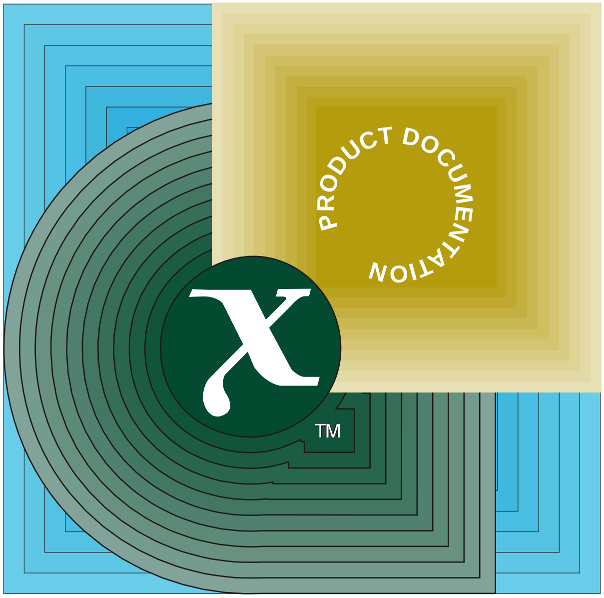
TETware Product Documentation

TETware
Training Guide



THE Open GROUP

[This page intentionally left blank]

THE Open GROUP

The Open Group

TETware Training Guide

February 2005

The Open Group

Document Number: F051

Comments relating to the material contained in this document may be submitted to:

The Open Group
Thames Tower
37-45 Station Road
Reading
Berkshire, RG1 1LX
United Kingdom.

Email: tetware_manager@opengroup.org

Introductory Module

Management overview

Introductory Module - Management overview

- 0.1 - What is TETware ?
- 0.2 - What has TETware been used for ?
- 0.3 - What are the benefits of using TETware ?
- 0.4 - Features and facilities
- 0.5 - TETware versions
- 0.6 - Supported systems
- 0.7 - TETware APIs
- 0.8 - Documentation
- 0.9 - List of support customers

0.1 - What is TETware ?

- TETware is The Open Group's supported version of the Test Environment Toolkit
- A system for the flexible construction and execution of tests in both single and multiple heterogeneous systems
- Originally developed in 1990 for UNIX only systems
- Distributed capability added with DTET and dTET2
- Extended functionality added with ETET

0.2 - What has TETware been used for ?

- Operating system tests
- Networking API tests
- GUI tests
- Network protocol tests
- Object oriented testing
- Data management tests

Some products that use TETware and its predecessors

- All The Open Group test suites; for example:
  - VSX,VSC (System interfaces and headers, Commands)
  - XNFS (Network File System)
  - XTEST (X-Windows)
  - VSORB (CORBA)
  - VST (XTI networking)
  - LSB-OS, LSB-FHS (LSB OS interfaces)
- ABI test suites for various processors:
  - INTEL
  - MIPS
  - Power PC
  - SPARC
  - LSB

THE Open GROUP

Feb-05 TETware training course 0-5

0.3 - What are the benefits of using TETware ?

- TETware provides a uniform framework, into which both non-distributed and distributed tests can be incorporated
- Test suites can share a common interface allowing for things such as ease of portability
- Test suite **authors** don't need to be concerned about the "administrative" aspects of testing, and so are able to concentrate on the actual testing task
- Test suite **users** only need to learn how to use a single, standard, test harness

THE Open GROUP

Feb-05 TETware training course 0-6

0.4 - Features and facilities

- Support for POSIX-style **assertion-based testing**
- Builds, executes and cleans up test suites
- Test scenarios can be defined using a powerful scenario language
- Test parameters can be specified using a flexible configuration variable mechanism
- Configuration information and test results are recorded in a journal
- Support for the standard POSIX result codes is built right in - user-defined results are also supported

THE Open GROUP

Feb-05 TETware training course 0-7

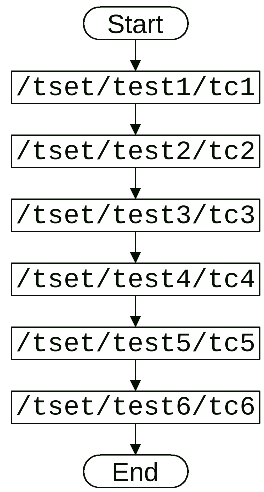
Features and facilities (cont'd)

- TETware processes several types of test case:
  - Non-distributed test cases on the local system
  - Non-distributed test cases on a single remote system
  - Concurrent processing of non-distributed test cases on several remote systems
  - Distributed test cases
- Test cases can be processed:
  - in sequence - one after the other
  - in parallel - several at the same time
  - at random - selected from a list
  - repeatedly

