.2.9 Succeeded

This means the test has been executed correctly and to completion without any kind of problem.

X/Open System Verification Suite

Part 3: Appendices

VSXgen1.5 December 1999
VSTH 5.2.1 January 2000

A. ACTION POINT SUMMARY

A.1 PREPARATION

A.1.1 PREPARING YOUR SYSTEM

File Space Requirements

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

VSX User Accounts

1. Create a distinct group entry for `vsxg0` and (if required by one of the test packages) distinct group entries for `vsxg1` and `vsxg2`; usually in the file `/etc/group`.
2. If you do not already have TETware installed, create a directory you wish to designate as your **TET_ROOT** directory.
3. Create a distinct user entry for `vsx0` in group `vsxg0`, with home directory located under your **TET_ROOT**, and with a login shell; usually in the file `/etc/passwd`.
4. Make sure that the user `vsx0` has write permission in the **TET_ROOT** directory.
5. Add `$HOME/BIN` and `$HOME/../bin` to the command search path for user `vsx0`, and include it in the **PATH** environment variable set in the login script for user `vsx0`.
6. Ensure that any extensions enabled by environment variables that would cause non-compliant behaviour are disabled in the login script for user `vsx0`.
7. For some test packages, if the implementation supports supplementary groups, the user `vsx0` should have the maximum number of supplementary groups associated with it. These supplementary groups must exclude the groups `vsxg1` and `vsxg2`. The group ID values chosen must not exceed the value of **INT_MAX** for the system.
8. If required by one of the test packages, create a distinct user entry for `vsx1` in group `vsxg1`, in the password file. The home directory must differ from that of user `vsx0`.
9. If required by one of the test packages, create a distinct user entry for `vsx2` in group `vsxg2`, in the password file. The home directory must differ from that of user `vsx0`.

A.1.2 LOADING THE VSX DISTRIBUTION

Unpacking the Distribution Files

1. Log in to the test system as the user `vsx0`, who must be the owner of all the loaded files.
2. Ensure you are working in the `vsx0` home directory and that you have write permission in that directory.
3. Unpack the distribution files for VSXgen and the test packages you wish to use, using appropriate commands to decompress each file and extract all files from the resulting POSIX `cpio` or `tar` archive, e.g.:

```
zcat ../VSXgen1.5/source.cpio | pax -r -pp -x cpio
zcat ../VSX4/source.cpio | pax -r -pp -x cpio
zcat ../VSX5/source.cpio | pax -r -pp -x cpio
```

Checking the Contents

1. Change to the `vsx0` home directory (using `cd`) and list the directory. Check the expected subdirectories and the release identification files for each test package are there.

   If they are not, check that there were no read errors while the archives were being read and that there is space available on the file system.

2. Check that the release numbers given in the test-package specific Action Points for this chapter correspond with the release identification files.

A.1.3 REMOVING UNWANTED VSX DATA (OPTIONAL)

1. Remove any unwanted sections from the directories `tset` and `MAN/tset`.

A.1.4 VSTH PREPARATION

VSTH User Accounts

1. The VSTH test package requires configuration of the user `vsx0` and the groups `vsxg0.` User `vsx0` must configured to be a member of the group `vsxg0.` If your implementation supports supplementary group ids, you must also configure these supplementary groups for user vsx0.

Loading the VSTH Distribution

1. The VSTH test package is distributed as a compressed tar archive. If you have downloaded the gzip'd version of VSTH, execute the following commands as user vsx0 from your home directory:

   ```
   gzip -d vsth-<release_number>.tar.gz
   tar -xvf vsth-<release_number>.tar
   ```

2. If you have downloaded the compressed version of VSTH, execute the following commands as user vsx0 from your home directory:

   ```
   uncompress vsth-<release_number>.tar.Z
   tar -xvf vsth-<release_number>.tar
   ```

3. When you have finished unpacking VSTH, you should find the file VSTHrel*num* in the home directory of user vsx0, where *num* is the release number of VSTH. This file identifies the version of VSTH that you have unpacked and that you have unpacked all the contents of the release.

Removing Unwanted VSTH Data (Optional)

1. Header file tests are collected into the `PTHR.hdr` section.
2. Tests of pthread functions and related thread-safe functions are grouped under the `PTHR.os` section. Underneath this directory are further subdirectories corresponding to functional subsets of functions under test.
3. We recommend that you edit your scenario file to control compilation and execution of VSTH subsets.

A.2 CONFIGURING VSX

A.2.1 INSTALLATION DIRECTORY

1. Choose a directory for the installation of testset executables. The default is `TESTROOT` under the `vsx0` home directory.
2. If you are testing for conformance to FIPS 151, XPG4 or UNIX and one of the VSX test packages you are using requires that the installation directory must support the inheritance of parent directory group ID, and your system implements the inheritance of parent directory group ID by means of a directory mode setting that is inherited by subdirectories (for example by setting the `S_ISGID` bit on the directory), and there is an existing directory hierarchy under the testroot which does not support this feature, then you must remove it using

   ```
   rm -rf TESTROOT/tset
   ```

3. Set the environment variable `TET_EXECUTE` in the `vsx0` login script to the pathname of the testroot directory.

A.2.2 VSX CONFIGURATION SCRIPT

Running the Configuration Script

1. Read through the configuration script section and write down any information you will need to use which is different from the defaults.
2. Execute the shell script `config.sh` which is in the `BIN` directory. When you have included this directory in your **PATH**, you can execute the command from any location.
3. Answer the questions which the configuration script asks.

