Release Notes for the Extended Test Environment Toolkit

Implementation for POSIX.1 Systems in ‘C’ and ‘C++’, XPG3 Shell, Korn Shell and Perl
by SunSoft, UNIX System Laboratories and the ETET Development Group.

Release 1.10.3 (03/31/94)

An Extension to

TET.1.10
by OSF, UNIX International and X/Open

Preface

These release notes describe an extended version of the *Test Environment Toolkit* known as the *Extended Test Environment Toolkit* (ETET). They describe the prerequisites for running this toolkit, what is supplied in the toolkit, and how to build the toolkit. The release notes also list the operating system platforms on which the Extended Test Environment Toolkit has been built and run.

Audience

This document is written for software engineers who will be porting their validation test suites into the Test Environment Toolkit.

Document Usage

This document is organized into the following chapters.

- Chapter 1 gives introductory information about the Extended Test Environment Toolkit.
- Chapter 2 describes how to build the Extended Test Environment Toolkit.
- Chapter 3 describes the new features provided with ETET.
- Chapter 4 describes the changes made since TET1.10
- Chapter 5 contains comments on specific areas of the specification.
- Chapter 6 describes extensions to the specification.
- Chapter 7 describes known problems with this release.
- Chapter 8 describes proposed future extensions to the ETET.

Related Documents

For additional information about the Extended Test Environment Toolkit, refer to the following documents which are provided with this release:

- *Test Environment Toolkit: Architectural, Functional, and Interface Specification, Version 4.5*
- *Extended Test Environment Toolkit: Programmer's Guide, Revision 1.10.3 (94/03/31)*

Typographic and Keying Conventions

This document uses the following typographic conventions:

Bold	In formats and command descriptions, **bold** words or characters represent commands or keywords that you must use literally, including pathnames. In examples, information that you enter appears in **bold**.
Italics	*Italic words or characters in formats and command descriptions represent values that you must supply.*
`Typewriter font`	`Information that the system displays appears in this typeface. Examples also appear in this typeface.`

Problem Reporting

ETET is supported by a group of volunteers, (see appendix A). If you have problems with the ETET send electronic mail to:

etet_support@novell.co.uk

Include the following information in your mailing:

- Your name, company or organization, and email address
- Hardware you are running on
- Operating system version ETET is installed on
- ETET version number
- Description of encountered problem, including how the problem occurred and how it can be reproduced
- A copy of the ETET configuration file(s) for the affected modes
- A copy of the ETET journal file
- A suggested fix for the problem, if possible

A template to use for reporting problems by email is provided in the file **doc/support/err.template**.

1. Introducing the Extended Test Environment Toolkit

The Extended Test Environment Toolkit (ETET) is an extension to the Test Environment Toolkit (TET) which provides a uniform framework, or test scaffold, into which test suites can be incorporated. This enables test suites from different vendors to share a common interface, and so improves portability.

The ETET has been designed to be upward compatible with TET1.10, so that existing tests require no changes to execute with an ETET harness. ETET implements several of the future directions stated in the *TET Architectural, Functional and Interface Specification*, and also introduces new ways of executing existing tests.

The objective of the ETET project is to extend the functionality of TET to support its commercial use outside of the conformance test arena in a way that is compatible with the base TET.

This release of ETET is the fourth revision of ETET based on Release 1.10 of the TET, and is known as ETET Release 1.10.3. This release is a maintenance release.

This implementation of the ETET currently consists of:

- A test case controller providing support for the building, execution, and clean-up of test scenarios (**tcc**) with the following extensions:
  - New Scenario key words: **parallel**, **group**, **repeat**, **timed_loop**, **random**
  - Scenario Syntax extensions
  - Addition of TET_SUITE_ROOT to allow test suites to be located in a different directory other than under TET_ROOT.
  - Addition of TET_RUN to support running of read-only copies of test suites.
  - Addition of TET_API_COMPLIANT and TET_TC_PASS_NAME that affect API compliance, output capture and passing arguments to the build and clean phases.
- ‘C’ and ‘C++’ Application Programming Interface;
- two ‘Shell’ Application Programming Interfaces (XPG3 Shell and Korn Shell).
- a ‘Perl’ Application Programming Interface.
- Improved and complete documentation.
- A contrib directory, containing several demonstration test suites, a test suite for the TET and ETET APIs, a TET portability library for non-POSIX systems and some sample tools.

Documentation

This release includes the TET Programmers’ Guide in source and formatted output. This is located in *tet_root*/**doc/prog_guide**.

A file of frequently asked questions is in *tet_root*/**doc/faq**.

A compressed tar archive containing postscript slides giving an overview of ETET is in *tet_root*/**doc/etet.slides.tar.Z**.

The following manual pages are included with this release:

Commands:

tet_root/**doc/man/man1/tcc.1**

Posix_c api routines

tet_root/**doc/man/man3/tet_child.3**
tet_root/**doc/man/man3/tet_fork.3**
tet_root/**doc/man/man3/tet_main.3**
tet_root/**doc/man/man3/tet_setblock.3**
tet_root/**doc/man/man3/tet_delete.3**
tet_root/**doc/man/man3/tet_getvar.3**
tet_root/**doc/man/man3/tet_reason.3**
tet_root/**doc/man/man3/tet_setcontext.3**
tet_root/**doc/man/man3/tet_exec.3**
tet_root/**doc/man/man3/tet_infoline.3**
tet_root/**doc/man/man3/tet_result.3**

File formats:

tet_root/**doc/man/man4/journal.4**
tet_root/**doc/man/man4/tet_scen.4**
tet_root/**doc/man/man4/tetclean.cfg.4**
tet_root/**doc/man/man4/tet_code.4**
tet_root/**doc/man/man4/tetbuild.cfg.4**
tet_root/**doc/man/man4/tetexec.cfg.4**

Refer to Chapter 6 of the TET specification for information on the Application Programming Interfaces and also to Chapter 5 of the ETET Programmers' Guide.

