Test Environment Toolkit

TETware Installation Guide
for UNIX Operating Systems
Revision 1.6
TET3-IGU-1.6

Released: July 2000

The Open Group

The information contained within this document is subject to change without notice.

This document is produced by UniSoft Ltd. at:

150 Minories
LONDON
EC3N 1LS
United Kingdom

CONTENTS

1. Introduction

1.1 Preface

There is one Installation Guide for each type of system on which TETware is supported. This document describes how to install both TETware-Lite and Distributed TETware on a UNIX system.

1.2 Audience

This document is intended to be read by software engineers and/or systems administrators who will install TETware on their computer systems. A knowledge of system administration is assumed when TETware installation and configuration instructions are presented. In addition, a knowledge of network administration is assumed when Distributed TETware is to be installed.

You may find it helpful to have a basic understanding of TETware architecture and its components before attempting to install TETware on your computer systems. This information is presented in the chapter entitled ‘‘TETware overview’’ in the TETware User Guide.

Test suite authors should refer to the TETware Programmers Guide for information about how to use the TETware Application Program Interfaces.

1.3 Conventions used in this guide

The following typographic conventions are used throughout this guide:

- `Courier font` is used for function and program names, literals and file names. Examples and computer-generated output are also presented in this font.
- The names of variables are presented in *italic font*. You should substitute the variable’s value when typing a command that contains a word in this font.
- **Bold font** is used for headings and for emphasis.

Long lines in some examples and computer-generated output have been folded at a \ character for formatting purposes. If you type such an example, you should type it in all on one line and omit the \ character.

1.4 Related documents

Refer to the following documents for additional information about TETware and its predecessors:

- *Test Environment Toolkit: TETware Programmers Guide*
- *Test Environment Toolkit: TETware User Guide*
- *Test Environment Toolkit: TETware Knowledge Base*

In addition, the TETware Release Notes contain important information about how to install and use TETware. You should read the release notes thoroughly before attempting to install and use each new release of TETware.

1.5 Problem reporting

If you have subscribed to TETware support and you encounter a problem while building or installing TETware, you can send a support request by electronic mail to the address given in the TETware Release Notes. Please follow the instructions contained in the release notes about how to submit such a request; in particular, please be sure to include all the information asked for by these instructions when submitting the request.

2. The TETware distribution

2.1 Introduction

This chapter describes what is in the TETware distribution and how to load it. You must load TETware on all the machines that you want to use to run local, remote or distributed tests.

2.2 The TET root directory

You must choose a directory on each machine in which to load the TETware distribution.

Note: Some of the file names in TETware are the same as those in other TET distributions. You should **not** load TETware in a directory that is occupied by an existing TET, DTET or ETET distribution, otherwise some files will be overwritten.

Each directory where TETware is loaded is known as the **TET root directory** and will be described here as *tet-root*. When you specify the path name of your TET root directory, replace *tet-root* with your equivalent path name.

2.3 Loading the distribution

You should load the TETware distribution on to every machine on which you want to run local, remote or distributed tests.

After you load the TETware distribution you will have a number of new files and directories on your machine. Some directories are present in both the source and the binary distributions, while others are present only in the source distribution.

The following list describes the directories which make up the distribution. Directories which are not present in the binary distribution are marked with a †. Directories which are empty in the source distribution but are populated when you build TETware are marked with a ‡. Directories for add-on products that may not be present in all distributions are marked with a §.

tet-root`/bin`	Directory containing TETware executables.‡
tet-root`/demo`	Test suite root directory for the example distributed test suite.[1]
tet-root`/doc/java`	Directory that will contain `javadoc` output files when the Java API is installed.§
tet-root`/doc/ksh`	Not used in TETware.[2]
tet-root`/doc/tet3`	Directory subtree containing TETware documentation files.
tet-root`/doc/xpg3sh`	Not used in TETware.[3]

1. This directory is empty in both the source and the binary distribution. Instructions for running the distributed demonstration are presented in the TETware User Guide.
2. The documentation below *tet-root*/doc/tet3 includes references to the Korn Shell API where appropriate. Thus, no separate documentation files appear in this directory.
3. The documentation below *tet-root*/doc/tet3 includes references to the Shell API where appropriate. Thus, no separate documentation files appear in this directory.

tet-root/inc/tet3	Directory containing public header files for use by test suites and report writers.
tet-root/jdemo	Test suite root directory for the example Java test suite.§
tet-root/lib/java	Directory containing library files for use when executing java test cases.‡§
tet-root/lib/ksh	Directory containing library files for use when executing Korn Shell test cases.‡
tet-root/lib/perl	Directory containing library files for use when executing Perl test cases.‡
tet-root/lib/tet3	Directory containing library files for use when building TETware tools and C language test cases.‡
tet-root/lib/xpg3sh	Directory containing library files for use when executing Shell test cases.‡
tet-root/src/defines	Directory subtree containing example machine-specific makefile definition files.†
tet-root/src/helpers	Directory containing shell scripts that are useful when preparing to build TETware.†
tet-root/src/java	Source directory for the Java API.†§
tet-root/src/ksh	Source directory for the Korn Shell API.†
tet-root/src/perl	Source directory for the Perl API.†
tet-root/src/scripts	Source directory for shell scripts.†
tet-root/src/tet3/apilib	Directory containing source files for the C API.†
tet-root/src/tet3/apishlib	Directory used when building the shared version of the C API library.†
tet-root/src/tet3/apithr	Directory containing source files for the Thread-safe C API.†
tet-root/src/tet3/apithrshlib	Directory used when building the shared version of the Thread-safe C API library.†
tet-root/src/tet3/bin	Directory containing scripts that are used when building TETware.†
tet-root/src/tet3/demo/master	Directory containing source files for the master part of a simple demonstration distributed test suite.
tet-root/src/tet3/demo/slave	Directory containing source files for the slave part of a simple demonstration distributed test suite.

tet-root/src/tet3/dtet2lib
: Directory containing source files for low-level library functions.†

tet-root/src/tet3/inc
: Directory containing private header files for the C sources.†

tet-root/src/tet3/inetlib
: Directory containing source files for INET transport-specific library functions.†

tet-root/src/tet3/japilib
: Directory containing source files for the Java API.†§

tet-root/src/tet3/jtools
: Directory containing source files for the tools that are used when processing Java test cases.†§

tet-root/src/tet3/llib
: Directory containing lint library source files.

tet-root/src/tet3/servlib
: Directory containing source files for the generic client/server code and server access library functions.†

tet-root/src/tet3/syncd
: Directory containing source files for the Synchronisation daemon `tetsyncd`.†

tet-root/src/tet3/tcc
: Directory containing source files for the Test Case Controller `tcc`.†

tet-root/src/tet3/tccd
: Directory containing source files for the Test Case Controller daemon `tccd`.†

tet-root/src/tet3/tccdsrv
: Directory containing souce files for the Windows NT service version of the `tccd` bootstrap program.†

tet-root/src/tet3/tcclib
: Directory containing source files for `tcc` action functions.†

tet-root/src/tet3/tcm
: Directory containing source files for the C and C++ Test Case Manager components.†

tet-root/src/tet3/tcmshlib
: Directory used when building versions of the C and C++ Test Case Managers that are used with the shared C API library.†

tet-root/src/tet3/tcmthr
: Directory used when building the Thread-safe C and C++ Test Case Managers.†

tet-root/src/tet3/tcmthrshlib
: Directory used when building versions of the C and C++ Test Case Managers that are used with the shared Thread-safe C API library.†

tet-root`/src/tet3/xresd`	Directory containing source files for the Execution Results daemon `tetxresd`.†
tet-root`/src/tet3/xtilib`	Directory containing source files for XTI transport-specific library functions.†
tet-root`/src/xpg3sh`	Source directory for the Shell API.†

2.4 What to do next

If you have loaded a TETware source distribution, you should build and configure TETware on your system by working through the instructions presented in the rest of this guide.