A.2.3 CHECKING THE PARAMETER FILES

Configuration Parameters File

1. Check the values in the configuration parameters file `SRC/vsxparams` and edit the *parameter–name* lines if necessary.

Configuration Header File

1. Check the `SRC/vsxconfig.h` file to ensure that the values are correct for your system.
2. Check that the values of all the varying defined constants have been changed from –1 to the values for your system.
3. Check that the values of the other defined constants are correct for your system.
4. Check that all dummy statements have been changed to valid ones.
5. If you re-run `config.sh` at a later time, it will overwrite `SRC/vsxconfig.h`, so make sure you copy it first.

A.2.4 TOP LEVEL MAKEFILE

1. Edit the `Makefile` in the `vsx0` home directory.
2. Configure the implementation-specific installation commands correctly for your system.
3. If you re-run `config.sh` at a later time, it will overwrite `Makefile`, so make sure you copy it first.

A.2.5 USER-SUPPLIED INTERFACE ROUTINES

1. Review the file `SRC/userintf.c` and identify if it needs to be modified.
2. Modify the file to meet your system's requirements.
3. If you re-run `config.sh` at a later time, it will overwrite `SRC/userintf.c` with the default version, so make sure you copy it first.

A.2.6 INSTALLING THE PSEUDO-LANGUAGES

Installing the Languages

1. Check the package-specific action points for this chapter to see whether the pseudo-languages are needed for the subsets and test mode you have selected. If they are not needed, skip the following action points.
2. If your system does not use the ASCII character set, modify the file `SRC/INC/ctrlcodes.h` so that it contains the correct control character encodings for your system. If your system uses an EBCDIC character set, copy the file `SRC/INC/ctrlebcdic.h` to `SRC/INC/ctrlcodes.h` first.
3. Check that the character encodings specified in the file `SRC/INC/pslcodes.h` are suitable for your system, and modify the file if they are not. If your system uses an EBCDIC character set, copy the file `SRC/INC/pslebcdic.h` to `SRC/INC/pslcodes.h` first.
4. Install the VSX pseudo-languages using the appropriate tools for your system. You may be able to make use of the files in `SUPPORT/psldefs` to do this.

A.2.7 WIDE CHARACTER CONFIGURATION FILE

1. If the wide character locales are needed for the subsets and test mode you have selected, update the file `SRC/wchars.cfg` with information about wide characters and multibyte characters supported on your system. If your system only supports single-byte characters, simply copy `SRC/wc_nosup.cfg` to `SRC/wchars.cfg`.
2. If you re-run `config.sh` at a later time, it will overwrite `SRC/wchars.cfg`, so make sure you copy it first.
3. Install additional locales if necessary to reflect the information in the file.
4. If you change `SRC/wchars.cfg` at a later stage there is no need to repeat the installation stage or to rebuild any testsets. Simply change directory to `SRC/common/wchars` and type `make`. This will update the binary file used by the tests from the new information in `wchars.cfg`.

A.2.8 CONFIGURING VSTH

VSX Configuration Script

1. VSTH uses the VSTH_PTHREAD_LIB and the VSTH_SYSLIBS parameters to identify libraries against which to link VSTH tests.

   `VSTH_PTHREAD_LIB` specifies the system library containing POSIX threads interfaces and must be set to -lpthread when performing conformance testing with VSTH.

   `VSTH_SYSLIBS` specifies any other system libraries needed to link VSTH and is often set to `-lrt`-laio when configuring VSTH.

Top Level Makefile

1. Implement the ENOSPC filesystem.

User-Supplied Interface Routines

1. Implement setprv() for the arguments `PRV_SETTHRSCHED`, `PRV_GETTHRSCHED` and `PRV_MOUNT`.
2. Implement vsth_setconcurrency().

A.3 INSTALLING VSX

A.3.1 INTRODUCTION

1. Obtain the necessary privileges for execution of the installation commands you have configured in the top level `Makefile` and execute `make` in the `vsx0` home directory. E.g.

   ```
   su root -c make
   ```

2. When `make` has completed, check the installation log in the `results` directory to ensure that no errors have occurred. If `make` encountered any errors, or there are errors in the log, you must correct them and re-run `make`.

A.3.2 INSTALLING VSTH

Build Common Software

1. Ensure that libvsth.a builds cleanly prior to any use of VSTH. Failure to do so will cause the subsequent build failure of all tests in the `tset/PTHR.os` area.
2. Ensure that libvsth_cp.a builds cleanly prior to use of VSTH as a conformance test. Failure to do so will cause the subsequent build failure of the `tset/PTHR.os/cancel/pthread_cancel/T.pthread_cancel` testset.
3. Ensure that libvsth_rt.a builds cleanly prior to use of VSTH as a conformance test. Failure to do so will cause the subsequent build failure of the `tset/PTHR.os/general/General/T.General` testset.

A.4 BUILDING VSX

1. Log in as the user `vsx0`.
2. Give the command

   ```
   tcc -b -s scen.bld
   ```

   with the other options you want to use. Use the command

   ```
   ../bin/tcc -b -s scen.bld
   ```

   from the `vsx0` home directory if `$HOME/../bin` is not in your **PATH**.