THE Open GROUP

Feb-05 TETware training course 0-8

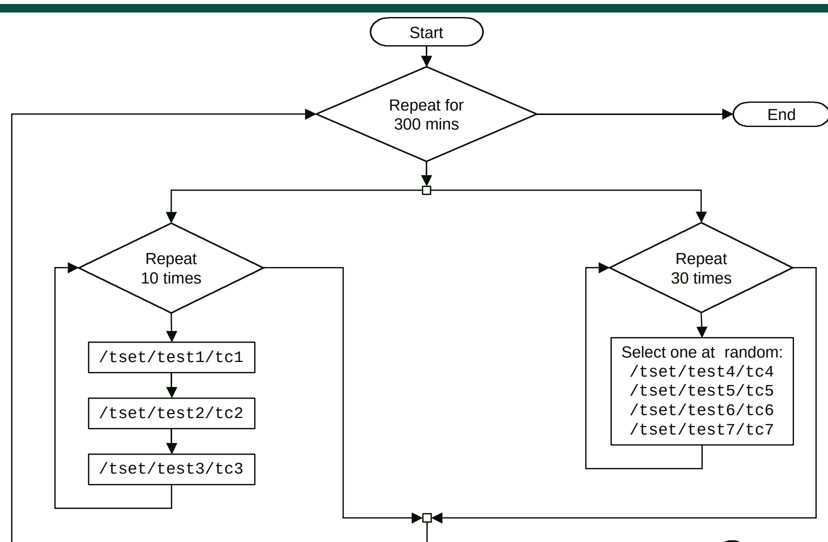
A typical verification test run



THE Open GROUP

Feb-05 TETware training course 0-9

An example stress test run



THE Open GROUP

Feb-05 TETware training course 0-10

0.5 - TETware versions

- TETware-Lite
  - processes non-distributed test cases on a single computer system (the local system)
- Distributed TETware
  - processes non-distributed test cases on
    - the local system
    - one or more remote systems
  - processes distributed test cases
    - a single test case which contains more than one part
    - each part runs on a different computer system

THE Open GROUP

Feb-05 TETware training course 0-11

0.6 - Supported systems

- TETware-Lite
  - UNIX systems (POSIX.1)
  - Linux
  - Windows NT, 2000, XP
  - Windows 9x
- Distributed TETware
  - UNIX systems (POSIX.1)
    - plus sockets or XTI for network support
  - Linux
  - Windows NT, 2000, XP

THE Open GROUP

Feb-05 TETware training course 0-12

Supported systems (cont'd)

- TETware has been built and exercised on:
  - AIX releases 4.x, 5L
  - HP-UX release 10.01, 10.10, 11.0, 11.22
  - Red Hat Linux
  - SuSE Linux
  - Solaris releases 2.4, 2.5, 2.6, 2.7,8,9,10
  - UnixWare 2.x, 7.x
  - Windows NT,2000, XP and Windows 9x

THE Open GROUP

Feb-05 TETware training course 0-13

0.7 - TETware APIs

- Supplied with TETware:
  - C
  - C++
  - XPG3 Shell (the Bourne Shell)
  - Korn Shell
  - POSIX Shell
  - Perl
  - Java
- Also available:
  - TCL, Python

THE Open GROUP

Feb-05 TETware training course 0-14

0.8 - Documentation

- TETware Programmers Guide
- TETware User Guide
- TETware Installation Guides for
  - UNIX operating systems
  - WIN32 operating systems
- TETware Release Notes
- TETware Knowledge Base

Feb-05 TETware training course THE Open GROUP 0-15

0.9 - List of Users

- The Open Group is grateful to the following companies who have contributed to supporting TETware development:
  - Aztek Engineering, Inc.
  - BrookTrout Technology
  - Compaq Computer Corporation
  - Fidelity Investments.
  - Fujitsu Ltd.
  - Hewlett-Packard
  - Hitachi Ltd.
  - Inprise Corporation
  - Intel Corporation
  - MKS, Inc.
  - Motorola, Inc.
  - Netscape Communications Corporation
  - Patni Computer Systems
  - Rational Software
  - Silicon Valley Networks Inc.
  - Sun Microsystems Inc.
  - Tenth Mountain Systems Inc.
  - Travellers Insurance Inc.
  - Verifone Inc.
  - Wind River Systems Inc.

Feb-05 TETware training course THE Open GROUP 0-16

Module 1

TETware basics

Module 1 - TETware basics

1.1 Product structure and concepts

- Test case types
- TETware versions
- The Test Case Controller (TCC)
- The Test Case Manager (TCM)
- The Application Program Interface (API)

Test case types

- Non-distributed test case
  - runs on a single computer system
  - does not usually interact with other computer systems
- Distributed test case
  - has two or more parts
  - each part runs on a different computer system

Non-distributed test case

- Local test case
  - processed on the local computer system
- Remote test case
  - processed on a remote computer system

Distributed test case

- Consists of several parts which interact with each other
- Typically used to test some kind of interaction between computer systems
- Each part is processed on a different system
- Each part contributes to a common result

TETware versions

- TETware-Lite
  - can process local test cases
  - easy to set up - just install and use
- Distributed TETware
  - can process local, remote and distributed test cases
  - uses a client-server architecture
  - more difficult to set up - a small amount of system administration is required

THE Open GROUP

Feb-05 TETware training course 1-7

The Test Case Controller (TCC)

- Processes test cases according to the selected **modes of operation**
- When **build** mode is selected, test cases are built
- When **execute** mode is selected, test cases are executed
- When **clean** mode is selected, test cases are cleaned up
- The TCC is a program called `tcc`

THE Open GROUP

Feb-05 TETware training course 1-8

The Test Case Manager (TCM)

- A wrapper that provides an execution environment for user-written test code
- Insulates the test code from the test environment
- Manages entries to the journal
- In Distributed TETware, ensures that parts of a distributed test case keep in step with each other
- The TCM is not a separate program but is linked with the user-written test code
- There is one TCM for each supported programming language

THE Open GROUP

Feb-05 TETware training course 1-9

The Application Program Interface (API)

- A library of functions which may be called from user-written test code
- For example, there are functions to:
  - make journal entries and register test results
  - access user-defined configuration variables
  - generate and execute processes
  - synchronise parts of a distributed test case
- API libraries are provided for use with each supported language

THE Open GROUP

Feb-05 TETware training course 1-10

1.2 TETware directory structure

- TETware expects to operate in a defined directory structure
- Many of these directories and the files in them have default names
- Most of these names can be changed using environment variables, configuration variables or `tcc` command-line options

TETware directories

- The top of the TETware directory hierarchy is called the **tet root** directory
- The directory that contains test suite files is called the **test suite root** directory
- TETware puts journal files and saved files below a directory called `results`
- TETware puts temporary files below a directory called `tet_tmp_dir`

TETware directories (cont'd)