If you have loaded Distributed TETware from a binary distribution, you should configure TETware on your system. Details of how to do this are presented in the chapter entitled ‘‘Configuring your system to run Distributed TETware’’ later in this guide.

If you have loaded TETware-Lite from a binary distribution, no further configuration is required and it is now ready to use.

3. Building TETware

3.1 Introduction

This chapter describes how to build TETware from a source distribution. If you are installing TETware from a binary distribution you should skip to the next chapter.

You should build TETware on every machine on which you want to run local, remote or distributed tests.

3.2 System requirements

Prior to building TETware on each machine, you should check that the following requirements are met as TETware will not build without them.

When TETware is built on a UNIX system, it is necessary for the operating system be compliant with **all** of the following:

- POSIX.1[4] with either standard C or common C support.
- XPG3, Vol. 1[5] (for the Shell and other commands).

These are the minimum requirements for building TETware-Lite.

In addition, each machine should provide **one** of the following types of network communication facility when Distributed TETware is to be built:

- The set of network-related system calls and/or library routines, sometimes known as **Berkeley sockets**,[6] that allow access to TCP/IP network facilities.
- X/Open Transport Interface (XTI) library routines.[7]

In order to make use of certain TETware APIs, additional software is required as follows:

- The Perl utility (Version 5.0 or later) is required in order to build and use the Perl API.
- A C++ compiler is required in order to build and use the C++ API.
- An implementation of the Korn Shell is required in order to use the Korn Shell (`ksh`) API.
- Support for either Unix International threads or POSIX threads is required in order to use the Thread-safe versions of the C and C++ APIs.
- The Java Development Kit (JDK) version 1.1 is required in order to build and use the Java API.[8]

---

4. IEEE Std 1003.1-1990, *Portable Operating System Interface for Computer Environments*.
5. *X/Open Portability Guide Issue 3, Volume 1: XSI Commands and Utilities*.
6. As implemented in BSD4.3 and Sun-OS.
7. Defined in *X/Open CAE specification: X/Open Transport Interface (XTI) version 2*.
   The transport provider must support automatic address generation as defined in the description of the `t_bind()` subroutine.
8. This API is only supported on certain UNIX implementations. Please refer to the TETware Release Notes for details.

In order to build shared versions of the C API libraries, support for dynamic linking is required. The shared library schemes supported by TETware are described in the section entitled ‘‘Support for shared API libraries’’ later in this chapter.

3.3 Before you build TETware

3.3.1 Introduction

The following subsections describe decisions that you must make before you build TETware.

3.3.2 Which version of TETware do you want to build?

You must first decide whether you want to build TETware-Lite or Distributed TETware. A description of the differences between these two options is presented in the chapter entitled ‘‘TETware overview’’ in the TETware User Guide.

If you decide to build TETware-Lite the next section is not applicable and you should skip to the section entitled ‘‘Preparing to build TETware’’.

3.3.3 Distributed TETware options

3.3.3.1 Which network interface do you want to use?

As indicated previously, Distributed TETware uses network facilities to communicate between different systems and processes.

On UNIX systems, Distributed TETware can be compiled to use one of the following interfaces:

- BSD-style network calls (i.e., `socket()`, `connect()`, etc.).
- The X/Open Transport Interface (XTI).

You must decide which of these you want to use before you build TETware. The choice that you make will determine which library of TETware transport-specific routines are used when Distributed TETware is built.

In addition, the choice that you make will affect the ease with which Distributed TETware can be installed and used on any particular machine. There are advantages and disadvantages associated with the use of each network access method, as follows:

- Advantages of using BSD-style networking are:
  - — BSD networking performs well and is comparatively easy to set up
  - — BSD networking is used as the basis of network services on almost all networked UNIX systems
- whereas disadvantages are:
  - — since the API has not traditionally been specified by standards, some systems vary in their network implementation; thus you may find it necessary to change some of the transport-specific part of the TETware code before you can compile it on your machine.
- Advantages of using XTI networking are:
  - — since the XTI API is defined by a standard, most of the TETware code should compile without change on a conforming system

- whereas disadvantages are:
  - — the system is difficult to set up, particularly when machine addresses must be specified
  - — the performance of TETware is poor when XTI is used, mainly due to the lack of a transport-independent event notification mechanism in XTI[11]
  - — since the XTI standard does not define a means to specify transport addresses in a machine-independent way, programs which must deal with addresses need to make use of network-related header files and functions not defined by the XTI standard; thus you may find it necessary to change some of the transport-specific part of the TETware code before you can compile it on your machine even when XTI is used
  - — it is not possible to build the Thread-safe C and C++ APIs in a way that conforms fully to the requirements for POSIX threads when XTI is used.

On balance, it is probably best not to consider using XTI for practical testing unless BSD networking is not available on your system, or the implementation of the BSD network API on you system is such that substantial changes to the transport-specific TETware code would be required.

If you **do** decide to use XTI, you must specify which underlying transport provider(s) you want to use. Support is provided for you to use XTI with TCP/IP or the OSI connection-orientated transport.

3.3.3.2 How do you want to start the TCC daemon?

TETware uses a client/server architecture and one of the servers — the TCC daemon (`tccd`) — is required to run on each machine on which tests are to be run.

On UNIX systems this daemon may be started in one of several ways as follows:

- At system boot time from an entry in the system initialisation file `/etc/rc`.
- At system boot time from an entry in the `init` configuration file `/etc/inittab`.
- On demand, from an entry in the `inetd` configuration file `/etc/inetd.conf`.[12]

The way in which `tccd` expects to be started depends on certain compiler options that you specify when TETware is built. You must decide which version of `tccd` you want to build before proceeding with the TETware build stage. You can choose to build a different version of `tccd` on different machines if you want to.

The versions of `tccd` that you may choose to build are known as the **rc** version, the **inittab** version and the **inetd** version of `tccd`, respectively, corresponding to the different ways in which `tccd` may be started, as listed above.

---

11. If your system implements a SVID3-conforming `poll()` system call, you can compile TETware to use `poll()` to perform event notification — this helps to improve system performance when XTI is used.