2. Building the Extended Test Environment Toolkit

This chapter describes how to set up your system and build the test environment. The next section lists the requirements which must be met before the ETET can be built.

2.1 Extended Test Environment Toolkit Requirements

ETET has the same base requirements as the TET. The Test Environment Toolkit requires that your system's operating system be compliant with the following:

- POSIX.1[1] with either standard 'C' or common 'C' support.
- XPG3, Vol. 1[2] (for the **shell, find, cp, rm,** and **make** commands)

To use the ETET Perl and C++ extensions requires the following:

- The Perl utility, Version 4.0 or later. This release has been tested with Perl 4.0 Patchlevel 35 and Patchlevel 36.
- A C++ compiler. This release has been tested with USL C++ Release 3.0.2 and GNU g++ Release 2.5.7

As of this writing, the Test Environment Toolkit has only been installed and tested on UNIX[3] type of systems. Note that this does not preclude its installation on other types of operating systems.

The ETET provides a series of makefiles that can be used to build the toolkit in a UNIX type of environment. If you are not running a UNIX type of operating system, you will have to develop your own equivalent method of building TET.

2.2 Installed Platforms

This release of the ETET has been installed and tested on the following operating system platforms (UNIX type of operating systems). Appropriate definitions for these are provided in the makefiles.

- UNIX System V Release 4.0, ICL DRS6000
- UNIX System V Release 4.2 or UnixWare 1.x, Intel processor.
- UNIX System V Release 4.2 MP or UnixWare 2.0, Compaq Systempro (2 i486 CPUs)
- AIX[4] Release 3.1, RS/6000.
- ULTRIX[5] 4.2, Decstation 3100
- SunOS[6] release 4.1

1. IEEE Std 1003.1-1990, *Portable Operating System Interface for Computer Environments*. ETET may support some non-POSIX operating systems through use of the *contrib/port* portability library (see later).
2. *X/Open Portability Guide Issue 3, Volume 1: XSI Commands and Utilities*
3. UNIX is a registered trademark of UNIX System Laboratories, Inc. in the U.S. and other countries.
4. AIX is a registered trademark of International Business Machines, Inc.
5. ULTRIX, DEC, and DIGITAL are a registered trademarks of Digital Equipment Corporation.
6. SunOS is a trademark of SUN Microsystems, Inc.

- SunOS 5.0 and later
- IRIX Release 4.0.5F System V, Silicon Graphics workstation.

2.3 What is in the Extended Test Environment Toolkit

This document assumes that your ETET root directory is *tet_root*. When you specify your ETET pathname, replace *tet_root* with your equivalent pathname. After you load the ETET, you should have the following subdirectories on your system:

tet_root/**inc/posix_c**	POSIX ‘C’ header files for use by test suites, currently only **tet_api.h**.
tet_root/**src**	A top-level makefile is provided in *tet_root*/**src/makefile**. This makefile will automatically install the test case controller, the POSIX C API, the XPG3 Shell API, the Korn Shell API and the Perl API. Refer to the README file for information on how to omit building certain language bindings. Before running this makefile you need to edit certain other makefiles, and its recommended you initially install the components separately.
tet_root/**src/posix_c**	Source directory for the TET tools and API, for POSIX ‘C’ and ‘C++’.
tet_root/**src/posix_c/tcc**	Source directory for the **tcc**.
tet_root/**src/posix_c/api**	Source directory for the TET Application Programming Interface for POSIX ‘C’.
tet_root/**src/posix_c/inc**	Header files, for the TET POSIX ‘C’ sources, including the file **tet_jrnl.h**.
tet_root/**src/posix_c/sun_lib**	Source directory for SunSoft API extensions.
tet_root/**src/posix_c/cplusplus**	Source directory for ‘C++’ language bindings.
tet_root/**src/xpg3sh/api**	Source directory for the TET Application Programming Interface for XPG3 shell.
tet_root/**src/ksh/api**	Source directory for the TET Application Programming Interface for Korn shell.
tet_root/**src/perl**	Source directory for the TET Application Programming Interface for perl shell.
tet_root/**bin**	Executable tools for the TET, including the **tcc**.
tet_root/**lib/posix_c**	POSIX ‘C’ library files for use by test suites, including **libapi.a**, **tcm.o**, and **tcmchild.o**.
tet_root/**lib/perl**	perl library files for use by test suites.
tet_root/**lib/sun_lib**	SunSoft API library.
tet_root/**contrib**	A directory of unsupported contributed software. For this release this contains several demonstration test suites, a test suite for the TET and ETET APIs, a TET portability library for non-POSIX systems and some tools.

tet_root/**doc** A directory of online documentation, including manual pages, the *TET Architectural, Functional and Interface Specification* and the *TET Programmer's Guide*.