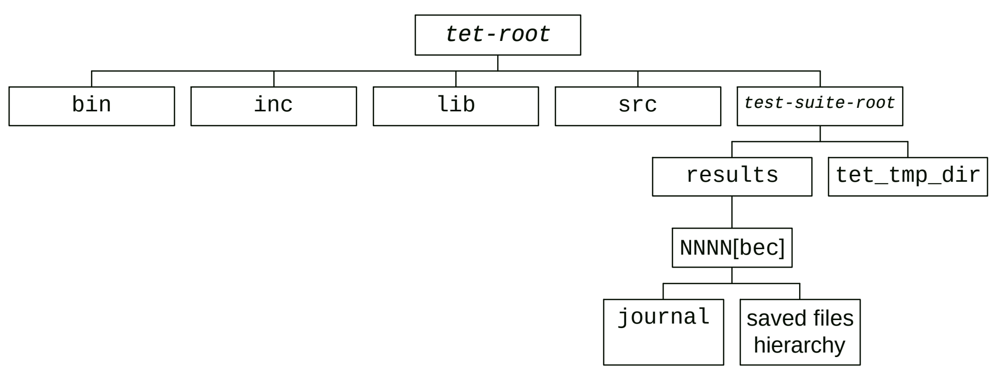
- It is possible to tell TETware to execute test cases below an **alternate execution directory**
- It is possible to tell TETware to copy the entire test suite to a **runtime directory** and process it there instead

THE Open GROUP

Feb-05 TETware training course 1-13

Directory structure diagram

TETware directory structure - top level



THE Open GROUP

Feb-05 TETware training course 1-14

1.3 Configurable files

- The scenario file
- Configuration files
- Result codes definitions
- System definitions

The scenario file

- A test scenario lists all the test cases and describes how they are to be processed
- The default name of the scenario file is `tet_scen`, located in the test suite root directory

Configuration files

- There is a set of configuration variables for each mode of operation
- The default names of the per-mode configuration files are:
  - `tetbuild.cfg`
  - `tetexec.cfg`
  - `tetclean.cfg`

Configuration files (cont'd)

- The build and clean mode configuration files are located in the test suite root directory
- The execute mode configuration file is located either in the alternate execution directory or in the test suite root directory
- In Distributed TETware these files must be provided on each system

Distributed configuration file

- In Distributed TETware this file is used to specify information about remote systems
- The name of this file is `tetdist.cfg`
- It is located in the test suite root directory on the local system

Feb-05 TETware training course THE Open GROUP 1-19

Result codes definitions

- TETware uses a default set of result codes
- Additional user-supplied result codes may be defined at the test suite level or at the installation level
- The default name of the result codes definition files is `tet_code`

Feb-05 TETware training course THE Open GROUP 1-20

Result codes definitions (cont'd)

- Each result code definition contains:
  - result code
    - a value between 0 and 127
    - values between 0 and 31 are reserved
  - result code name
    - a string describing the result, enclosed in double quotes
  - action indicator
    - tells the TCM what to do when this result occurs
    - possible values are `Continue` and `Abort`

Standard result codes

```
# example result codes file
0    "PASS"            Continue
1    "FAIL"            Continue
2    "UNRESOLVED"      Continue
3    "NOTINUSE"        Continue
4    "UNSUPPORTED"     Continue
5    "UNTESTED"        Continue
6    "UNINITIATED"     Continue
7    "NORESULT"        Continue
```

System definitions

- Distributed TETware identifies each computer system using a three-digit system ID
- System `000` always refers to the local system and other values refer to remote systems
- Each entry in the system definitions file maps a logical system ID to a value (such as a host name) that can be used to identify the system
- The name of this file is `systems` and it is located in the **tet root** directory on each system
- It is possible to map more than one logical system ID on to a physical machine

THE Open GROUP

Feb-05 TETware training course 1-23

Example `systems` files

- When the socket interface is used, fields are system ID and host name; for example:

```
000     argon
001     neon
002     xenon
```

- When the XTI interface is used, fields are system ID, host name and address string; for example:

```
000     argon      000204010a01020000000000000000
001     neon       000204010a01030000000000000000
002     xenon      000204010a01040000000000000000
```

THE Open GROUP

Feb-05 TETware training course 1-24

1.4 Journal files

- `tcc` generates a journal file for each test run
- The file includes output from test cases and tools
- The format of this file is designed to be read by user-supplied report writers
  - and is human-readable as well (with a little practice)

Journal file name and location

- The default name of the journal file is `journal`
- It is created in a subdirectory below the results directory
- The name of the subdirectory is *`NNNN`*`[bec]`
  - where *`NNNN`* is an ascending sequence number
  - and b, e and c indicate the selected modes of operation

Journal file format

- Each line in the file contains three fields:
  - Line type
  - Up to five parameters
  - Information area
- Each field is separated from the next by a vertical bar
- The parameters in the second field are separated from each other by spaces

Journal line types

- Each type of journal line is identified by a number
- For convenience of report writer authors, these numbers are defined in the file *tet-root*`/inc/tet3/tet_jrnl.h`
- Appendix C in the TETware User Guide contains a description of each type of journal line

Example journal file

```
0|3.2 19:23:02 19970710|User: andrew (105) TCC Start, Command line: tcc -ep
5|UNIX_SV deimos 4.2 1 386|Local System Information
20|/home/andrew/tet3/tests/tetexec.cfg 1|Config Start
30||TET_EXEC_IN_PLACE=false
30||TET_API_COMPLIANT=True
30||TET_PASS_TC_NAME=False
30||TET_VERSION=3.2
40||Config End
10|0 /ts/args/args 19:23:02|TC Start, scenario ref 1-0
15|0 3.2 1|TCM Start
400|0 1 1 19:23:04|IC Start
200|0 1 19:23:04|TP Start
520|0 1 00009857 1 1|this is tc16 parent
520|0 1 00009858 2 1|this is tc16 child
520|0 1 00009858 2 2|argument is "an-argument-string"
520|0 1 00009857 3 1|child exit status = 0
220|0 1 0 19:23:05|PASS
410|0 1 1 19:23:05|IC End
80|0 0 19:23:06|TC End, scenario ref 1-0
 ...
900|19:24:30|TCC End
```