12. This option is only available if you decide to use the socket network interface.

3.4 Preparing to build TETware

3.4.1 Introduction

Once you have made the decisions described above, you can prepare to build TETware with the makefiles supplied. The following subsections describe what you must do on each machine in order to configure TETware in accordance with the decisions that you have made.

3.4.2 The TETware configuration script — `configure`

The TETware distribution includes a configuration script which can be used to perform the tasks described in the next few sections. The name of this script is `configure` and it resides in the *tet-root* directory. This script attempts to guess the type of operating system on which it is running, invokes the `tetconfig` script and installs a suitable `defines.mk` file. (The `tetconfig` script and the `defines.mk` file are described in the next two sections.)

You should perform the following operations to run the `configure` script on each system:

1. To start, if you are not already there, change directory to *tet-root* thus:

   `cd` *tet-root*

2. Type

   `sh configure -t` *transport*

   where *transport* is one of `inet` to build Distributed TETware using sockets, `xti` to build Distributed TETware using XTI or `lite` to build TETware-Lite.

3. Perform any manual operations if instructed to do so by the `configure` script.

If you are building Distributed TETware to use sockets and `configure` is successful in configuring the TETware source tree, you can skip the next few sections and resume at the section entitled ‘‘Compiling the source’’ later in this chapter. Otherwise, `configure` will outline the things that you must do before TETware can be built. Details of these things are presented in the sections that follow.

3.4.3 The `tetconfig` script

If you did not use the `configure` script described in the previous section, you must run the configuration script in the file *tet-root*`/src/tetconfig`. This script asks you which network interface you want to use when building Distributed TETware, or whether you want to build TETware-Lite. You should reply with `inet` to build Distributed TETware to use the socket network interface, `xti` to build Distributed TETware to use the XTI network interface or `lite` to build TETware-Lite.

You should perform the following operations to run `tetconfig` on each system:

1. To start, if you are not already there, change directory to *tet-root*`/src` thus:

   `cd` *tet-root*`/src`

2. Type

   `sh tetconfig`

3. You must then select which version of TETware that you want to build. You should respond with `inet` to build Distributed TETware using sockets, `xti` to build Distributed

TETware using XTI, `lite` to build TETware-Lite, or `q` to quit without configuring TETware.

Each source directory contains makefile **include** files which contain transport-specific makefile fragments for TETware-Lite and each supported network interface. The TETware configuration script arranges for the appropriate **include** file to be picked up by each directory's makefile.

3.4.4 The `defines.mk` file

3.4.4.1 Introduction

There is a makefile definition file that you must supply on each system before TETware can be built; the name of this file is *tet-root*`/src/defines.mk`. This file contains system-specific definitions and is **include**d in the makefiles that are used to build TETware.

Several example makefile definition files suitable for use with various systems are included with the distribution and may be found in the directory *tet-root*`/src/defines`. If you are building TETware on one of the systems for which an example makefile definition file is supplied and are satisfied that the definitions provided are correct for your system, it should be sufficient to link or copy the example `.mk` file to *tet-root*`/src/defines.mk`. The `configure` script that is described in a previous section is able to recognise most of the common UNIX systems and performs this operation for you.

Note: The definitions in each of the example files supplied with the distribution assume that you are building Distributed TETware to use the socket network interface. If you want to build TETware-Lite or use a different network interface, you may need to edit the `defines.mk` file before it can be used.

If you have decided to use one of the example `defines.mk` files:

- If:
  - — the Java API source files are included in the TETware distribution; **and**
  - — the JDK is installed on your machine in a non-standard location; **and**
  - — the Java API is supported for use with your JDK version on your operating system; **and**
  - — you want to build the Java API:

  you will need to specify the location(s) of the JDK include files in the `defines.mk` file. You can skip the next few sections, follow the instructions in the section entitled ‘‘Support for Java’’, then resume at the section entitled ‘‘Compiling the source’’ later in this chapter.
- Otherwise: you can skip the next few sections and resume at the section entitled ‘‘Compiling the source’’ later in this chapter.

Alternatively, if you are building TETware on a system for which an example makefile definition file is not supplied or wish to change any of the example definitions, you should copy one of the supplied `.mk` files in the `defines` directory to *tet-root*`/src/defines.mk` and edit it as required.

A template is provided in the file *tet-root*`/src/defines/template.mk` for use in cases where it is necessary to create the `defines.mk` file from scratch. You can take a copy of this file and use it to construct your own makefile definition file if so required.

The following subsections describe each variable that must be provided in a `defines.mk` file.

3.4.4.2 `make` definitions related to the transport-independent source

The `make` variable `TET_CDEFS` in the `defines.mk` file should be set to the compiler definitions that are to be used when compiling the transport-independent code. This code is written to POSIX.1 using the common C language, although prototypes are provided in the appropriate header files for use with ANSI C compilers. One extension to POSIX.1 is required by the transport-independent source; namely, the symbol `NSIG` which defines the highest signal number plus one for signals supported by the implementation. Many systems already define `NSIG` in `<signal.h>` as an extension to POSIX.1. In this case, `NSIG` can be made available by compiling with the appropriate feature test macro defined in addition to `_POSIX_SOURCE`.

The set of compiler definitions (but not other compiler options) that is to be be used when compiling the transport-independent source should be assigned to the variable `TET_CDEFS` in `defines.mk`. On UNIX systems, `TET_CDEFS` should always include a definition for `_POSIX_SOURCE` (either explicitly or as a result of defining some other feature test macro).

For example, on a system which conforms fully to POSIX.1 and for which the highest signal number is 31, `TET_CDEFS` might be set as follows:

```
TET_CDEFS = -D_POSIX_SOURCE -DNSIG=32
```

3.4.4.3 `make` definitions related to the transport-specific source

3.4.4.3.1 Description

The `make` variable `DTET_CDEFS` in the `defines.mk` file should be set to the compiler definitions that are to be used when compiling the transport-specific portion of the C source in Distributed TETware. This code uses features that are not specified in POSIX.1, although they are available on most UNIX systems on which TCP/IP networking is implemented.

In addition, `DTET_CDEFS` is used in both TETware-Lite and Distributed TETware when compiling a few library files which need access to the full name space in order to recognise non-POSIX `errno` values and signals. Therefore, `DTET_CDEFS` should not normally include a definition for `_POSIX_SOURCE` (either explicitly or as a result of defining some other feature test macro).

`DTET_CDEFS` should also include a definition for `NSIG` (as described in the previous section) on systems which do not define this symbol in `<signal.h>` even when `_POSIX_SOURCE` is not defined.[16]

3.4.4.3.2 General compiler definitions for the transport-specific source

The set of compiler definitions (but not other options) that should be used when compiling the transport-specific source should be assigned to the variable `DTET_CDEFS` in `defines.mk`.

16. Alternatively on such systems, `NSIG` can be defined once in the `CDEFS` variable (described in a later subsection) instead of defining it in both the `TET_CDEFS` and `DTET_CDEFS` variables.