A.4.1 TERMINAL INTERFACE TESTING

Cable Wiring

1. Check the package-specific action points for this chapter to see whether the terminal loop-back is needed for the subsets and test mode you have selected. If it is not needed, skip the following action points.
2. If the system has at least two terminal ports, wire and connect two ports to provide a loop-back.

3. Make the two ports readable and writable by user `vsx0`.
4. There must not be any processes attached to these ports. (For example, you may have to change `/etc/inittab` or `/etc/ttys` to remove login processes at this point.)

A.4.2 BUILDING VSTH

Troubleshooting

1. None.

A.5 EXECUTING VSX

A.5.1 THE EXECUTION PARAMETERS FILE

Setting the Execution Parameters

1. Check the execution parameters file `tetexec.cfg` before you start the VSX execution stage and edit any values which are not correct for your system.

A.5.2 EXECUTING THE VSX TEST SUITE

1. Log in as the user `vsx0`.
2. Give the command

   ```
   tcc -e -s scen.exec
   ```

   with the other options you want to use. Use the command

   ```
   ../bin/tcc -e -s scen.exec
   ```

   from the home directory if `$HOME/../bin` is not in your **PATH**.

A.5.3 EXECUTING VSTH

Subset Specific Execution Parameters

1. Edit TESTROOT/tetexec.cfg and supply values for all VSTH execution parameters.

A.6 REPORTING

A.6.1 REPORTING PROGRAM OPTIONS

1. Log in as the user `vsx0`.
2. Check that the environment variable PATH is set correctly.
3. Change to the directory `results` if required.
4. Give the command `vrpt` with the options you want to use, on a journal file generated from the results of running the `tcc`.

A.6.2 POSIX CONFORMANCE REPORTING

1. Log in as the user `vsx0`.
2. Check that the environment variable PATH is set correctly.
3. Change to the directory `results` if required.
4. Give the command `prpt` with the options you want to use, on journal files generated from the results of running the `tcc`.

A.6.3 COMPARATIVE REPORTING

1. Log in as the user `vsx0`.
2. Check that the environment variable PATH is set correctly.
3. Change to the directory `results` if required.
4. Give the command `vrptm` with the options you want to use, on journal files generated from the results of running the `tcc`.

B. TESTS GIVING WARNINGS

B.1 WARNING CLASSIFICATIONS

The package-specific tables in the following sections show which tests can produce warnings and the classification of the warning. Unless stated otherwise, the classifictions given in each table are as follows.

Warnings in classification A are cases where many implementations differ from the functionality described in the relevant specification. In each of these cases, the functionality tested is not considered important enough to merit a failure, but in future releases of the verification suite these tests may be re-classified as failures.

Warnings in classification B are cases where different implementations produce different error numbers from those expected by the test. The specifications are not explicit about the precedence of errors and, therefore, these tests are only classified as warnings.

Warnings in classification C are cases where many implementations differ from the functionality described in the relevant specification but agree with a known future direction in which the specification is likely to alter.

Warnings in classification D are cases where the relevant specification describes the feature being tested as optional, and the implementation under test does not support it.

Warnings in classification E are cases where the behaviour expected by the test is not actually required by the relevant specification, but most implementations behave that way, and some applications may (incorrectly) rely on it. Such cases arise where a test has been downgraded to a warning rather than being removed, as it can still provide useful information to users.

C. PRIVILEGED PROGRAMS

The following pages list the programs in each test package that are assigned privileges other than `PRV_DEVICE` when the corresponding testsets are built.

C.1 LIST OF PRIVILEGED PROGRAMS IN VSTH

All paths are relative to TESTROOT/tset/PTHR.os .

```
cancel/pthread_cancel/T.pthread_cancel
thread/pthread_create/T.pthread_create
mutex/pthread_mutex_init/T.pthread_mutex_init
mutex_rt/pthread_mutex_getprioceiling/T.pthread_mutex_getprioceiling
mutex_rt/pthread_mutex_setprioceiling/T.pthread_mutex_setprioceiling
mutexattr_rt/pthread_mutexattr_getprioceiling/T.pthread_mutexattr_getprioceiling
mutexattr_rt/pthread_mutexattr_setprioceiling/T.pthread_mutexattr_setprioceiling
mutexattr_rt/pthread_mutexattr_getprotocol/T.pthread_mutexattr_getprotocol
mutexattr_rt/pthread_mutexattr_setprotocol/T.pthread_mutexattr_setprotocol
concurrency/pthread_setconcurrency/T.pthread_setconcurrency
concurrency/pthread_getconcurrency/T.pthread_getconcurrency
sched_rt/pthread_setschedparam/T.pthread_setschedparam
sched_rt/pthread_getschedparam/T.pthread_getschedparam
rwlock/pthread_rwlock_init/T.pthread_rwlock_init
ioprim/pwrite/pwrite_13
sched/pthread_attr_setschedparam/T.pthread_attr_setschedparam
sched/pthread_attr_setschedpolicy/T.pthread_attr_setschedpolicy
cond/pthread_cond_init/T.pthread_cond_init
condattr/pthread_condattr_setpshared/T.pthread_condattr_setpshared
mutexattr/pthread_mutexattr_init/T.pthread_mutexattr_init
mutexattr/pthread_mutexattr_setpshared/T.pthread_mutexattr_setpshared
```

D. SUPPORT SERVICES

D.1 FAULT REPORTING

The `SUPPORT` directory contains an error template file `error.template` for use in generating error reports. The file `error.example` gives an example of how to fill in the error template file.