Feb-05 TETware training course 1-29

1.5 Test suite structure

- Test case files
  - may be source files or executable programs
  - TETware will build test cases from source files if required
- Required utilities and data files
- Optional utilities and data files

Feb-05 TETware training course 1-30

Directory organisation

- All the files which make up a test suite reside below the test suite root directory
- It is a good idea to locate the test case files below a subdirectory
- In all but the most trivial of test suites, it is a good idea for each test case to have its own source directory
  - this organisation style is required if test cases are to be processed in parallel

Required files and utilities

- In both TETware Lite and Distributed TETware:
  - Configuration files for each mode of operation
  - Scenario file
- When building and/or cleaning test cases:
  - Build tool and/or Clean tool
- When processing remote or distributed test cases using Distributed TETware:
  - Per-mode configuration files on each system
  - Distributed configuration files

Optional files and utilities

- Prebuild tool
- Build fail tool
- Exec tool
- Result codes file
- Report writer

Scenario file

- Contains the scenarios that are to be used with the test suite
- Should at least contain a scenario named `all` which causes all the test cases to be processed

Report writer

- Is not used by TETware but is nevertheless an important part of any practical test suite
- There are probably almost as many report writers as there are test suites
- For example, the VSX report writer prints the assertion being tested when a test reports a non-PASS result

THE Open GROUP

Feb-05 TETware training course 1-35

API-conforming and non API-conforming test cases

- TETware is able to execute test cases that:
  - use a TETware API
  - do not use a TETware API
- Execution of non API-conforming test cases is only supported in order to enable existing test suites to be run by TETware
- New test cases should be written to use one of the TETware APIs
- `tcc` needs to know whether or not test cases in the test suite use an API

THE Open GROUP

Feb-05 TETware training course 1-36

Tools

- TETware uses user-defined tools to process test cases in each mode of operation
- Usually, tools are not API-conforming but some may use the API
- The tools are:
  - Pre-build tool
  - Build tool
  - Build fail tool
  - Exec tool
  - Clean tool

API-conforming test cases and tools

- May call TETware API functions
- Uses API functions to write to the journal and register results
- An API-conforming test case may contain several invocable components and test purposes

Non API-conforming test cases and tools

- Does not use the API - may not call API functions
- `tcc` captures the standard output and standard error and writes it to the journal
- `tcc` treats the test case as if it contained a single invocable component and test purpose
- `tcc` generates the IC and TP start and end lines that would be generated by the API
- `tcc` generates a single result based on the exit status (zero = PASS, non-zero = FAIL)

Feb-05 TETware training course THE Open GROUP 1-39

Pre-build tool

- Is optional
- Is executed before the build tool is invoked
- In Distributed TETware, only runs on the master system
- May be used to perform operations before the build tool is invoked, such as copying source files to slave systems
- Is always non API-conforming

Feb-05 TETware training course THE Open GROUP 1-40

Build tool

- Is required if `tcc` is to be invoked in build mode
- Should build the test case and install it if an alternate execution directory is to be used
- May be API-conforming or non API-conforming
- It is common to use `make` as the build tool

Feb-05 TETware training course 1-41

Build fail tool

- Is optional
- Is invoked if the build tool returns non-zero exit status
- Might be used to install a dummy test case if the real one could not be built
- Is always non API-conforming

Feb-05 TETware training course 1-42

Exec tool

- Is optional
- Is not normally required, but could be used to:
  - set up the test environment in some way
  - execute the test case under the control of a debugger
- Is always non API-conforming

THE Open GROUP

Feb-05 TETware training course 1-43

Clean tool

- Is required if `tcc` is to be invoked in clean mode
- Should clean up the test case, removing files generated during the build phase
- May be API-conforming or non API-conforming
- If the test cases are simple, `rm -f` is probably sufficient
- More complex test cases might use `make clean` or `make clobber`

THE Open GROUP

Feb-05 TETware training course 1-44

1.6 Test case structure

- A test suite contains one or more **test cases**
- Each test case contains one or more **invocable components**
- Each invocable component contains one or more **test purposes**

Test case

- A test case is an executable program
- It is the smallest unit that can be processed by `tcc`
- It is linked with a TCM and its API
- When `tcc` processes a test case in execute mode, it can select which invocable components are to be called by the TCM

Invocable component

- An invocable component is the smallest unit that the TCM will call
- In most cases an invocable component contains a single test purpose
- Normally when a test case is executed, the TCM calls all the invocable components in turn
- It is possible to specify individual invocable components in the test scenario

Test purpose

- A test purposes tests a single item of functionality and reports a result to the journal
- In an assertion-based test suite, a test purpose corresponds to an assertion to be tested
- A distributed test purpose:
  - consists of several parts running on different systems
  - each part contributes to the overall result

1.7 Relationship between TCC and test case

- This relationship defines how `tcc` executes test cases and tools
- It is affected by:
  - the selected mode(s) of operation
  - values of configuration variables
  - whether TETware-Lite or Distributed TETware is being used

The test case execution environment

- Execution directory
- Command-line usage
- Configuration variables
- Communication variables
- Output capture mode
- Execution results file
- Distributed TETware servers

Execution directory

- When in build or clean mode:
  - the test case source directory
- When in execute mode:
  - if `TET_EXEC_IN_PLACE` is true:
    - the alternate execution directory (if specified), otherwise the test case source directory
  - if `TET_EXEC_IN_PLACE` is false
    - a temporary execution directory