3.4.4.3.3 Specifying which version of `tccd` to build

You may need to add a compiler definition to `DTET_CDEFS` to cause the required version of `tccd` to be built, depending on which choice you made earlier as described in the section entitled ‘‘Before you build TETware’’ above:

- If you have decided to build the **rc** version of `tccd`, you do not need to add an extra compiler definition to `DTET_CDEFS`.
- If you have decided to build the **inittab** version of `tccd`, you should append `–DINITTAB` to `DTET_CDEFS`.
- If you have decided to build the **inetd** version of `tccd`, you should append `–DINETD` to `DTET_CDEFS`.[17]

3.4.4.3.4 Name of the Test Case Controller daemon

The name by which the Test Case Controller daemon is to be known should be assigned to the `TCCD` variable in `defines.mk`. `TCCD` should be set to `in.tccd` on a system where the TCC daemon is to be started by `inetd` (in accordance with established conventions); otherwise, `TCCD` should be set to `tccd`.

3.4.4.3.5 XTI-related compiler definitions

If you have chosen to use the XTI network interface, you must add extra compiler definitions to `DTET_CDEFS` in order to specify which transport provider(s) may be used. At least one of these definitions must appear when the XTI network interface is used, as follows:

- If you want to use XTI over TCP/IP, you should append `–DTCPTPI` to `DTET_CDEFS`.
- If you want to use XTI over the OSI connection-oriented transport, you should append `–DOSITPI` to `DTET_CDEFS`.

If your system implements a SVID3-conforming `poll()` system call, you can compile TETware to use `poll()` to perform event notification instead of having to use a polling loop that includes a `sleep()` function call. It is recommended that you do this if possible since use of `poll()` for this purpose helps to overcome the performance problems that would otherwise occur when only XTI-conforming functions are used. If you want to use `poll()` in this way, you should append `–DSVID3_POLL` to `DTET_CDEFS`.

3.4.4.3.6 Examples

All the examples presented here assume that you will build Distributed TETware to use the socket network interface.

On a system that makes network-related symbols visible in header files when the feature test macro `_ALL_SOURCE` is defined and you choose to build the **rc** version of `tccd` and to use the socket network interface, you might make the following definitions in the `defines.mk` file:

```
TCCD = tccd
DTET_CDEFS = -D_ALL_SOURCE
```

17. This option is only available if you have chosen to use the socket network interface.

or, to build the **inetd** version of `tccd` on the same system, you might make the following definitions in the `defines.mk` file:

```
TCCD = in.tccd
DTET_CDEFS = -D_ALL_SOURCE -DINETD
```

To build the **inetd** version of `tccd` on a system that conforms to the UNIX98 standard, you should make the following definitions in the `defines.mk` file:

```
TCCD = in.tccd
DTET_CDEFS = -D_XOPEN_SOURCE=500 -DINETD
```

3.4.4.4 Other `make` definitions

3.4.4.4.1 Introduction

The following subsections describe other `make` variables that you may need to set in your `defines.mk` file.

3.4.4.4.2 C compiler

The name of the C compiler is specified by the `CC` variable. For example, the following is appropriate on many systems:

```
CC = cc
```

Other examples could be:

```
CC = c89
```

or

```
CC = gcc
```

3.4.4.4.3 The command to perform partial linking

The name of the command that performs partial linking is specified by the `LD_R` variable. This command should invoke the linker in a mode that preserves relocation bits in the output file so that the output file may be used an input to a subsequent linker invocation.

For example, the following is appropriate on many UNIX systems:

```
LD_R = ld -r
```

3.4.4.4.4 Common compiler definitions and header file directories

Compiler definitions and header file directories that are to be specified when compiling all TETware source should be assigned to the `CDEFS` variable in `defines.mk`. These will always include at least the following:

```
CDEFS = -I$(INC) -I$(DINC)
```

and may include other –`I` and –`D` options as well.

3.4.4.4.5 Other compiler options

Other compiler options should be assigned to the `COPTS` variable. For example, the following assignment might be used to invoke the optimiser on many systems:

```
COPTS = -O
```

3.4.4.4.6 Loader options

Loader options should be assigned to the LDFLAGS variable. For example, you might make the following assignment to pass the −z option to the loader:

```
LDFLAGS = -z
```

3.4.4.4.7 System libraries

System library names and −l options to be added to the end of the compiler link-edit command line should be assigned to the SYSLIBS variable. For example, if network access routines are not in the standard C library, you might make the following assignment:

```
SYSLIBS = -lsocket -lnsl
```

3.4.4.4.8 Archive library maintainer

The name of the archive library maintainer is specified by the AR variable. For example, the following assignment is appropriate on most systems:

```
AR = ar
```

3.4.4.4.9 lorder and tsort

If your system has working lorder and tsort utilities, set the LORDER and TSORT variables thus:

```
LORDER = lorder
TSORT = tsort
```

otherwise, make the following assignments:

```
LORDER = echo
TSORT = cat
```

3.4.4.4.10 mcs

On some systems, the format of an a.out file includes a .comment section that can be compressed by mcs −c. If your system has the mcs utility, set the MCS variable thus:

```
MCS = mcs
```

otherwise, make the following assignment:

```
MCS = :
```

3.4.4.4.11 ranlib

If your system requires that ranlib must be used to process an archive library after it has been updated by the archive library maintainer, set the RANLIB variable thus:

```
RANLIB = ranlib
```

otherwise, make the following assignment:

```
RANLIB = :
```

3.4.4.5 Support for C++

The variable C_PLUS is used to determine whether or not the C++ API should be built. If you do not want to build the C++ API, you should make the following assignment:

```
C_PLUS = CPLUSPLUS_NOT_SUPPORTED
```

Otherwise, to build the C++ API you should define the following variables:

C_PLUS The name of the C++ compiler.

C_SUFFIX The suffix used for C++ source files (without a leading . (dot)).

For example, if the name of the C++ compiler on your system is CC and it recognises *filename*.C as a C++ program source file, you would make the following assignments in the defines.mk file:

```
C_PLUS = CC
C_SUFFIX = C
```

3.4.4.6 Support for Threads

3.4.4.6.1 Generic Threads support

It is possible to build Thread-safe versions of the C and C++ APIs. Support is provided to build these components for use with either UNIX International (UI) threads or POSIX threads (but not both at the same time).

Note: On UNIX systems you only need to build the Thread-safe API libraries if you intend to write multi-threaded test cases. These libraries are not used by TETware programs.

The variable THR_COPTS is used to determine whether or not the Thread-safe APIs should be built. If you do not want to build the Thread-safe APIs, you should make the following assignment:

```
THR_COPTS = THREADS_NOT_SUPPORTED
```

Otherwise, to build the Thread-safe APIs you should define the following variables:

THR_COPTS The set of compiler options that are to be used **instead of** COPTS when compiling the Thread-safe code. To enable support for POSIX threads, this variable assignment should include –DTET_POSIX_THREADS. (The default is to use UI threads.)