2.4 Building the Environment

After the ETET is loaded, you can build the Test Environment Toolkit, including the **tcc** and the application programming interfaces, with the makefiles supplied. However, there is a makefile that needs to be edited before you can build the ETET.

- *tet_root*/**src/posix_c/makefile**

This makefile calls other makefiles in *tet_root*/**src/posix_c/tcc** and *tet_root*/**src/posix_c/api**. Edit the *tet_root*/**src/posix_c/makefile** and follow the instructions in the comments. You may wish to examine the directory *tet_root*/**src/posix_c/mkfiles** which contains sample makefiles for several different platforms.

The TET 'C' source is written to POSIX.1 using 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: the symbol NSIG. Many UNIX 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 **tcc** catches all signals it can from 1 up to (NSIG-1). On some systems this can cause problems with non-standard signals, for example SIGCLD if it is supported separately from SIGCHLD. Also, with the default signal handling the **tcc** cannot be suspended and restarted using job control signals. Two variables can be set in **makefile** to alter the signal handling. One specifies a list of signal numbers to ignore, the other a list of signal numbers to leave alone. If you wish to use job control with the **tcc**, you should include the job control signals in SIG_LEAVE.

2.5 Building the Tools and 'C' API

1. To start, if you are not already there, change directory to *tet_root*/**src/posix_c**.

   cd *tet_root*/**src/posix_c**

2. Type **make**. This will build the TET tool; **tcc**, as well as the 'C' API.

When you use **make**, an executable file for each TET tool component is placed in the directory *tet_root*/**bin**. The TET API files are generated and placed into the appropriate directories:

Name	**Path**	**Description**
tcm.o	*tet_root*/**lib/posix_c**	Test Case Manager **main()** routine. For processes to be executed by the **tcc**
tcmchild.o	*tet_root*/**lib/posix_c**	Executed processes **main()** routine. For processes to be executed by **tet_exec()**
libapi.a	*tet_root*/**lib/posix_c**	Library of API interfaces

In addition, the following files are also available for use with the Application Programming Interface.

Name	Path	Description
tet_api.h	*tet_root*/**inc/posix_c**	Header containing useful symbol definitions and declarations/prototypes for all API interfaces
llib-ltcm.c **llib-ltcmc.c**	*tet_root*/**src/posix_c/api**	Lint libraries defining the API interfaces. Use the one which corresponds to the object file in which **main()** is defined (**tcm.o** or **tcmchild.o**). They can be used directly or turned into libraries for use with **lint -l** by doing a **make LINTLIB** and installing the **.ln** files on the system.

2.6 Building the XPG3 Shell API

1. To start, if you are not already there, change directory to *tet_root*/**src/xpg3sh/api**.

   cd *tet_root*/**src/xpg3sh/api**

2. Edit the **makefile** and change the lists of signal numbers to the correct values for your system. Note that if you are using SunOS the makefile is called **makefile.sunos**. Some other sample makefiles are located in the directory *tet_root*/**src/xpg3sh/api/mkfiles**

3. Type **make**. This will install the shell API files **tcm.sh** and **tetapi.sh** in the directory *tet_root*/**lib/xpg3sh**.

2.7 Building the Korn Shell API

The Korn Shell API builds in a similar way to the XPG3 Shell API:

1. To start, if you are not already there, change directory to *tet_root*/**src/ksh/api**.

   cd *tet_root*/**src/ksh/api**

2. Edit the **makefile** and change the lists of signal numbers to the correct values for your system. Note that, some other sample makefiles are located in the directory *tet_root*/**src/ksh/api/mkfiles**

3. Type **make**. This will install the shell API files **tcm.ksh** and **tetapi.ksh** in the directory *tet_root*/**lib/ksh**.

See the later section *Using the Korn Shell API binding*.

2.8 Building the Perl API

1. To start, if you are not already there, change directory to *tet_root*/**src/perl**

   cd *tet_root*/**src/perl**

2. The **Makefile** should not need editing except for the PATH **PERL_DIR**.

3. Type **make**. This will install the files needed to build under this API.

2.9 Building the C++ API

1. To start, if you are not already there, change directory to *tet_root*/**src/posix_c/cplusplus**

   cd *tet_root*/**src/posix_c/cplusplus**

2. Edit the **makefile** and change the name of the ‘C’ compiler and the ‘C++’ compiler to the correct values for your system. The default values are **cc** and **CC** respectively.

3. Type **make**. This will install the files needed to build under this API.
4. To run a simple test of the ‘C++’ API refer to the **cplusdemo** described in the next section.

2.10 Running the Extended Test Environment Toolkit Demonstrations

The ETET contains several demonstration test suites in *tet_root*/**contrib. These are as follows:**

Name	Path	Description
demo	*tet_root*/**contrib/**	TET1.10 demonstration test suite for ‘C’ API. This is documented in the TET Programmers’ Guide.
cplusdemo	*tet_root*/**contrib/**	ETET demonstration test suite for ‘C++’ API. This is the ‘C’ API demonstration suite modified to use the ‘C++’ API.

The ‘C’ API Demonstration

This section describes how to run the simple demonstration TET test suite in *tet_root*/**contrib/demo**. This is the standard TET1.10 demonstration. The given invocation will build, execute, and clean the test suite.