Command-line usage

- Build mode
  - prebuild tool
  - build tool
  - build fail tool
- Execute mode
  - exec tool specified
  - exec tool not specified
- Clean mode
  - clean tool

Build mode

- A TET_PREBUILD_TOOL is executed with TET_PREBUILD_FILE and the test case name as arguments
- The TET_BUILD_TOOL is executed with TET_BUILD_FILE as argument
  - The value of TET_PASS_TC_NAME determines whether or not the test case name is appended to the build tool command line
- A TET_BUILD_FAIL_TOOL is executed with TET_BUILD_FAIL_FILE and the test case name as arguments

THE Open GROUP

Feb-05 TETware training course 1-53

Execute mode

- If a TET_EXEC_TOOL has been specified, it is executed with TET_EXEC_FILE and the test case name as arguments
- If no exec tool has been specified, the test case is executed directly
- If an IC list has been specified in the scenario, it is appended to the command line

THE Open GROUP

Feb-05 TETware training course 1-54

Clean mode

- The TET_CLEAN_TOOL is executed with TET_CLEAN_FILE as argument
  - The value of TET_PASS_TC_NAME determines whether or not the test case name is appended to the clean tool command line

Feb-05 TETware training course THE Open GROUP 1-55

Communication variables

- tcc uses communication variables to pass information to the TCM
- Communication variables are environment variables, so they may also be accessed by test cases and tools
- tcc puts communication variables in the environment when it executes a test case or tool

Feb-05 TETware training course THE Open GROUP 1-56

List of communication variables

- `TET_ACTIVITY`
  - value of the `tcc` activity counter, used when making journal entries
- `TET_CODE`
  - name of a file containing the current set of result code definitions
- `TET_CONFIG`
  - name of a file containing the set of configuration variables for the current mode of operation

THE Open GROUP

Feb-05 TETware training course 1-57

List of communication variables (cont'd)

- `TET_EXECUTE`
  - path name of the alternate execution directory (if specified)
- `TET_ROOT`
  - path name of the tet root directory
- `TET_RUN`
  - path name of the runtime directory (if specified)
- `TET_SUITE_ROOT`
  - path name of the alternate test suite location (if specified)

THE Open GROUP

Feb-05 TETware training course 1-58

Output capture mode

- If output capture mode is enabled, `tcc` invokes a test case or tool with standard output and standard error redirected to a file
- When execution ends, `tcc` copies the captured output to the journal
- Typically, output capture mode is enabled when executing non API-conforming test cases and tools

Feb-05 TETware training course THE Open GROUP 1-59

Execution results file

- In TETware-Lite, an API-conforming test case or tool writes execution results to a file called `tet_xres` in the test case execution directory
- `tcc` copies the contents of this file to the journal
- If the test case or tool does not use an API, `tcc` deduces the result from the process exit status

Feb-05 TETware training course THE Open GROUP 1-60

Distributed TETware servers

- In Distributed TETware, `tcc` starts servers to provide additional facilities for use by APIs that support distributed testing
- These servers are:
  - `tetsyncd` - the synchronisation daemon
  - `tetxresd` - the execution results daemon

Feb-05 TETware training course THE Open GROUP 1-61

`tetsyncd` - the synchronisation daemon

- The API uses this server when synchronising between different parts of a distributed test case

Feb-05 TETware training course THE Open GROUP 1-62

tetxresd - the execution results daemon

- APIs that support distributed testing do not write execution results to a `tet_xres` file but instead send them to this server
- In Distributed TETware, `tcc` first checks this server for execution results and only looks for a `tet_xres` file if the server has not been used
- Thus the Distributed `tcc` is able to process test cases that use both types of API

Feb-05 TETware training course THE Open GROUP 1-63

Executing test cases without using tcc

- A test case which uses an API which does not support distributed testing can be run stand-alone
  - It may be necessary to put some of the communication variables in the environment; for example `TET_CODE` and `TET_CONFIG`

Feb-05 TETware training course THE Open GROUP 1-64

Executing test cases without using `tcc` (cont'd)

- A test case which uses an API which supports distributed testing expects Distributed TETware servers to be available
- Because of this, test cases that use these APIs cannot be run stand-alone but **must** be run under control of `tcc`
- This holds for both distributed and non-distributed test cases which use this type of API

THE Open GROUP

Feb-05 TETware training course 1-65

1.8 Comparison between TETware-Lite and Distributed TETware

- Local and remote systems
- Operation of `tcc`
- Test case types
- Operation of TCM and API
- Test case synchronisation
- Simple architecture diagrams

THE Open GROUP

Feb-05 TETware training course 1-66

Local and remote systems

- TETware-Lite can process test cases on the local system
  - that is: the system on which `tcc` runs
- Distributed TETware can process test cases on both local and remote systems

Operation of `tcc`

- In TETware-Lite, `tcc` performs all processing actions itself
- Distributed TETware uses a client-server architecture
  - `tcc` does not perform processing actions itself but instead sends requests to a server called `tccd`
  - this server runs on all the systems on which tests are to be processed
  - thus a single `tcc` invocation can control processing on a number of systems at once

Test case types

- TETware-Lite can process non-distributed test cases on a single computer (the local system)
- Distributed TETware can process:
  - non-distributed test cases on:
    - the local system (local test cases)
    - one or more remote systems (remote test cases)
  - distributed test cases

Feb-05 TETware training course THE Open GROUP 1-69

Operation of TCM and API

- In TETware-Lite none of the APIs support distributed testing
- In Distributed TETware:
  - the C, C++ and Java APIs
    - support distributed testing
    - can be used when writing both distributed and non-distributed test cases
  - the other APIs do not support distributed testing
    - but the Distributed `tcc` can process test cases which use these APIs as non-distributed test cases on remote systems