For example, to build the Thread-safe APIs to use UI threads on a system where the compiler requires a –mt option when compiling multi-threaded code, you might make the following assignment:

```
THR_COPTS = $(COPTS) -mt
```

Or, to build the Thread-safe APIs to use POSIX threads on a system where the compiler requires the _THREAD_SAFE macro to be defined when compiling multi-threaded code, you might make the following assignment:

```
THR_COPTS = $(COPTS) -DTET_POSIX_THREADS -D_THREAD_SAFE
```

`TET_THR_CDEFS` The set of compiler definitions to be used **instead of** `TET_CDEFS` when compiling the transport-independent portion of the Thread-safe API code. This assignment will be similar to that of `TET_CDEFS` but should also make the threads symbols visible in the system header files.

`DTET_THR_CDEFS` The set of compiler definitions to be used **instead of** `DTET_CDEFS` when compiling the transport-specific portion of the Thread-safe API code. This assignment should be no less restrictive than the `DTET_CDEFS` assignment and should also make the threads symbols visible in the system header files.

3.4.4.6.2 Considerations when POSIX threads are used

The POSIX threads standard imposes restrictions on what functions can be called in a child process when the parent process contains more than one thread at the time that the child process is created. In order to conform to the POSIX requirements a child of a multi-threaded process may only call async-signal safe functions until such time that one of the `exec()` functions is called.

Note: When Distributed TETware is built to use the XTI network interface, it is not possible to build the Thread-safe APIs to conform fully to the POSIX requirements in this respect.

These requirements impose severe restrictions on both the test case and the API when the C and C++ thread-safe APIs are built to support POSIX threads. When a multi-threaded test purpose calls the `tet_fork()` or `tet_spawn()` API functions to create a new process:

— when the new process is started by a call to `tet_fork()`, the only API functions that the `(*childproc)()` function may call are `tet_exec()` or `tet_exit()`; it may not call any other API functions;

— the API must take care only to call async-signal safe functions in the child process; this means that the API cannot perform normal operations such as dynamic memory allocation or the reporting of error conditions to the journal.

If the API detects an attempt to call an API function other than `tet_exec()` or `tet_exit()` from the `(*childproc)()` function after a call to `tet_fork()` from a multi-threaded test purpose, it prints an error message to its standard error stream and exits with non-zero status.

The API must allocate static storage for argument and environment lists in calls to `tet_exec()` and `tet_spawn()` when the parent of the calling process (in the case of `tet_exec()`) or the calling process itself (in the case of `tet_spawn()`) contains more than one thread.

By default the API allocates space for 256 arguments and 256 environment strings, which should be sufficient for most purposes. If necessary you can change these values by including one or more of the following compiler definitions in the value of the `TET_COPTS` variable:

`TET_EXEC_MAXARGS`

The maximum number of arguments that can be supported by `tet_exec()` or `tet_spawn()` when the Thread-safe APIs are built in strict POSIX mode.

`TET_EXEC_MAXENVS`

The maximum number of environment strings that can be supported by `tet_exec()` or `tet_spawn()` when the Thread-safe APIs are built

in strict POSIX mode.

For example, to increase the maximum number of environment strings to 300, you would make an assignment similar to the following:

```
THR_COPTS = -DTET_EXEC_MAXENVS=300 [...]
```

3.4.4.6.3 Support for unrestricted POSIX threads

In practice, not all implementations of POSIX threads actually require the restriction imposed by the standard to be observed.

If:

— you know that the implementation of POSIX threads on your system does not restrict which functions may be called in a child of a multi-threaded parent process; **and**
— you do not need to write tests that must be portable to systems that require this restriction to be observed;

you can define `TET_UNRESTRICTED_POSIX_THREADS` instead of defining `TET_POSIX_THREADS` in the `THR_COPTS` assignment.

For example:

```
THR_COPTS = $(COPTS) -DTET_UNRESTRICTED_POSIX_THREADS [...]
```

When you do this, the API will continue to work ‘‘normally’’ even in the child of a multi-threaded parent; memory is dynamically allocated as required, diagnostics are printed to the journal and so forth.

3.4.4.7 Support for shared API libraries

It is possible to build shared versions of the C API libraries. The following shared library schemes are supported on UNIX systems:

- True dynamic linking, where library code is position-independent and the values of symbols are resolved by a linker at program runtime.
- The mechanism in which the runtime linker fills in pointers to imported and exported symbols whose names must be defined when the library is built.

Note: The main advantage of using shared API libraries is that the amount of disk space occupied by a test suite is reduced when test cases are built to use shared API libraries. However, the use of shared API libraries introduces an additional element of complexity when setting up and running test suites, and makes it difficult to write test cases that are portable between systems. On balance, therefore, it is likely that many people will not want to use shared API libraries and so will not need to build them.

Note: You only need to build shared API libraries if you want to use them when building test cases. These libraries are not used by TETware programs.

The variable SHLIB_COPTS is used to determine whether or not the shared versions of the API libraries should be built. If you do not want to build the shared API libraries, you should make the following assignment:

```
SHLIB_COPTS = SHLIB_NOT_SUPPORTED
```

Otherwise, to build the shared API libraries you should define the following variables:

SO — The suffix used to denote a shared library file (including an initial . (dot)). For example, if a shared library is called lib*name*.so on your system, you would make the following assignment:

```
SO = .so
```

SHLIB_COPTS — The set of compiler options that are to be used **as well as** COPTS or THR_COPTS when compiling the shared library code. On systems that support true dynamic linking, this variable should be set to the option that causes the compiler to generate position-independent code. For example, on a system where the –kPIC option instructs the compiler to generate position-independent code, you would make the following assignment:

```
SHLIB_COPTS = -kPIC
```

SHLIB_CC — The name of the compiler that is to be used **instead of** the one specified by the CC variable when generating the object files that are to be put in a shared library. On systems that support true dynamic linking, the following assignment is usually sufficient:

```
SHLIB_CC = $(CC)
```

On systems that require lists of import and export symbols to be generated, this can be the name of a shell script that generates a file containing the symbol lists as well as compiling the source file.

SHLIB_CC is invoked by the makefiles exactly like a compiler, thus:

$(SHLIB_CC) *compiler-options* ... -c *filename*.c

When a user-supplied shell script specified by SHLIB_CC generates a symbol list file for *filename*.c, the name of the symbol file generated by the script should be *filename*.sym.

The file src/tet3/bin/symbuild.sh contains an example of a shell script that performs these tasks. If you decide that you can use symbuild.sh as part of your scheme for building shared libraries, you should make the following assignment:

```
SHLIB_CC = CC=$(CC) sh -x ../bin/symbuild.sh
```

SHLIB_BUILD — The name of the utility that builds a shared library from a set of object files. On systems that support true dynamic linking, this is usually $(CC) invoked with special options.

For example, on a system where the –G option instructs the compiler to build a shared library, you might make the following assignment:

```
SHLIB_BUILD = $(CC) -G
```

On systems that require lists of import and export symbols to be specified, SHLIB_BUILD can specify the name of a shell script provided by you that does whatever is necessary to generate the lists and invoke the library building tool(s). The directory src/tet3/bin contains examples of shell scripts that perform these tasks on various systems.