Fault reports may be formatted by using the command

`nroff tmac.vsx` *reportfile*

Support is provided to VSX users by their supplier, who in turn may request support from The Open Group.

For this reason, support templates should be sent by electronic mail to your supplier, who will respond. In the instance where users have a multi-site licence, all support must be reported through the site which is supported by the supplier of the software.

Thus when reading the remainder of this appendix, users who have purchased support from a distributor should replace the contact electronic mail address and telephone numbers with those of their supplier.

The addresses given below are for problems relating to VSXgen files. For problems in test package files, please use the addresses given for each test package on the following pages.

Since files from VSXgen and the test packages coexist in a merged directory hierarchy, it may not be immediately clear which product a particular file belongs to. It may be helpful to keep a list of files extracted from each distribution file as you install each product (by using the `-v` option of *cpio*, for example).

Unformatted versions of VSXgen fault reports should be sent by electronic mail to the support mail box at:

`VSX_support@rdg.opengroup.org`

VSXgen support requests other than fault reports may be submitted either to the above address or direct to UniSoft at:

`vsx_support@unisoft.com`

Problems which can be attributed to TETware rather than to VSX (for example, incorrect behaviour of the `tcc` command) should be sent to the address specified in the TETware release notes and not to the addresses given in this appendix.

D.2 VSTH FAULT REPORTING

Fault reports relating to files in the VSTH test package should be sent using the VSTH support request form at:

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

or they may be submitted by email to the VSTH support mail box at:

```
vsth_support@opengroup.org
```

E. INTERPRETATIONS/WAIVERS AND CSQs

E.1 INTERPRETATIONS AND WAIVERS

Information about the procedures for applying for interpretations and waivers can be found on The Open Group's World Wide Web site, at the URL

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

A searchable database of existing interpretations and waivers is available at the URL

```
http://www.opengroup.org/interpretations/database
```

E.2 ELECTRONIC CONFORMANCE STATEMENTS

The Open Group has introduced electronic conformance statements which can be completed online. Conformance Statement Questionnaires, for use when making a formal application to The Open Group for product registration, are available on the world wide web at

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

If you have any questions on setting up an account for completing your conformance statement questionnaire, please contact the conformance administrator by electronic mail:

```
conformance@opengroup.org
```

E.3 PURPOSE OF THE CONFORMANCE STATEMENTS

The Conformance Statements provide details of the manner in which a specific implementation meets the definitions contained in the relevant specification, and define the environment in which conformant behaviour, and any test results, may be reproduced.

In addition, the Conformance Statements provide details of waivers granted by The Open Group in respect to product errors of a minor nature, which have a negligible effect on application portability. Such waivers, referred to as *temporary waivers*, are granted for a limited period, during which the implementor undertakes to effect a correction.

The majority of the definitions in the specifications are mandatory and all branded components will conform to these definitions. There are however a number of optional features which an implementor can choose to provide. The conformance statements identify whether these optional features have been provided or not.

The conformance statements characterise the variances between the different conforming implementations. The conformance statements are useful documents for any person who requires knowledge of the details of a conforming implementation.

Minor implementation choices that should not affect a portable application are not included in the conformance statements. A well behaved implementation should not rely on behaviour that is stated to be undefined or unspecified in the specification. Only cases where the behaviour is explicitly stated to be optional are considered in the conformance statements.

E.4 RELATIONSHIP TO CONFORMANCE TESTING

Conformance Testing is one element in the determination of a conformant system. There are a number of reasons why conformance testing is not, by itself, adequate as a complete measure of the conformance of an implementation. As the conformance tests are improved and enhanced to cover additional areas within the specifications, the degree of confidence associated with conformance testing will increase.

The conformance statements act as a specification of the implementation and are the basis of the statement of conformance provided by the implementor. The conformance tests act as a monitor of the statement of conformance, providing evidence of any deviant behaviour in those areas covered by the conformance tests.

The conformance statements should align with the evidence provided by the execution of the conformance tests. For example, if a conformance statement states that the implementation provides a certain optional feature, then the conformance tests should have verified that the implementation provides this facility according to the relevant specification.

X/Open System Verification Suite

Part 4: Manual Pages

VSXgen1.5 December 1999

NAME

prpt - POSIX summary report generator

USAGE

prpt [**-H**] *[-***L***len] [-***W***wid]* jnlfile ...

DESCRIPTION

Prpt produces a POSIX conformance report from one or more test journal files. If the test suite was not installed and run in one of the POSIX modes, the extra test results in the journal file(s) are ignored by **prpt**. Reports are produced on the standard output.

Each report is prefaced by several cover pages, one for each test journal file being processed, listing information from the file as follows: file name, validation test name, test date, test agency and system, test operator and all test parameters.

The cover pages are followed by tables of test results, sorted by POSIX chapter then interface name. Each line contains the interface name, assertion number and result code (PASS, FAIL, UNRESOLVED, UNSUPPORTED, UNTESTED or NOTINUSE).