1. To start, if you are not already there, change directory to to where the demonstration suite is:

   cd *tet_root*/**contrib/demo**

2. Make sure that *tet_root*/**bin** is in your PATH, and type the following command invocation:

   tcc -bec contrib/demo

   As the demonstration executes, it displays the following messages:

```
Journal file is: tet_root/contrib/demo/results/0001bec/journal
We have not set TET_OUTPUT_CAPTURE=True so all normal stdin/stdout/stderr
files are available.
If we had set output capture, the results logged by the API would not
be in the journal.
But these lines would.
```

   You can then look in *tet_root*/**contrib/demo/results/0001bec/journal** to see the results of the demonstration.

   A sample filter to generate a results summary is contained in *tet_root*/**contrib/usltools/vres**. This filter is an **awk** script. To install and run this script do the following:

   cp $TET_ROOT/contrib/usltools/vres $TET_ROOT/bin/vres
   chmod a+x $TET_ROOT/bin/vres
   vres results/0001bec/journal

   Which produces the following output:

```
Results: results/0001bec/journal
Total tests:            3       PASS = 3        FAIL = 0

Pass Breakdown:
Number of successes:    3       Number of warnings:     0
Number unsupported:     0       Number not in use:      0
Number of untested:     0       Number of FIP:          0

Failure Breakdown:
Number of failures:     0       Number unresolved:      0
Number uninitiated:     0       Number unreported:      0
```

An alternate execution scenario file is provided to demonstrate some of the new features of the ETET. This is provided in *tet_root*/contrib/demo/etet_scen.

This file contains

```
#       Demonstration test suite.
#       A default execution scenario to demo the ETET features
tests
        "starting scenario"
        /ts/tc1
        /ts/tc2
        "next is the last test case"
        /ts/tc3
        "done"

all
        "pick a random test"
        :random:^tests
        "repeat four times /ts/tc1"
        :repeat,4:@/ts/tc1
        "run all the tests in parallel"
        :parallel:^tests
        "pick random tests for 35 seconds"
        :timed_loop,35;random:^tests
```

The given invocations will build, and execute using the new features.

tcc -b contrib/demo

tcc -e -p -s etet_scen contrib/demo

The ‘C++’ API Demonstration

This section describes how to run the simple ‘C++’ demonstration TET test suite in *tet_root*/**contrib/cplusdemo**. The given invocation will build, execute, and clean the test suite.

1. To start, if you are not already there, change directory to to where the demonstration suite is:

   cd *tet_root*/**contrib/cplusdemo**

2. Review the contents of **ts/makefile** and ensure that it is correct for your system.

3. Make sure that *tet_root*/**bin** is in your PATH, and type the following command invocation:

   tcc -bec contrib/cplusdemo

   As the demonstration executes, it displays the following messages:

```
journal file name is: tet_root/contrib/cplusdemo/results/0001bec/journal
This is from the startup routine for tp1
Its interesting to note that C++ output
using cout does not get journaled.
Test 2 will explain why!.
This output comes from test purpose tp1
We have not set TET_OUTPUT_CAPTURE=True so all normal stdin/stdout/stderr
files are available.
If we had set output capture,  the results logged by the API would not be in the
 journal.
But these lines would.
tp3 child process called ok
```

You can then look in *tet_root*/**contrib/cplusdemo/results/0001bec/journal** to see the results of the demonstration.

3. New Features Of The Extended Test Environment Toolkit

3.1 Scenario Extensions

With the ETET comes a set of new scenario keywords that can be used when executing tests. These new functions are very useful since the machine under test can be subjected to stress testing and regression testing. The new keywords are described below.

parallel

> *:parallel[,<#count>]:<test_list>*
> *:group[,<#count>]:<test_list>*
>
> In ETET execute mode, all the tests in <test_list> are executed in parallel as separate processes. If <#count> is given, <#count> copies of each item in <test_list> are started. Scenario comments and journal messages are ignored in <test_list>. Scenario execution will not continue until all of the processes created by this execution mode have either completed or timed out. In build and clean mode each item in <test_list> is built or cleaned once sequentially. This capability allows for tests to execute in parallel. (Note: The "group" execution mode as described in the "TET Architectural, Functional, and Interface Specification" section 7.1.2.1 (2) is provided with ETET as a synonym for the "parallel" mode. It is proposed that the identifier "parallel" is much more descriptive of the execution mode and should be used.)

random

> *:random:<test_list>*
>
> In ETET execute mode, one test is chosen randomly from all the tests in <test_list>. Scenario comments and journal messages are ignored in <test_list>. In build and clean mode each item in <test_list> is built or cleaned once sequentially. This capability allows for Monte Carlo style testing.

remote

> *:remote,<node_id>:<test_list>*
>
> In ETET execute mode, the entire <test_list> is executed sequentially on the remote node referred to by <node_id>. In build & clean mode all tests referred to by <test_list> are built or cleaned on the remote machine referred to by <node_id> once each sequentially. The <node_id> identifier is a virtual node name that is used to look up the actual node name and method of connection in the tetremote.cfg file. Using a configuration file makes the test portable from one environment to another with a minimum amount of change. This capability allows for distributed, network end network oriented systems testing. **This has not yet been implemented.**

timed_loop

> *:timed_loop,<#seconds>:<test_list>*
>
> In ETET execute mode, each item in <test_list> is executed sequentially as long as <#seconds> seconds have not elapsed. When the last item in <test_list> has been executed, execution will start over again at the beginning of <test_list> as long as <#seconds> seconds have not elapsed. In build & clean mode all items in <test_list> are built or cleaned once each sequentially. This capability allows for a given execution duration. The timed_loop

capability is particularly useful in combination with the random and parallel capabilities to create Monte Carlo scenarios that are more predictable in duration.

repeat

:repeat

Continuously repeat the testlist. It is assumed that the loop will be terminated after expiration of some timeout such as the time_loop above.