Feb-05 TETware training course THE Open GROUP 1-70

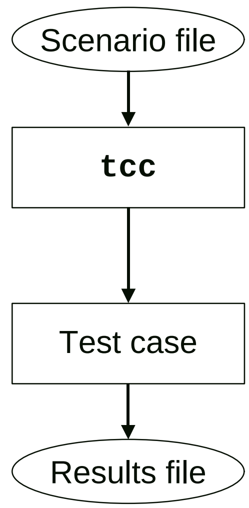
Operation of TCM and API (cont'd)

- In Distributed TETware: the C, C++ and Java APIs use the execution results daemon
- The other APIs - and all the APIs in TETware Lite - write to the `tet_xres` file

Test case synchronisation

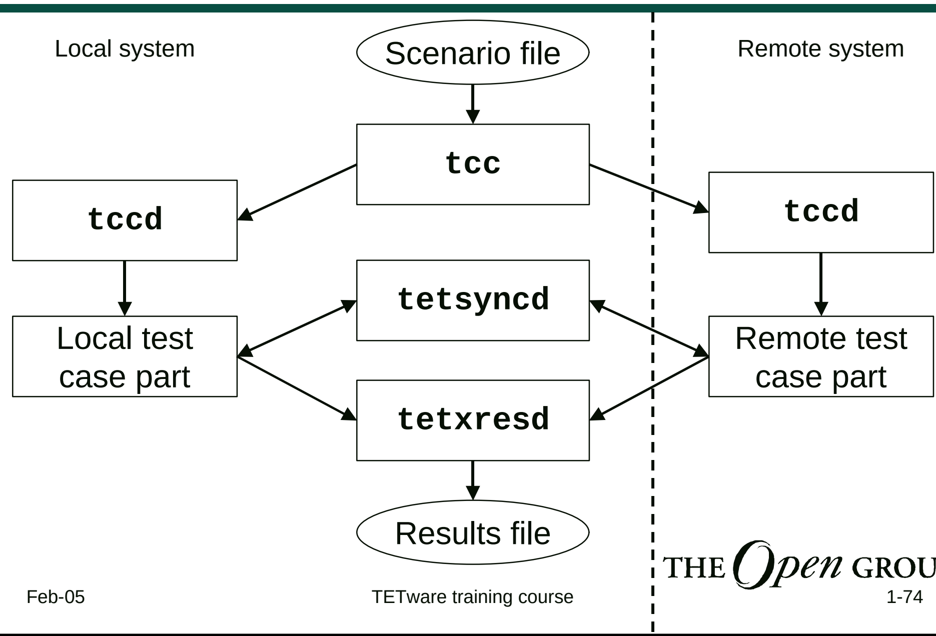
- When a distributed test case is being executed, the TCMs synchronise with each other at the following points:
  - when the test case starts executing
  - before the user-supplied startup function is called
  - at the start of each invocable component
  - at the start of each test purpose
  - at the end of each test purpose
  - before the user-supplied cleanup function is called

Simple architecture diagram for TETware-Lite



Feb-05 TETware training course THE Open GROUP 1-73

Simple architecture diagram for Distributed TETware



Feb-05 TETware training course THE Open GROUP 1-74

Exercise 1

Module 2

Building and installing TETware

Module 2 - Building and installing TETware

- 2.1 System requirements
- 2.2 Directory layout
- 2.3 Building the source distribution
- 2.4 Installing a binary distribution
- 2.5 Configuring your system to run Distributed TETware

2.1 System requirements

- UNIX systems
- Win32 systems:
  - Windows NT, 2000, XP
  - Windows 9x
- TETware-Lite
- Distributed TETware

THE Open GROUP

TETware-Lite on UNIX systems

- Base requirements on UNIX systems:
  - POSIX.1
  - XPG3 Volume 1 (for the shell and other commands)
- For the Korn Shell API:
  - the Korn Shell (`ksh`)
- For the POSIX Shell API:
  - A POSIX conforming shell (`sh`)
- For the Perl API:
  - `perl` version 5.0 or later

THE Open GROUP

TETware-Lite on UNIX systems (cont'd)

- For the C++ API
  - a C++ compiler
- For the thread-safe APIs - one of:
  - Unix International threads
  - POSIX threads
- For the Java API
  - JDK v1.1 or later

Feb-05 TETware training course THE Open GROUP 2-5

Support for shared API libraries

- It is possible to build shared versions of the libraries that are used by the C and C++ APIs
- On UNIX systems, support is provided for the following shared library schemes:
  - True dynamic linking (as found on Linux, Solaris, SVR4, Solaris and UnixWare)
  - HP-UX
  - AIX

Feb-05 TETware training course THE Open GROUP 2-6

Distributed TETware on UNIX systems

- Requirements as for TETware-Lite
- Network interface – one of:
  - Sockets
  - XTI

TETware on Win32 systems

- Uses a defined build environment:
  - Microsoft Visual C++
  - The MKS Toolkit for Windows NT (version 6 or later)
- For the Shell and Korn Shell APIs:
  - The MKS Toolkit for Windows NT
- For the Perl API
  - `perl` for Windows NT
- For the Java API
  - JDK v1.1

Support for Win32 systems

- TETware-Lite
  - Windows NT, 2000, XP
  - Windows 9x
- Distributed TETware
  - Windows NT, 2000, XP

Feb-05 TETware training course THE Open GROUP 2-9

2.2 Directory layout

- *`tet-root/`*
  - `bin` - executable programs
  - `contrib` - user-contributed software
  - `demo` - test suite root for the distributed demo
  - `doc` - TETware documentation
  - `inc` - API header files
  - `jdemo` - test suite root for the Java demo
  - `lib` - API library files
  - `src` - TETware source files