SHLIB_BUILD_END A set of arguments that are to be appended to the SHLIB_BUILD command line. The following command is invoked by the makefiles when a shared library is to be built:

```
$(SHLIB_BUILD) -o library-name object-files ... \
        $(SHLIB_BUILD_END)
```

On systems that support true dynamic linking, SHLIB_BUILD_END is normally left blank. On systems that require all symbols to be resolved at the time that a shared library is built, SHLIB_BUILD_END may be used to specify additional libraries (such as those specified by SYSLIBS).

THRSHLIB_BUILD_END

Used **instead of** SHLIB_BUILD_END when a thread-safe version of a shared library is to be built.

3.4.4.8 Support for Java

When the Java API source files are included in the distribution, it is possible to build the Java API on certain platforms on which the JDK is installed. Please refer to the TETware Release Notes for details of the platforms on which this API is supported.

When you configure your system to build the Java API it is helpful to understand the following points:

1. The TETware Java API consists of the following parts:
   - Java class files.
     These are built from Java source files and are put in an archive file.
   - The API libraries.
     These libraries are built from most of the source files that are used to build the C API, together with some C files which are specific to the Java API. These files are compiled into shared libraries which are accessed by the Java runtime using the Java Native Interface (JNI). Therefore, some of the make variables described in the section entitled ‘‘Support for shared API libraries’’ are also used when building the Java API libraries.
2. Since on most UNIX platforms there is no ‘‘standard’’ place to install the JDK, the locations of the Java include files cannot be specified in the example defines.mk files that are included with the TETware distribution. Instead, you will need to specify these locations on each system on which you build the API.
3. The Java runtime system makes extensive use of signals, and so it is not appropriate for the TCM to attempt to catch all the signals as is done in the C API. Neither is it appropriate to

expect users to specify the signals used by the Java runtime in execute mode configuration variables in each Java test suite. This is because some of the signals used by the Java runtime are POSIX signals and thus cannot be specified in TET_SIG_LEAVE in the execute mode configuration.

Instead, the list of signals used by the Java runtime (and thus must not be touched by the TCM) must be specified when the API library is built.

If the Java compiler javac can be executed and the directory *tet-root*/src/java is present, the API class files will be built.

The variable JAVA_CDEFS is used to determine whether or not the API libraries and support tools should be built. If you do not want to build the Java API libraries and support tools, you should make the following assignment:

```
JAVA_CDEFS = JAVA_NOT_SUPPORTED
```

Otherwise, to build the API libraries and support tools you should define the following variables:

JAVA_CDEFS — The set of compiler definitions to be used **as well as** TET_CDEFS or DTET_CDEFS when compiling the API support library. This should include the following values:

- A −I option that specifies the location of the generic Java include files.
- A −I option that specifies the location of the platform-specific Java include files.
- A −D option defining TET_SIG_LEAVE, that lists the signals that are used by the Java runtime system and therefore must be left alone by the C TCM.

For example, if the JDK has been installed in /usr/local/java on a particular system and the Java runtime uses the SIGALRM, SIGSEGV, SIGIO and SIGCHLD signals, you would make the following assignment:

```
JAVA_CDEFS = -I/usr/local/java/include \
        -I/usr/local/java/include/platform-name \
        -DTET_SIG_LEAVE=SIGALRM,SIGSEGV,SIGIO,SIGCHLD
```

Note: Please take time to determine the correct signal list for your platform, otherwise the Java API will not function correctly.

JAVA_COPTS — The list of compiler options that are to be used **as well as** COPTS when compiling the Java API library. Since the API library must be built as a shared object, on most systems this variable should have the same value as SHLIB_COPTS that is described in the section entitled ‘‘Support for shared API libraries’’ earlier in this chapter.

For example:

```
JAVA_COPTS = $(SHLIB_COPTS)
```

SHLIB_BUILD
SHLIB_CC
SO

These variables all have the meanings described in the section entitled ‘‘Support for shared API libraries’’ earlier in this chapter.

3.4.4.9 make definitions related to the Shell API

Certain make variables must be set in the defines.mk file for use with the Shell API. These variables are used to customise Shell API source files when those files are installed.

SH_STD_SIGNALS defines a space-separated list of standard signal numbers which must be specified to the Shell Test Case Manager. You should assign signal numbers to this variable that the Shell uses to identify the following signals on your system: SIGHUP, SIGINT, SIGQUIT, SIGILL, SIGABRT, SIGFPE, SIGPIPE, SIGALRM, SIGTERM, SIGUSR1, SIGUSR2, SIGTSTP, SIGCONT, SIGTTIN and SIGTTOU.

SH_SPEC_SIGNALS defines a space-separated list of signal numbers which must be left alone by the Shell Test Case Manager. You should assign signal numbers to this variable that the Shell uses to identify the following signals on your system: SIGKILL, SIGCHLD and SIGSTOP. In addition, many shells cannot trap other signals (e.g., SIGSEGV), so you may need to add these signals to SH_SPEC_SIGNALS as well.

For example:

```
# SIGHUP, SIGINT, SIGQUIT, SIGILL, SIGABRT, SIGFPE, SIGPIPE, SIGALRM,
# SIGTERM, SIGUSR1, SIGUSR2, SIGTSTP, SIGCONT, SIGTTIN, SIGTTOU
SH_STD_SIGNALS = 1 2 3 4 6 8 13 14 15 16 17 25 26 27 28

# SIGKILL, SIGCHLD, SIGSTOP, SIGSEGV
SH_SPEC_SIGNALS = 9 18 24 11
```

SH_NSIG should be set to the highest signal number supported by the shell, plus one. It normally has the same value as the −DNSIG included in TET_CDEFS.

The TETware distribution includes a shell script called sigtool which will help you determine the correct values for these variables. This script is located in the *tet-root*/src/helpers directory.

3.4.4.10 make definitions related to the Korn Shell API

The variables KSH_STD_SIGNALS, KSH_SPEC_SIGNALS and KSH_NSIG are the Korn Shell equivalents of the Shell API variables described in the previous subsection. By default they take their values from the corresponding Shell API variables, but they can be set to different values if necessary.

3.5 Compiling the source

When all the `make` definitions are correct, you may build TETware.

Note: Ensure that you have run the *tet-root*`/src/tetconfig` command and installed a suitable `defines.mk` file before you start on this section.[18]

You should perform the following operations on each system:

1. To start, if you are not already there, change directory to *tet-root*`/src` thus:

   `cd` *tet-root*`/src`

2. Type

   `make install`

   to build the tools and APIs and install them in their destination directories under *tet-root*.

3. If you have existing TET, ETET or dTET2 test cases that you want to run using TETware, these can be supported if you create compatibility files in the locations used by earlier TET versions (namely, the `posix_c` and `dtet2` directories under each of *tet-root*`/inc` and *tet-root*`/lib`).