Syntactic Extensions

ETET introduces two new test list formats for the <**test_list**> specifier. Firstly the ability to refer to another scenario as a test list:

^<scenario_name>

Secondly the ability to refer to a single test directly:

@<executable_path>

Additionally, the ability to compound execution modes on the same line makes the scenario file much more readable. This feature also avoids the need to create arbitrary scenarios or test_list files, and the name collision problems that these arbitrary scenarios can create. Syntactically, the group_execution modes can be combined as follows:

:<group_execution>;<group_execution>;<group_execution>:<test_list>

The following example scenario file exemplifies the kind of capabilities these new features would provide.

```
# This is an example scenario file that shows how the new
# execution modes are used.

# The parallel_tests scenario is a list of tests from which the
# monte_carlo scenario randomly chooses tests to execute.
parallel_tests
		/test/one
		/test/two
		/test/three
		/test/four

# The monte_carlo would test four parallel process each repeatedly
# executing random choices from the parallel_tests scenario for
# five minutes (300 seconds).
monte_carlo
		:parallel,4;timed_loop,300;random:^parallel_tests

# The net_test
net_tests
		/nettest/local_test
		:remote,net_partner:@/nettest/remote_test

# The network_test scenario executes both the tests in
# net_tests scenario in parallel.
network_test
		:parallel:^net_tests
```

3.2 Using the Korn Shell API binding

This is the ksh binding donated by Hewlett-Packard to the TET Workshop in *(tetworks 318)*. These bindings are thought to be between 3 and 5 times faster than the original XPG3 Shell API bindings.

The ETET Korn Shell API has been modified to make it consistent with the ETET XPG3 Shell API to allow use of the Extended TET features such as parallel execution.

The Korn Shell API bindings may be used as a direct replacement for the XPG3 Shell API bindings. This can be accomplished in several ways:

i. create links from *tet_root*/lib/ksh to *tet_root*/lib/xpg3sh, for example:

```
cd $TET_ROOT/lib/xpg3sh
ln -s ../ksh/tcm.ksh tcm.sh
ln -s ../ksh/tetapi.ksh tetapi.sh
```

ii. replace the xpg3sh bindings with the ksh bindings:

```
cp $TET_ROOT/lib/ksh/tcm.ksh $TET_ROOT/lib/xpg3sh/tcm.sh
cp $TET_ROOT/lib/ksh/tetapi.ksh $TET_ROOT/lib/xpg3sh/tetapi.sh
```

iii. alternately they can be used directly by placing the following line at the bottom of Korn Shell test scripts.

```
.       $TET_ROOT/lib/ksh/ksh.tcm
```

3.3 The Perl bindings to the Extended Test Environment Toolkit

The ETET adds support for Perl. In many respects this is similar to the **xpg3sh** binding, but it overcomes some of the problems inherent to the shell binding, such as poor performance and the limitations of the expressiveness of the shell language.

The ETET Perl bindings are generated and placed into the appropriate directories:

Name	Path	Description
tcm.pl	*tet_root*/**lib/perl**	Perl Test Case Manager.
api.pl	*tet_root*/**lib/perl**	Perl library of API interfaces.

The Perl API is equivalent to the posix_c API provided in TET1.10.

The following Perl Calling conventions should be observed:

```
&tet’setcontext;
&tet’setblock;
&tet’infoline("<info>");
&tet’result("<RESULT>");
&tet’delete("<test name>", ["reason"]);
&tet’reason("<test name>");
```

The default result code list is PASS, FAIL, UNRESOLVED, NOTINUSE, UNSUPPORTED, UNTESTED, UNINITIATED, NORESULT

For the exact usage of each call/variable, consult the TET Programmer’s Guide.

Variable references should take the form:

```
@iclist=(ic1,ic2...icn);
@ic1=(my_tp1);
@ic2=(my_tp2,my_tp3);

$tet’startup="my_startup_routine";
$tet’cleanup="my_cleanup_routine";

@tet’sig_leave_list=(...);
@tet’sig_ignore_list=(...);

$tet’thistest;
```

An ETET Perl API compliant program should adhere to the following structure:

set iclist, ic’s, optional setup/cleanup routines
code for subroutines
"require "$ENV{\"TET_ROOT\"}/lib/perl/tcm.pl";"

A example test suite written in Perl that tests both the TET API and the ETET extensions is provided in *tet_root*/**contrib/suite** and *tet_root*/**contrib/api**.

The default location for perl is assumed to be **/bin/perl**. If perl is not located there on your system, or is not symlinked, then to change the perl scripts used in the **contrib** directory, review the script *tet_root*/**contrib/suite/fixperl** and run it.

sh fixperl

The default example given by **fixperl** changes the execution path from **/bin/perl** to **/usr/local/gnu/bin/perl**.

3.3.1 The ETET Test Suite

ETET comes complete with a test suite for itself written in Perl. For a description of the test suite refer to the *ETET Test Plan Version 0.5* in *tet_root*/**doc/testplan**.