PARAMETERS

Command Line

-H Disable page headers and footers. *Prpt* will normally print page headers and footers whose placement and size depend on the page size flags given below. They contain the page number, report date and report type. The report date is determined from the host system at the time of prpt invocation. Headers and footers include blank lines between each header and footer and page text.

Prpt produces headers and footers by default if no -H flag is given.

-L*len* Page length is *len* lines - used to place headers and footers properly on the output pages. Defaults to 66 lines if no -L flag given.

-W*wid*

Page width is *wid* columns - used to generate headers and footers and to wrap long lines. Defaults to 80 columns if no -W flag given.

Environment

VSXBIN

specifies the directory where prpt executables and scripts reside. If **VSXBIN** is not set this directory is assumed to be *$HOME/BIN*.

RETURNS

0 Report terminated successfully

1 Unknown option argument or command line usage error

2 Unreadable file or other unrecoverable error during report generation

DIAGNOSTICS

"cannot read input file <filename>"
An input file given on the command line could not be opened.

EXAMPLES

prpt journal01
Generate a report from journal file "journal01, using the default page size with page headers and footers, and put the report onto the standard output.

prpt -H -W132 journal*
Generate a report from all journal files in the current directory, using the default page length, a page width of 132 columns, and with no page headers or footers.

SEE ALSO

vrpt(vprog)
vrptm(vprog)

AUTHORS

Geoff Clare, UniSoft Ltd.
Stuart Boutell, UniSoft Ltd.
J. A. Nave, UniSoft Ltd.

RELEASE

VSXgen 1.5

NAME

tcc – TETware test case controller

SYNOPSIS

tcc –{**bec**} [*options*] [*test-suite* [*scenario*]]

tcc –{**bec**} –**m** *codelist* [*options*] *old-journal-file* [*test-suite* [*scenario*]]

tcc –{**bec**} –**r** *codelist* [*options*] *old-journal-file* [*test-suite* [*scenario*]]

DESCRIPTION

tcc is the TETware test case controller. It provides support for the building, execution and clean-up of test scenarios.

When TETware-Lite is built, scenarios may only contain test cases which are to be executed on the local system and **tcc** performs all the actions required to process such test cases itself. When Distributed TETware is built, scenarios can contain local, remote and distributed test cases. The distributed version of **tcc** does not perform the actions required to process test cases itself but instead sends requests to the test case controller daemon **tccd** which runs on the local system and also on each participating remote system (see the **tccd**(1) manual page for details).

Apart from the scenario directives which relate to the processing of remote and distributed test cases, the user interface to **tcc** is the same irrespective of whether TETware-Lite or Distributed TETware is being used.

tcc has three modes of operation, namely **build**, **execute** and **clean**, which may be invoked singly or in any combination. These modes are specified by the –**b**, –**e** and –**c** command-line options, at least one of which must appear. All of the other options modify the behaviour of **tcc** in one or more of these operational modes. Each mode (with optionally modified behaviour) is applied to the test cases and invocable components selected for processing.

By default, **tcc** builds, executes or cleans test cases in the named *scenario* contained in the scenario file **tet_scen**, which is located in the test suite root directory for *test-suite* (see DIRECTORIES below). If no *scenario* is specified, the default scenario named **all** is used. If no *test-suite* is specified, **tcc** attempts to deduce a default test suite name using the following rules:

1. If the **TET_SUITE_ROOT** environment variable is set and the current directory lies under the directory hierarchy specified by this variable, then the test suite is the component of the current directory's path name which lies immediately below **$TET_SUITE_ROOT**. For example, if **$TET_SUITE_ROOT** is **/usr/tet3** and the current directory is **/usr/tet3/suite1/results**, then the name of the default test suite is **suite1**.

2. If the **TET_SUITE_ROOT** environment variable is not set and the current directory lies under the directory hierarchy specified by the **TET_ROOT** environment variable, then the test suite is the component of the current directory's path name which lies immediately below **$TET_ROOT**.

3. If the current directory lies outside of the directory hierarchy specified by the **TET_SUITE_ROOT** environment variable (if set) or the **TET_ROOT** environment variable (if **TET_SUITE_ROOT** is not set), then no default test suite name can be deduced.

DIRECTORIES

By default, **tcc** interprets test case names relative to the **test suite root** directory. The location of this directory is determined as follows on the local system:

1. If the **TET_SUITE_ROOT** environment variable is set, the **test suite root** directory is determined by the test suite name, relative to **$TET_SUITE_ROOT**.

2. If the **TET_SUITE_ROOT** environment variable is not set, the **test suite root** directory is determined by the test suite name, relative to **$TET_ROOT**.

3. If the **TET_RUN** environment variable is set, then the directory subtree below the **test suite root** (determined as described above) is copied to the location below **$TET_RUN** and this location becomes the new **test suite root** directory.

However, an alternate execution directory on the master system may be specified by the **TET_EXECUTE** environment variable or by a command-line option (see OPTIONS below). If an alternate execution directory is specified, **tcc** interprets test case names relative to this directory when operating in execute mode.

By default, **tcc** creates a directory called **tet_tmp_dir** below the test suite root directory. However, a different temporary directory name on the local system may be specified by the **TET_TMP_DIR** environment variable. Each invocation of **tcc** creates a unique subdirectory below the temporary directory on startup and removes it and its contents on normal completion.