Feb-05 TETware training course THE Open GROUP 2-10

The TETware source tree

- *tet-root*/src/
  - `defines` - makefile definition files
  - `helpers` - configuration scripts
  - `java` - source for the Java API
  - `ksh` - source for the Korn Shell API
  - `perl` - source for the Perl API
  - `scripts` - source for shellscript tools
  - `tet3` - source for TETware programs, the distributed demo, the C and C++ APIs and parts of the Java API
  - `xpg3sh` - the Shell API

THE Open GROUP

Feb-05 TETware training course 2-11

2.3 Building the source distribution

- The method used to build the source is as near as possible the same on:
  - UNIX systems
  - Win32 systems
    - Windows NT, 2000, XP
    - Windows 9x
- Step-by-step instructions are presented in each of the TETware Installation Guides

THE Open GROUP

Feb-05 TETware training course 2-12

Building the source distribution on UNIX systems

- Load the distribution on to your machine
- Decide which version you want to build
- Select Distributed TETware options
- Configure the source tree (`configure`)
- Do you need to edit the `defines.mk` file?
- Build the source
- Install the compatibility links (optional)
- What to do next?

THE *Open* GROUP

Feb-05 TETware training course 2-13

Load the distribution on to your machine

- Create a directory and load the distribution into it
- This directory will become the **tet root** directory
- In the instructions that follow, this directory is referred to as *`tet-root`*

THE *Open* GROUP

Feb-05 TETware training course 2-14

Decide which version you want to build

- Which TETware version?
  - TETware-Lite
  - Distributed TETware
- Which Threads implementation should be used by the thread-safe APIs?
  - none
  - UNIX International threads
  - POSIX threads
- Is C++ supported?
- Is Java supported?

THE Open GROUP

Feb-05 TETware training course 2-15

Distributed TETware options on UNIX systems

- Which network interface?
  - sockets
  - XTI
- How do you want to start the TCC daemon?
  - from `/etc/rc` (the **rc** version)
  - from `/etc/inittab` (the **inittab** version)
  - from `/etc/inetd.conf` (the **inetd** version)
- These days most UNIX systems have sockets and `inetd`, so most people will build the **inetd** version using sockets

THE Open GROUP

Feb-05 TETware training course 2-16

Configure the source tree (`configure`)

- Run the configuration script, thus:

  ```
  cd tet-root
  sh configure -t transport
  ```

  where *`transport`* is one of:
  - `inet` to build Distributed TETware using sockets
  - `xti` to build Distributed TETware using XTI
  - `lite` to build TETware-Lite
- If `configure` can't work out which UNIX version you are running, you will have to run `tetconfig` and install a `defines.mk` file by hand

THE Open GROUP

Feb-05 TETware training course 2-17

Do you need to edit the `defines.mk` file?

- The file *`tet-root`*`/src/defines.mk` contains system-dependent definitions that will be used by `make` when you build the source
  - This file is installed for you when you run the `configure` script
- The `defines.mk` files that are supplied with the distribution assume that you are building Distributed TETware to use sockets

THE Open GROUP

Feb-05 TETware training course 2-18

Editing the `defines.mk` file

- If you want to build TETware-Lite or use a different network interface, you may need to edit the file after it is installed
- For example, when building TETware-Lite on a Solaris or UnixWare system you don't need to link with the network libraries:
  - edit `defines.mk` and search for the `SYSLIBS` assignment
  - remove `-lsocket -lnsl`

THE Open GROUP

Support for Java

- If you want to build the Java API, you may need to tell `make` where the JDK header files are located on your system
  - For example, on a Linux machine where the JDK is installed in `/usr/local/java`, the header files are located in `/usr/local/java/include` and `/usr/local/java/include/genunix`
- Search for the `JAVA_CDEFS` assignment in the `defines.mk` file, then follow the instructions in the comment

THE Open GROUP

Part of *tet-root*/src/defines/linux.mk

```
# support for Java
#
# JAVA_CDEFS is used in addition to TET_CDEFS/DTET_CDEFS when compiling
# the Java API.
# It is normally set to -Ipath-to-jdk-include-directory
# and includes a list of signals that the TCM should leave alone.
# Set JAVA_CDEFS to JAVA_NOT_SUPPORTED if Java is not supported on your
# system or if you don't want to build the Java API.
#
# Although the Java API is supported on Linux, the location of the JDK
# on your machine must be specified here before you can build the Java
# API support library.
# For example, if the JDK is installed in /usr/local/java on your machine,
# you would say:
# JAVA_CDEFS = -I/usr/local/java/include -I/usr/local/java/include/genunix \
#       -DTET_SIG_LEAVE='SIGALRM,SIGSEGV,SIGIO,SIGCHLD,SIGINT,SIGQUIT,SIGBUS,\
#               SIGILL,SIGABRT,SIGFPE,SIGTRAP,SIGXCPU,SIGXFSZ,SIGPIPE'
# but because I don't know where the JDK is installed on your machine,
# for now I must say:
JAVA_CDEFS = JAVA_NOT_SUPPORTED
# See "Support for Java" in the TETware Installation Guide for UNIX systems
# for more details.
```


Creating your own defines.mk file

- If none of the example `defines.mk` files are suitable, you must create your own
- Take a copy of the template file, thus:

  cd *tet-root*/src

  cp defines/template.mk defines.mk
- Customise the `defines.mk` file for your system
  - information to help you do this are contained in:
    - the comments in this file
    - section 3.4 of the TETware Installation Guide for UNIX Operating Systems

Build the source

- Now you can build the source, thus:

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

  `make install`
- Check that the build was successful

Feb-05 TETware training course THE Open GROUP 2-23

Install the compatibility links (optional)

- On UNIX systems it is possible to provide backwards compatibility with previous TET implementations
- To install the compatibility links, type:

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

  `make compat`
- If you need these links you must repeat this operation each time you rebuild TETware

Feb-05 TETware training course THE Open GROUP 2-24

What to do next?