This test suite comprises the *tet_root*/**contrib/api** and *tet_root*/**contrib/suite** test suites:

Name	Path	Description
suite	*tet_root*/**contrib/**	This is a test suite written in Perl that tests the ETET ‘C’, Shell and Perl API’s. It tests both the base TET features and the new features of ETET. This test suite runs the tests in the *tet_root*/**contrib/api** directory. The **README** file describes the setup requirements to run this test suite.
api	*tet_root*/**contrib/**	This is a set of tests driven by by the **suite** above. These tests are written in ‘C’, Shell and Perl.

To run this suite, execute the **runtest** script provided:

```
sh runtest
```

3.4 The C++ bindings to the Extended Test Environment Toolkit

The C++ binding is for the *posix_c* API. ETET can support both C and C++ test suites simultaneously.

The C++ binding provided can be thought of as a "lightweight" binding, in that only a small part of the ETET requires compilation in C++, the majority of the code being compiled as before using the C compiler. This approach was suggested by [Wijnen, Harrison].

This method is used because of the differences between C and C++ (IO, memory management, use of void etc) which make it impossible to compile C code using a C++ compiler without modifying the existing code.

To overcome possible linkage problems, [Stroustrup] recommends using conditional compilation to create a common C and C+ header:

```
#ifdef __cplusplus
extern "C" {
#endif

    //C and C++ code here
```

```
#ifdef __cplusplus
}
#endif
```

The predefined macro `__cplusplus` is used to ensure the C++ constructs are edited out when the file is used as a C header. A function declared `extern "C"` still obeys the C++ type checking rules, so all functions must have full Standard C prototypes.

This C++ implementation also requires a C++ `main()` module.

Changes and Additions for C++"

The header file inc/posix_c/tet_api.h has been changed in the manner described above to support linkage to both C and C++ bindings.

Linkage to the ETET API for C++ test suites is provided by the following archive libraries.

Name	Path	Description
libCtcm.a	*tet_root*/**lib/posix_c**	C++ Test Case Manager **main()** routine. For processes to be executed by the **tcc**. This is equivalent to the 'C' API module **tcm.o**.
libCtcmc.a	*tet_root*/**lib/posix_c**	C++ executed processes **main()** routine. For processes to be executed by **tet_exec()**. This is equivalent to the 'C' API **tcmchild.o**.
libapi.a	*tet_root*/**lib/posix_c**	The 'C' API library interfaces

There have been minimal changes to the common C and C++ code. Two existing C modules containing the C main() routines have been copied and had their entry names changed.

C++ Module	C Module	Description
c_tcm.c	**tcm.c**	C **main()** routine renamed to **t_cmain()** for 'C++' API.
c_tcmchild.c	**tcmchild.c**	C **main()** routine renamed to **tet_childmain()** for 'C++' API.

A simple demonstration C++ test suite is included in contrib/cplusdemo.

References:


```
[Wijnen] B. Wijnen, Private communication, May 92

[Harrison] R. Harrison, Private Communication, May 93

[Stroustrup] B. Stroustrup, The C++ Programming Language, 2nd edition,
            Addison Wesley, 1991
```


3.5 The Portability library for BSD systems

ETET includes a TET portability library for BSD systems. This is taken from the MIT X test suite release 1.2. The portability library is designed for BSD systems, although may be useful as a base for porting the TET to other non-POSIX systems with some change. The file *tet_root/***contrib/port/README** should be examined for an overview and adaptability hints.

4. Changes Since the TET1.10 Release

4.1 'C' API

- Changes have been made to the handling of the temporary results file to allow parallel running of testsets. Backwards compatibility is the default and no change in behavior should be observed unless ETET is compiled with the definition -DSUNTEST.
- A sprintf statement in **resfile.c** has been updated to use **%d** instead of **%ld** for the **tet_thistest** variable (see tet bug report tet1.10.004).
- **libapi.a** has been added to the list of files to remove for the make clean target in *tet_root/***src/posix_c/api/makefile** (see tet bug report tet1.10.001).
- Several of the header files have had fixes to the prototypes (see tet bug reports tet1.10.062, tet1.10.064, tet1.10.065).

4.2 Shell API

- If IC's are numbered from 01,02,03...10,11 a problem was found that prevented IC's 02-09 from being executed in the all scenario. This has been fixed in this release (see tet bug report tet1.10.067).
- The setcontext in a subshell is implemented according to section 6.2.3.2 of the TET specification. This differs from the TET1.10 implementation (see tet bug report tet1.10.068).
- There have been some changes to tcm.sh to improve performance when processing large numbers of TP's and IC's.
- The Korn Shell API was added in ETET 1.10.2, providing a faster alternative to the existing XPG3 Shell API. Revisions have been made to the Korn Shell API in 1.10.3 for further compatibility with the XPG3 Shell API.

4.3 tcc

- Improved error reporting.
- Improved test suite termination and changes for termination when executing in parallel.
- Additional journal file entry for uname() information.
- Improved checking on value of TET_ROOT, and support of TET_SUITE_ROOT.
- Addition of support for new scenario keywords and extensions to the scenario syntax.
- The '-p' option has a fix to report the time as well as the test in progress. This was present in TET releases prior to 1.10, and was a problem with the scope of the zeetime[] array.
- Introduction of TET_EXTENDED for better support of parallel execution of tests.
- Introduction of TET_LOCK to control locking behavior, defaults to TET1.10 style locking.
- Reduce the memory allocation in the function collate_and_sort() (see tet bug report tet1.10.049).
- In some instances when tcc fails to exec a test case a child process is not waited for. A fix has been made to add an additional call to waitpid() (see tet bug report tet1.10.069).