CONFIGURATION FILES

During execution, **tcc** reads configuration variables from certain configuration files on both the local and the remote systems (if any). By default, the name of the build mode configuration file is **tetbuild.cfg**, that of the execute mode configuration file is **tetexec.cfg** and that of the clean mode configuration file is **tetclean.cfg**. The build and clean mode configuration files reside in the test suite root directory on each system. The execute mode configuration file resides in the alternate execution directory if one has been specified, otherwise in the test suite root directory.

The Distributed version of **tcc** reads distributed configuration variables are read from the file named **tetdist.cfg** in the test suite root directory on the local system. This file must at least contain definitions for the **tet root** and test suite root directories for any remote systems that are specified in the scenario being processed.

JOURNAL FILE

By default, **tcc** creates a sequentially numbered directory below the **results** directory in the test suite root directory for the named *test-suite* on the local system, and places the journal file and saved intermediate result files there. On startup, **tcc** writes the name of the journal file being used to the standard output.

RESULT CODES

tcc uses a table of result codes to interpret the results generated by API-conforming test cases. A default table containing standard codes is built in to **tcc**. It is possible to specify additional codes in user-supplied result codes files located below the **tet root** and test suite root directories on the local system. These files are optional but, if they exist, the codes specified in them are added to the table of standard codes. The default name for each of these files is **tet_code** but this name can be changed by means of the TET_RESCODES_FILE configuration variable.

OPTIONS

The following *options* alter the default behaviour described above:

–I Enable interactive journal trace. Journal lines which indicate the start and end of processing of each test case in each of the chosen modes of operation are written to the standard error as well as being written to the journal file.

–a *directory*
Use *directory* as the alternate execution directory instead of the one specified by the **TET_EXECUTE** environment variable (if any).

–f *file* Use *file* as the clean mode configuration file instead of the default.

–g *file*
Use *file* as the build mode configuration file instead of the default.

–i *directory*
Place the default journal file and saved intermediate results files in *directory* instead of in the default location.

–j *file* Use *file* as the journal file instead of the default.

–l *scenario-line*
Process *scenario-line* as if it appeared in a scenario file below a scenario named **all**. More than one **–l** option may be specified; the *scenario-lines* are processed in the order in which they appear on the command line. *scenario-line* must be presented as a single argument so it must be quoted if it contains embedded spaces. If a scenario file is specified by a **–s** option, any *scenario-lines* are processed before that scenario file is read. If no **–s** option is specified, the default scenario file **tet_scen** is not read when **–l** is used.

–n *string*
Do not process test case names that contain *string*. More than one **–n** option may appear.

–p Enable progress reporting. As each build, execute or clean operation is started, a line indicating the time, mode and scenario line being processed is printed on the standard output.

–s *file*
Use *file* as the scenario file instead of the default.

–t *timeout*
Terminate the build, execute or clean of an individual test case if processing would continue for more than *timeout* seconds.

–v *variable=value*
The specified configuration *variable* is set to *value*, overriding any assignment in the configuration file for the current mode. It is probably best to surround *value* with single quotes if it contains characters which have special meaning to the Shell. More than one **–v** option may appear.

–x *file*
Use *file* as the execute mode configuration file instead of the default.

−y *string*

Only process test case names that contain *string*. More than one **−y** option may appear. The **−n** option has higher precedence than the **−y** option; thus, a test case is not processed if its name is matched by *strings* specified with both the **−n** and the **−y** options.

RERUN AND RESUME OPTIONS

The following options are mutually exclusive:

−m *code-list*

Causes **tcc** to resume the previous run of the specified *scenario* in the named *test-suite* whose results are in *old-journal-file*. *code-list* specifies the point in the previous run from which processing is to be resumed and may consist of a comma-separated list of result codes, or of one or more of the letters **b**, **e** and **c** to specify failures in particular processing modes. If *code-list* consists of result codes, then processing resumes at the first invocable component whose result in the previous run matched one of those in the list. If *code-list* specifies processing modes, then processing resumes at the first test case which failed to build or clean or the first invocable component which, when executed, did not report PASS in the previous run.

For example:

tcc −b −m b

Resume building from the first test case that failed to build.

tcc −e −m FAIL,UNRESOLVED

Resume execution from the first invocable component that reported FAIL or UNRESOLVED.

tcc −bec −m b,e

Resume building, execution and cleaning from the first test case which failed to build or from the first invocable component that did not report PASS.

−r *code-list*

Causes **tcc** to re-run individual test cases and invocable components from the specified *scenario* in the named *test-suite* whose results are in *old-journal-file*. *code-list* specifies the elements that are to be re-run and may consist of a comma-separated list of result codes, or of one or more of the letters **b**, **e** and **c** to specify failures in particular processing modes. If *code-list* consists of result codes, then test cases and invocable components are re-run if the corresponding result in the previous run matched one of the result codes in the list. If *code-list* specifies processing modes, then a test case is re-run if it failed to build or clean and an invocable component is re-run if it did not report PASS when it was executed in the previous run.

For example:

tcc −b −r b

Re-build test cases that previously failed to build.

tcc −e −r FAIL,UNRESOLVED

Re-execute all invocable components that previously reported FAIL or UNRESOLVED.

tcc −bec −r b,e

Re-build, execute and clean all test cases that previously failed to build or execute, and all invocable components that did not previously report PASS when executed.