   To create the compatibility files, type

   `make compat`

   If you want to use these compatibility files you must perform this operation **each time** that you (re)build TETware.[19]

Note: Some of the contributed test suites under *tet-root*`/contrib` can only be built if the compatibility files have been created.

3.6 Re-building TETware to use different options

If, having once built TETware, you decide that you want to re-build TETware to use a different network transport mechanism or to switch between TETware-Lite and Distributed TETware, you should perform the following operations:

1. If your system does not allow a file that is being executed to be written or unlinked, ensure that there is no `tccd` currently running.

2. Change directory to *tet-root*`/src`, thus:

   `cd` *tet-root*`/src`

18. Either by running the `configure` script or by performing these operations by hand.

19. The top-level makefile also contains a target called `compat_clean` which may be used to remove the compatibility files.

3. Type

   ```
   make clobber
   ```

   to remove the API library files and all the TETware object files and executables.[20]

4. Refer back to the section entitled ‘‘Before you build TETware’’ earlier in this chapter, and follow all the instructions from that point up to and including the section entitled ‘‘Compiling the source’’ above.

If you decide that you do not want to change the network transport mechanism but simply want to build a different version of `tccd`, you should perform the following operations on each system:

1. If your system does not allow a file that is being executed to be written or unlinked, ensure that there is no `tccd` currently running. If the current version is started automatically, be sure to disable the means by which it is started.
2. Refer to the sections entitled ‘‘How do you want to start the TCC daemon?’’ and ‘‘`make` definitions related to the transport-specific source’’ earlier in this chapter. Edit `defines.mk` to select the new version of `tccd` that you wish to build.
3. Change directory to *tet-root*`/src/tet3/tccd`, thus:

   ```
   cd tet-root/src/tet3/tccd
   ```

4. Type

   ```
   make clobber install
   ```

   to remove the old `tccd` executable and object files, and build and install the new version.
5. Refer to the section entitled ‘‘Starting `tccd`’’ elsewhere in this guide and follow the instructions on how to arrange for the new version of `tccd` to be started automatically.

20. Be sure to perform this action **before** you use the `tetconfig` script to configure TETware for your new option.

4. Configuring your system to run Distributed TETware

4.1 Introduction

If you have installed Distributed TETware, you must now configure each system to run TETware. If you have installed TETware-Lite, no configuration is necessary and so you do not need to perform the operations described in this chapter.

You may need to have administrative privilege in order to perform some of the operations described in this chapter.

If you have loaded Distributed TETware from a binary distribution, you will need to know what options were used when the distribution was built when following the instructions presented in this chapter. Binary distributions of Distributed TETware for UNIX systems that are supplied by The Open Group are normally built to use the socket network interface and include the **inetd** version of `tccd`.

4.2 Required system database entries

4.2.1 Introduction

The following subsections describe entries that you must make in system databases on each system.

4.2.2 Password database entry

When the Test Case Controller daemon `tccd` starts up on a UNIX system, it attempts to change its user and group IDs to those specified for the user `tet` in the system password database. In addition, `tccd` changes directory to the home directory specified for the user `tet`.

You should create a home directory for the user `tet` and add a suitable entry to the password database on each system. The user and group IDs allocated to the user `tet` need not imply any special privilege (i.e., they should have value 100 or greater).

4.2.3 Services database entry

If TETware has been built to use the socket network interface, you must add an entry to the **services** database[21] on each system where Distributed TETware is installed.

When `tccd` is built to use the socket network interface, it listens for requests on the well-known Internet TCP port number specified for the **tcc** service in the **services** database. This port number must be the same on all systems that are to participate in a set of remote or distributed tests. The **tcc** port number should be that of a non-privileged port (i.e., 1024 or greater).

---

21. When this database is contained in a file, it usually resides in the file `/etc/services`. If the **services** database on your system is obtained from a NIS server, you should ask your system administrator how best to add the entry described here to the database.

For example, to define the well-known port as TCP port 1234, you might add the following line to the **services** database on each TETware system:

```
tcc     1234/tcp
```

4.3 The systems equivalence file

If TETware has been built to use the socket network interface, you must create a file called `systems.equiv` which specifies the names of systems from which `tccd` may accept connection requests. This file resides in the home directory of the user `tet` (or in the home directory of the user that you will specify if you decide to run `tccd` with the `–u` command-line option).

You should create an entry in this file for each system that is to be permitted to send requests to `tccd` on this system. `tccd` will only process requests from another system if an entry for that system appears in the `systems.equiv` file.

An example `systems.equiv` file is included in the TETware distribution. You may copy this file to the `tccd` user's home directory and edit it as required.

Information about the format of the `systems.equiv` file is presented in a manual page at the back of the TETware User Guide.

4.4 Starting `tccd`

4.4.1 Introduction

You should make arrangements for `tccd` to be started on each machine where Distributed TETware is installed.

Note: You must start `tccd` on the local system as well as on remote systems.

Although it is possible for some of the versions of `tccd` to be started interactively by users, it is recommended that you arrange for `tccd` to be started automatically on each TETware system. This is so as to ensure that all instances of `tccd` execute in a known environment and with a known set of privileges. If this is not done, you will be unable to guarantee that test cases execute in a repeatable way.

If you have built Distributed TETware from a source distribution, the way that you arrange for `tccd` to be started depends on which version you chose to build as described in the section entitled ''Before you build TETware'' in the previous chapter. If you have loaded Distributed TETware from a binary distribution, this choice was made when the distribution was built: all the binary distributions for UNIX systems normally include the **INETD** version of `tccd`.

The following sections describe how to arrange to start the different versions of `tccd`. For each machine on which you want `tccd` to be started, you should perform the instructions in only **one** of the sections depending on which version of `tccd` has been built.

Additional command-line options are required by `tccd` when Distributed TETware is built to use the XTI network interface. For details of these options, refer to the section entitled ''Using `tccd` with XTI'' later in this chapter.

Further information about `tccd` is presented in a manual page at the back of the TETware User Guide.

You should **not** arrange to start `tccd` on a machine connected to an external network without first considering the security implications of doing so. This issue is discussed further in the section entitled ‘‘Network security considerations for Distributed TETware’’ in the TETware User Guide.

4.4.2 Starting the RC version of `tccd`

If you are using the **rc** version of `tccd`, you should add the following lines at a suitable place in one of the `/etc/rc` files:

```
if test -x tet-root/bin/tccd
then
        tet-root/bin/tccd && echo tccd started
fi
```

Once this is done, `tccd` will be started each time the system enters multi-user mode. It will be necessary for you to reboot the system in order to start `tccd`. When you reboot the system, look for the message

```
tccd started
```

on the system console when the system comes up multi-user, and check that `tccd` has printed a time-stamped START message to the file `/tmp/tccdlog`. Then, use `ps` to ensure that an instance of `tccd` is indeed running. If `tccd` failed to start correctly, you should check the file `/tmp/tccdlog` for diagnostic messages indicating the reason why `tccd` was unable to start execution.