4.4 Specification

This release is made with version 4.5 of the TET specification.

5. Notes on the Specification

The aim of this chapter is to clarify some points about the functionality of the TET that may not be immediately clear from reading the specification.

5.1 Reserved Result Code Names

The rerun and resume options of the **tcc** make use of the strings: 'b', 'e' and 'c'. Setting a result code name equal to one of these strings may cause unexpected results when the rerun or resume options are used.

5.2 The TET_OUTPUT_CAPTURE variable

This variable may be set independently for each mode of the **tcc**. When set to 'False' (the default) the **tcc** expects the appropriate mode of operation to generate results in a file called **tet_xres**. The format of the results contained in this file are dictated by section 7.2.1 of the specification.

When set to 'True', the **tcc** captures the *stderr* and *stdout* from the appropriate mode of operation, and makes no attempt to open the **tet_xres** file.

See also the section entitled ETET Further Extensions which introduces the TET_API_COMPLIANT and TET_TC_PASS_NAME configuration variables.

5.3 The Alternate Execution Directory

An alternate execution directory can be set using the **-a** option of the tcc (See also section 6.1.2 of these notes). Test cases are specified in a scenario in terms of a relative directory pathname ending with the test case target name. For each test case, the build and clean modes of the **tcc** are initiated in the appropriate test case directory which is relative to the test suite root directory. The execute mode of the **tcc** is initiated in a directory that is either relative to the alternate execution directory (if it is set), or else the test suite root directory.

5.4 The TET_EXEC_IN_PLACE variable

When set to 'False' (the default) this variable instructs the **tcc** to recursively copy the directory in which execution would have taken place (see the above section) to a new temporary directory (See section 6.1.3 of these notes). The execute mode of the **tcc** then takes place in the temporary directory, after which, the temporary directory is recursively removed.

6. Extensions to the Specification

The following features are implemented in addition to those required by the TET specification. This first section is provided by TET1.10.

6.1 Environment Variables

6.1.1 TET_ROOT

You can override the value of TET_ROOT compiled into **tcc** by setting TET_ROOT in your shell environment before executing **tcc**. For example

- For the C shell, type a line like the following at the shell prompt:

  `setenv TET_ROOT` <*your TET_ROOT path*>

- Or, for the Bourne shell, type the following line at the shell prompt:

  `TET_ROOT=`<*your TET_ROOT path*>`;export TET_ROOT`

  By doing this, you can then refer to <*your TET_ROOT path*> as $TET_ROOT in the shell.

6.1.2 TET_EXECUTE

The environment variable TET_EXECUTE can be set in a similar fashion to that of the TET_ROOT, and allows you to specify a default value for the alternate execution directory that is used by the **tcc**. Note however that a command line supplied value for the alternate execution directory has a higher precedence than that supplied by the TET_EXECUTE environment variable.

6.1.3 TET_TMP_DIR

During the execution of the **tcc** temporary files are usually created in unique sub_directories under a directory called **tet_tmp_dir**, which is located in the *tet_root*. Whilst every effort has been made to ensure that each sub-directory is removed as the **tcc** finishes, it is possible that an abnormal termination will mean that a sub-directory remains. Such directories should be removed to conserve disk space.

The user may specify an alternative temporary directory for use by **tcc** by setting the environment variable TET_TMP_DIR. This variable should be set in a similar fashion to the TET_ROOT variable discussed above. Performance problems due to network file systems can be helped by setting this variable to a directory on a local file system.

6.2 Extra Options

The following **tcc** command line options are provided in addition to those in the specification:

-l may be used to supply a one-off scenario line for execution without the need to create a scenario file.

-p enables progress reporting to standard output.

-y -n may be used to select parts of a scenario for processing. Test case execution lines are processed only if they match the specified search string (**-y**) or don't match it (**-n**).

See the manual page in *tet_root*/**doc/posix_c/tcc.1** for more details.

6.3 ETET Further extensions

6.3.1 TET_SUITE_ROOT

The environment variable TET_SUITE_ROOT can be set in a similar fashion to TET_ROOT, and allows you to specify the root directory of a test suite to be in a directory other than immediately under under TET_ROOT. This is an extension that allows flexibility in the location of test suites, a typical use of this feature is to enable a copy of ETET to be held in a central location and served across a network on a read-only mount point.

For example to run a test suite under **/home/ts** with the ETET located in **/TET_ROOT**, the following would be set in the users **.profile**.

```
# ETET
TET_ROOT=/TET_ROOT
TET_TMP_DIR=/tmp    # allows TET_ROOT to be read-only
TET_SUITE_ROOT=/home/ts
TET_EXECUTE=$HOME/TESTROOT
PATH=$PATH:$HOME/BIN:$TET_ROOT/bin
EDITOR=vi
export TET_ROOT TET_TMP_DIR TET_SUITE_ROOT PATH EDITOR TET_EXECUTE
```

If you are running an X/Open test suite some additional changes are required since they make assumptions based on relative pathnames for the location of the **bin** and **lib** directories.

```
cd $HOME
cd ..
ln -s $TET_ROOT/bin bin
ln -s $TET_ROOT/lib lib
```

6.3.2 TET_RUN

The environment variable TET_RUN should be set to a pathname which can be used as an alternate TET_SUITE_ROOT directory. Upon startup when this is set, the testsuite selected is copied across to the TET_RUN directory using the tar command. The build, clean, and execute modes then use the copy of the test suite in TET_RUN. A typical use of this extension is to permit running of read-only copies held on a central server.