FILES

test-suite-root/**tet_scen**
Default scenario file. In Distributed TETware, only required on the local system.

test-suite-root/**tetbuild.cfg**
Default build mode configuration file.

alt-exec-dir/**tetexec.cfg**
Optional default execute mode configuration file when an alternate execution directory has been specified.

test-suite-root/**tetexec.cfg**
Default execute mode configuration file when *alt-exec-dir*/**tetexec.cfg** does not exist or an alternate execution directory has not been specified.

test-suite-root/**tetclean.cfg**
Default clean mode configuration file.

test-suite-root/**tetdist.cfg**
The distributed configuration file. Not used by TETware-Lite. In Distributed TETware, only required on the local system.

$TET_ROOT/tet_code
test-suite-root/**tet_code**
Default result code files. In Distributed TETware, only accessed on the local system.

test-suite-root/**tet_tmp_dir**
Default temporary directory hierarchy.

test-suite-root/**results/***nnnn*{**bec**}
Default results and saved files directory.

results-dir/**REMOTE***nnn*
In Distributed TETware on the local system, the saved files directory for system *nnn*.

results-dir/**journal**
Default journal file. In Distributed TETware, only created on the local system.

NAME

vrpt - validation test report generator

USAGE

vrpt *[-***l***level] [-***r***coverage] [-***f***file]* [**-v**] [**-H**] [**-P**] [**-p**]
*[-***L***len] [-***W***wid] [-***t***lines]* [*jnlfile*...]

DESCRIPTION

Vrpt generates a report from journal files specified by the *jnlfile* argument. Reports are generated on the standard output.

Reports can be generated at one of three **levels** - the **section**, **area**, and **testset** levels - and covering a specified range of sections, areas or testsets within these levels.

At the **section** level, for each section specified by the **coverage** parameter, a section report is produced listing section start and end times, and a section summary listing the number of areas and testsets run and the number of test/make results in each category. Test result categories are: Succeeded, Failed, Warning, FIP (Further Information Provided), Unresolved, Uninitiated, Unsupported, Untested and Not In Use. Make result categories are: Succeeded, Failed and Unsupported.

At the **area** level, for each area specified by the **coverage** parameter, an area report is produced giving area start and end time, a table showing the number of results in each category (see list above) for each testset run, a list of all unsuccessful tests under result category headings, and an area summary listing the number of testsets run and the number of results in each category. A section summary is produced after the area reports in each section.

At the **testset** level, for each testset specified by the **coverage** parameter, a report is produced listing testset start and end time, detailed testset results, and a summary listing the number of results in each category (see list above). The detailed results list each unsuccessful test/make with its result category and any supplementary information produced by the test/make stage. In verbose mode (-v flag given) all successful and "Not In Use" tests/makes are also listed. Area and section summaries are produced after the testset reports in each area/section.

Each section, area, or testset level individual report starts with the section/area/testset identifier as passed through from the test/make stage.

Each validation test report run is normally prefaced by cover pages giving a contents list, an operational summary and a conformance summary. Details are described under the "-P" flag description below.

Any input that *vrpt* does not understand will be ignored, with a warning being issued for the first of a sequence of lines not understood. Processing will attempt to continue normally from the first understandable line.

PARAMETERS

Command Line

-l*level* generate report at *level*. *Level* can be one of "**sect**", "**area**", or "**tset**", for section, area, and testset levels respectively.
level defaults to **tset** if no -l parameter given.

-r*coverage*
report only on the specified range of tcc output within the level specified above. Note that the tables in the conformance summary cover page always give the complete results for the journal files being processed - only the detailed reports are affected when a reduced coverage is specified.

Coverage can be specified as follows:
1) as a list in the format "name1 name2 ... name*n*", where a report will be generated for each section/area/testset (depending on report level) whose name appears in the list;
2) as a range, in the format "name1:name2", where reporting will start with the section/area/testset (depending on report level) named name1 and will continue until the report for section/area/testset name2 is completed; note that ordering of sections/areas/testsets within each journal file depends on the scenario files, and may differ between runs; or,
3) a combination of the above, in the format "name1:name2 name3 name4 name5:name6", etc - with the combination of the above meanings.

It is *not* an error if specified names do not actually appear in the journal files.

If range names are specified to a greater level than the level as given in the -l option, then the extra levels are ignored. E.g. **-l sect -r section1/area1** will be processed the same as **-l sect -r section1**.

Vrpt defaults to reporting on every section/area/testset in the journal files if no -r list is specified.

-f*file* take the coverage specifier list from *file*; this file should contain a list of section/area/testset identifiers or names one per line.

The **-f** and the **-r** parameters can be used together - the two lists are merged, and the resulting list used, ignoring repetitions.

-v Verbose mode - list names of successful and "Not In Use" tests/makes in testset level reports. Without the -v flag, detailed output at this level is produced only for unsuccessful tests/makes.

-H Disable page headers and footers. *Vrpt* will normally print page headers and footers whose placement and size depend on the page size flags given below. They contain the report level, system and agency names, test/make date, and page number.

Vrpt produces headers and footers by default if no -H flag is given.

-P Do not print the cover pages normally produced with reports, or the parameter list at the end of all reports. Test cover pages consist of a banner page; a contents page; an operational summary listing test date, test agency and operator, report date, report level and coverage, and the journal files reported on; and, a conformance summary showing tables of the total number of tests in each result category for each section, over the whole of the journal files being processed (not just the coverage specified with -r).