4.4.3 Starting the INITTAB version of `tccd`

If you are using the **inittab** version of `tccd`, you should add the following line to the file `/etc/inittab`:[22]

```
tet1:3:respawn:tet-root/bin/tccd
```

Then, type

```
init q
```

to make `init` take account of the new `/etc/inittab` entry and start `tccd`. When you have done this, check that `tccd` has printed a time-stamped START message to the file `/tmp/tccdlog`. Then, use `ps` to check that an instance of `tccd` is indeed running. If `tccd` failed to start correctly, you should check the file `/tmp/tccdlog` for diagnostic messages indicating the reason why `tccd` was unable to start execution.

22. Note that this example assumes that your system starts multi-user operation with network services enabled when `init` enters run level 3. If your system uses a different run level (or levels) for this purpose, you must replace the `3` in this example with the correct value(s).

4.4.4 Starting `in.tccd` (the INETD version of `tccd`)

The name of the **inetd** version of `tccd` is `in.tccd`.

If you are using `in.tccd`, you should add the following line to the file `/etc/inetd.conf`:[23]

```
tcc  stream  tcp  nowait  tet  tet-root/bin/in.tccd  in.tccd [options ...]
```

where *options* are any command-line arguments that you want to pass to `in.tccd`.[24] (Information about command-line arguments is presented in the `tccd` manual page near the back of the TETware User Guide.)

Then, use `ps` to determine the process ID of `inetd` and type

```
kill -1 inetd-pid
```

where *inetd-pid* is the process ID of `inetd`. This will cause `inetd` to take account of the new entry in `/etc/inetd.conf` and start an instance of `in.tccd` each time a TETware process connects to the well-known **tcc** port.

Since `in.tccd` is run on demand by `inetd`, no START message is printed to the file `/tmp/tccdlog` and it is not possible to use `ps` to check whether or not `in.tccd` is running until another process connects to the well-known **tcc** port. If you have difficulties in getting `in.tccd` to start when you attempt to run a remote or distributed test case, it might be helpful to try and run one of the other versions of `tccd` first in order to isolate the cause of the problem.

4.4.5 Using `tccd` with XTI

The information in this section applies to both the **rc** and **inittab** versions of `tccd` when Distributed TETware is built to use the XTI network interface. (An **inetd** version of `tccd` cannot be built when the XTI network interface is used.)

When Distributed TETware is built to use the XTI network interface, you must specify additional command-line options when you start `tccd`.

The following options should appear on the `tccd` command line:

`-M` *mode* Specifies which transport provider to use. *mode* should be one of the following:

- `TCP` use XTI over TCP/IP (this is the default)
- `OSICO` use XTI over OSI connection-oriented transport

`-P` *tpi* Specifies the name of the transport provider identifier to use in XTI function calls. This is usually the name of a streams special file; for example `/dev/tcp`.

---

23. Note that older versions of `inetd` require that this line should appear **before** lines describing `inetd` internal services.

24. Note that on some systems the number of command-line arguments that `inetd` will pass to executed commands is limited. If you want to pass more arguments to `in.tccd` than are supported by `inetd` on your system, you will need to provide a shellscript wrapper for `in.tccd` and specify that in the `/etc/inetd.conf` file instead.

`-p` *addr* Specifies the XTI address string to use when listening for incoming connections. *addr* is a sequence of 2-digit hexadecimal values whose contents depend on the transport provider, network implementation and machine architecture.

For example, the following command might be appropriate when the TCP transport provider is to be used on a MIPS machine running UNIX System V Release 4:

```
tccd -P /dev/tcp -M TCP -p 0002nnnn00000000000000000000000
```

where *nnnn* is the hexadecimal value of the TCP port number on which `tccd` should listen.

APPENDICES

A. The TETware end-user licence

++++++++++++++++++++++++++ TET END USER LICENCE ++++++++++++++++++++++++++

BY OPENING THE PACKAGE, YOU ARE CONSENTING TO BE BOUND BY THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, DO NOT INSTALL THE PRODUCT AND RETURN IT TO THE PLACE OF PURCHASE FOR A FULL REFUND.

TETWARE RELEASE 3.5 END USER LICENCE
REDISTRIBUTION NOT PERMITTED

This Agreement has two parts, applicable to the distributions as follows:

A. Free binary evaluation copies – valid for 90 days, full functionality – no warranty.

B. Free binary restricted versions – no warranty, limited functionality.

C. Licenced versions – full functionality, warranty fitness as described in documentation, includes source, binary and annual support.

PART I (A & B above) – TERMS APPLICABLE WHEN LICENCE FEES NOT (YET) PAID (LIMITED TO EVALUATION, EDUCATIONAL AND NON-PROFIT USE).

GRANT.

X/Open grants you a non-exclusive licence to use the Software free of charge if

a. you are a student, faculty member or staff member of an educational institution (K-12, junior college, college or library) or an employee of an organisation which meets X/Open's criteria for a charitable non-profit organisation; or

b. your use of the Software is for the purpose of evaluating whether to purchase an ongoing licence to the Software.

The evaluation period for use by or on behalf of a commercial entity is limited to 90 days; evaluation use by others is not subject to this 90 day limit. Government agencies (other than public libraries) are not considered educational or charitable non-profit organisations for purposes of this Agreement. If you are using the Software free of charge, you are not entitled to hard-copy documentation, support or telephone assistance. If you fit within the description above, you may use the Software for any purpose and without fee.

DISCLAIMER OF WARRANTY.

Free of charge Software is provided on an ''AS IS'' basis, without warranty of any kind.

X/OPEN DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL X/OPEN BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

PART II (C above) – TERMS APPLICABLE WHEN LICENCE FEES PAID.

GRANT.

Subject to payment of applicable licence fees, X/Open grants to you a non-exclusive licence to use the Software and accompanying documentation (‘‘Documentation’’) as described below.

LIMITED WARRANTY.

X/Open warrants that for a period of ninety (90) days from the date of acquisition, the Software, if operated as directed, will substantially achieve the functionality described in the Documentation. X/Open does not warrant, however, that your use of the Software will be uninterrupted or that the operation of the Software will be error-free or secure.

SCOPE OF GRANT.

Permission to use for any purpose is hereby granted. Modification of the source is permitted. Redistribution of the source code is not permitted without express written permission of X/Open. Distribution of sources containing adaptations is expressly prohibited.

Redistribution of binaries or binary products containing TETware code is permitted subject to the following conditions:

— this copyright notice is included unchanged with any binary distribution;

— the company distributing binary versions notifies X/Open;

— the company distributing binary versions holds an annual TET support agreement in effect with X/Open for the period the product is being sold, or a one off binary distribution fee equal to four years annual support is paid.

Modifications sent to the authors are humbly accepted and it is their prerogative to make the modifications official.

Portions of this work contain code and documentation derived from other versions of the Test Environment Toolkit, which contain the following copyright notices:

The unmodified source code of those works is freely available from `ftp.xopen.org`. The modified code contained in TETware restricts the usage of that code as per this licence.

++