6.3.3 TET_JOURNAL_PATH

This feature is still subject to revision

The environment variable TET_JOURNAL_PATH is used internally by ETET.

6.3.4 TET_EXTENDED

This environment variable allows tcc and the language bindings to alter behavior at runtime. This relates to temporary results file behavior. The TET specification states that the file tet_xres is to be used for temporary results and this is the default behavior. If set to true, then ETET uses an alternate strategy that allows multiple copies of **tcc** to execute without corrupting the temporary results files.

TET_EXTENDED is a boolean and may be set to "true" or "false", abbreviations of "t", "T", "f" and "F" are also accepted.

6.3.5 TET_LOCK

The environment variable TET_LOCK can be set in a similar fashion to TET_ROOT, and allows you to specify whether the locking method described in the TET specification is to be used (set by setting TET_LOCK=t). If unset then a different locking strategy is used which allows parallel execution for ETET test cases. When TET_EXTENDED is true, TET_LOCK is set to false.

6.3.6 TET Output Capture

This release includes the introduction of two new configuration variables that affect API compliance, output capture and passing arguments to the build and clean phases. These features are an implementation of the proposal generated by action items 9308.07 & 9308.08 from the August 1993 meeting of the TET workshop.

6.3.6.1 **TET_API_COMPLIANT**

This flag is introduced in ETET as a way to specify TET API compliance. This has been introduced to overcome the problem that there was only one flag to control both Output Capture and API compliance. These are separate features that need to be able to be specified independently.

TET_API_COMPLIANT is a boolean variable. If TET_API_COMPLIANT is not explicitly set then it defaults to !TET_OUTPUT_CAPTURE. If it is set, it controls whether or not **tcc** outputs IC and TP start and end messages and whether or not **tcc** uses the exit value of the executed test as the result code in the journal.

Note that, TET_API_COMPLIANT only applies during the execution phase. If a test is not API compliant, you have 1 TP per IC and results are reported or returned via the program exit code.

6.3.6.2 **TET_PASS_TC_NAME**

ETET introduces a new flag to specify that the test case name be passed to the build and clean tools. This has been introduced to overcome the overloading of the TET_OUTPUT_CAPTURE flag which among other things specified whether or not the test case name is passed to the build and clean tools.

TET_PASS_TC_NAME is a boolean variable.. This variable controls passing the TET_BUILD_FILE argument and the test name to the build tool. If it isn’t explicitly set, it gets it’s value from TET_OUTPUT_CAPTURE setting.

In order to maintain compatibility with current tests, it is necessary to have TET_OUTPUT_CAPTURE’s behavior remain unchanged in the absence of TET_PASS_TC_NAME. Note also that TET_PASS_TC_NAME only applies during the build & clean phases.

7. Known Problems With This Release

The following problems are carried forward from the TET1.10 release into ETET.

1. When using either the rerun or resume options of the **tcc** an error occurs if a test case has more than one hundred and twenty-eight invocable components. An interim solution, is to enlarge the value of ICS_MAX in **again.c**. A value should be used that is greater than the maximum number of invocable components that will be used in any test case.

2. The **tcc** makes use of the command **cp -r**. If this use of **cp** is not supported on your system you should install a script in your PATH that correctly interprets, and replicates the recursive option of **cp**.

8. ETET Future Directions

This section documents proposed future directions for ETET.

- *remote* scenario keyword

  See the section *Scenario Extensions* for a description of the proposed extension. This will allow a logical extension of the ETET to allow distributed testing with minimal change to the existing API's.

- TET_SEED

  In random mode, the journal will record the value of the seed used. Setting the environment variable TET_SEED will allow a random mode to replay a given scenario.

- Multi-threaded Support

  The C API will be enhanced to support multi-threading.

- Develop more abilities to deal with stress testing.
- Look at ways of making TET suites more maintainable.
- Ability to deal with testing of embedded systems.
- An Expect API (under development right now) - based on the OSF TCL API.

9. APPENDIX A: List of Contributors

W. Turnbow, SunSoft Inc., California, USA.
Email: William.Turnbow@eng.sun.com

R. Valenzuela, SunSoft Inc., California, USA
Email: Ricardo.Valenzuela@eng.sun.com

M.J. Convey, Novell UK.
Email: mike@novell.co.uk

Dr. R.Harrison, University of Southampton, UK.
Email: R.Harrison@ecs.soton.ac.uk

A.J. Josey, Novell UNIX Systems Group, UK.
Email: a.josey@novell.co.uk

J. Birchfield, Qualtrak Corporation.
Email: jb@qualtrak.com

S.Mullan, Hewlett-Packard Co.
Email: mullan_s@apollo.hp.com

S. March, Motorola.
Email: smarch@urbana.mcd.mot.com

M.Kam
Email: mkam@sco.com

J. Sanfilippo, Hewlett-Packard Co.
Email: jss@apollo.hp.com

If you wish to contribute and or volunteer to assist with the support of the ETET send electronic mail to etet_support@novell.co.uk.

CONTENTS