Report date is determined from the host system at the time of vrpt invocation. Report level and coverage are as given on the vrpt command line. All other items are determined from the journal files.

The -P parameter implies -p, so that progressive report output is generated.

Vrpt defaults to producing the cover pages and parameter list if the -P flag is not given.

-p Disable "Page n of N" page numbering style. *Vrpt* will normally report the page numbers in the style "Page 4 of 40". However, this means that no output is generated all the pages have been prepared. **-p** allows the user to disable this feature, and thus allowing progressive report output.

Vrpt will produce footers with the "Page n of N" page numbering, and generates a progress message every 25 pages on stderr, if the -p flag is not given.

-L*len* Page length is *len* lines - used to place headers and footers properly on the output pages. Defaults to 66 lines if no -L flag given.

-W*wid*

Page width is *wid* columns - used to generate headers and footers and to wrap long lines in the journal file. Defaults to 80 columns if no -W flag given.

-t*lines* Truncate test failure information after the specified number of lines. Some tests can produce hundreds of lines of failure information. This option may be used to reduce the size of a full report, with the complete test information for tests of interest subsequently being obtained by using the **-r** option. The default is no truncation.

Environment

VSXBIN

specifies the directory where vrpt executables and scripts reside. If **VSXBIN** is not set this directory is assumed to be *$HOME/BIN*.

RETURNS

0 Report terminated successfully

1 Unknown option argument

2 Unrecoverable error during report generation

DIAGNOSTICS

"warning: input does not start with control sequence"

- input file was probably not a correctly formed test/make journal file (i.e. file did not start with the special "start report" sequence); this is not fatal, but may produce some strange output.

"warning - line XXX: line ignored"

vrpt ignores lines it does not expect to see at that point, or that appear to be malformed. Not fatal, and only produced for the first such error in each sequence of malformed lines in the input file.

EXAMPLES

vrpt foo

Generate a validation report from the journal file named foo, using default levels and coverage (testset/all), producing page headers and footers and a parameter list, and put the report onto standard output.

vrpt -H -P -larea -r'area1 area3:area7' journal[12]

Generate a validation report from the vtest journal files "journal1" and "journal2", at the area level, for the area "area1" and every area from "area3" to area7" inclusive; don't produce any page headers or footers or cover pages.

SEE ALSO

prpt(vprog)
vrptm(vprog)

AUTHORS

Hamish Reid, UniSoft Ltd.
Stuart Boutell, UniSoft Ltd.
J. A. Nave, UniSoft Ltd.

RELEASE

VSXgen 1.5

NAME

vrptm - multiple test run comparison report generator

USAGE

vrptm [**-H**] *[***-L***len] [***-W***wid]* file1 file2 ...

DESCRIPTION

Vrptm produces a report comparing the results of two or more VSX validation test runs. It takes as input the test journal files from the runs to be compared. The file names given on the command line are used in the report to identify the results from the corresponding runs. Reports are produced on the standard output.

Each report is prefaced by several cover pages, one for each test journal file being processed, listing information from the file as follows: file name, validation test name, test date, test agency and system, test operator and all test parameters.

The cover pages are followed by tables of test results, one table per testset, showing the results for each test across all the input files. The tables contain one word entries giving the test result category (Succeeded, Failed, Warning, FIP (Further Information Provided), Unresolved, Uninitiated, Unsupported, Untested or Not In Use). A '-' character in the table indicates that no result was found for that test in the corresponding file. Any additional test information in the journal file is not reproduced.

PARAMETERS

Command Line

-H Disable page headers and footers. *Vrptm* will normally print page headers and footers whose placement and size depend on the page size flags given below. They contain the page number, report date and report type. The report date is determined from the host system at the time of vrptm invocation. Headers and footers include blank lines between each header and footer and page text.

Vrptm produces headers and footers by default if no -H flag is given.

-L*len* Page length is *len* lines - used to place headers and footers properly on the output pages. Defaults to 66 lines if no -L flag given.

-W*wid*

Page width is *wid* columns - used to generate headers and footers and to wrap long lines in the journal file. Defaults to 80 columns if no -W flag given.

Environment

VSXBIN

specifies the directory where vrptm executables and scripts reside. If **VSXBIN** is not set this directory is assumed to be *$HOME/BIN*.

RETURNS

0 Report terminated successfully

1 Unknown option argument or command line usage error

2 Unreadable file or other unrecoverable error during report generation

DIAGNOSTICS

"cannot read input file <filename>"
An input file given on the command line could not be opened.

"insufficient page width for number of files - using <wid>"
The page width specified with the -W option was less than the minimum required for the number of input files being processed. The program will produce the report using <wid> instead of the requested width.

EXAMPLES

vrptm journal1 journal[45]
Generate a report comparing test results from journal files journal1, journal4 and journal5, using the default page size with page headers and footers, and put the report onto the standard output.

vrptm -H -W132 jo*
Generate a report comparing test results from all "journal" files in the current directory, using the default page length, a page width of 132 columns, and with no page headers or footers.

SEE ALSO

vrpt(vprog)
prpt(vprog)

AUTHORS

Geoff Clare, UniSoft Ltd.
Stuart Boutell, UniSoft Ltd.

RELEASE

VSXgen 1.5

CONTENTS