- If you have built TETware-Lite, it is now ready to use
- If you have built Distributed TETware, you must now configure your system

Feb-05 TETware training course THE Open GROUP 2-25

Building the source distribution on Win32 systems

- Load the distribution on to your machine
- Decide which version you want to build
- Configure the source tree (`configure`)
- Do you need to edit the `defines.mk` file?
- Build the source

Feb-05 TETware training course THE Open GROUP 2-26

Load the distribution on to your machine

- Create a directory and load the distribution into it
- This directory will become the **tet root** directory
- In the instructions that follow, this directory is referred to as *`tet-root`*

Feb-05 TETware training course THE Open GROUP 2-27

Decide which version you want to build

- Which TETware version?
  - TETware-Lite
    - Windows NT, 2000, XP and Windows 9x
  - Distributed TETware
    - Windows NT, 2000, XP only
- There is no decision to make about Threads
  - the thread-safe APIs use the multi-threaded DLL version of the C runtime support library
- The C++ API is always built
- Is Java supported?

Feb-05 TETware training course THE Open GROUP 2-28

Distributed TETware options on Windows NT

- There are no choices to make when you build Distributed TETware on Windows NT
- The network transport is provided by the Windows Socket library (Winsock)
- The TCC daemon is started by a program called `tccdstart` - this behaves much like `inetd` on a UNIX system

Feb-05 TETware training course THE Open GROUP 2-29

Set up the defined build environment

- Set up the defined build environment now if you have not already done so
- Install Microsoft Visual C++ and the MKS Toolkit
  - the following instructions assume that:
    - you install MSVC++ in `c:/msdev`
    - you install the MKS toolkit in `c:/`
- Configure MKS `make` for MSVC++
- Arrange to use the MKS Shell as your command interpreter

Feb-05 TETware training course THE Open GROUP 2-30

Set up the defined build environment (cont'd)

- Modify your `profile.ksh:`
  - set and export the following environment variables:

```
TET_ROOT=tet-root
CCG=$TET_ROOT/src/tet3/compiler.ccg
INCLUDE="c:/msdev/vc/inc;c:/msdev/vc/mfc/inc"
LIB="c:/msdev/vc/lib;c:/msdev/vc/mfc/lib"
```

  - include `c:/mksnt` in `PATH`
- Ensure that the directory `c:/tmp` exists and is accessible to all users

Feb-05 TETware training course THE Open GROUP 2-31

Configure the source tree (`configure`)

- Run the configuration script, thus:

```
cd tet-root
sh configure -t transport
```

where *`transport`* is one of:

- `inet` to build Distributed TETware using sockets
- `lite` to build TETware-Lite

Feb-05 TETware training course THE Open GROUP 2-32

Do you need to edit the `defines.mk` file?

- The file *`tet-root`*`/src/defines.mk` contains system-dependent definitions that will be used by `make` when you build the source
  - This file is installed for you when you run the `configure` script
- The `defines.mk` files that are supplied with the distribution assume that you are building Distributed TETware

THE *Open* GROUP

Feb-05 TETware training course 2-33

Editing the `defines.mk` file

- If you want to build TETware-Lite, you need to edit the file after it is installed
- For example, when building TETware-Lite you don't need to link with the Winsock library:
  - edit `defines.mk` and search for the `SYSLIBS` assignment
  - remove `wsock32.lib`

THE *Open* GROUP

Feb-05 TETware training course 2-34

Support for Java

- If you want to build the Java API, you need to tell `make` where the JDK header files are located on your system
  - For example, if the JDK is installed in `c:/jdk1.1.8`, the header files are located in `c:/jdk1.1.8/include` and `c:/jdk1.1.8/include/win32`
- Search for the `JAVA_CDEFS` assignment in the `defines.mk` file, then follow the instructions in the comment

Part of *tet-root*/src/defines/msc+mks.mk

```
# support for Java
#
# JAVA_CDEFS is used in addition to TET_CDEFS/DTET_CDEFS when compiling
# the Java API.
# On Win32 systems it is set to -Ipath-to-jdk-include-directory.
# Set JAVA_CDEFS to JAVA_NOT_SUPPORTED if Java is not supported on your
# system or if you don't want to build the Java API.
#
# Although the Java API is supported on Win32 systems, the location of
# the JDK on your machine must be specified here before you can build
# the Java API support library.
# For example, if the JDK is installed in c:/jdk1.1.8 on your machine,
# you would say:
# JAVA_CDEFS = -Ic:/jdk1.1.8/include -Ic:/jdk1.1.8/include/win32
# but because I don't know where the JDK is installed on your machine,
# for now I must say:
JAVA_CDEFS = JAVA_NOT_SUPPORTED
# See "Support for Java" in the TETware Installation Guide for Win32 systems
# for more details.
```

Build the source

- Now you can build the source, thus:

  ```
  cd tet-root/src
  make install
  ```

- Check that the build was successful

Feb-05 TETware training course THE Open GROUP 2-37

What to do next?

- If you have built TETware-Lite, it is now ready to use
- If you have built Distributed TETware, you must now configure your system

Feb-05 TETware training course THE Open GROUP 2-38

2.4 Installing a binary distribution

- Load the distribution on to your machine
  - Create a directory and load the distribution into it
  - This directory will become the **tet root** directory
- If you have installed TETware-Lite, it is now ready to use
- If you have installed Distributed TETware, you must now configure your system

2.5 Configuring your system to run Distributed TETware

- UNIX systems
- Windows NT, 2000, XP

UNIX systems

- Create a new user called `tet`
- When the socket interface is used:
  - Add a `tcc` entry in the services database
  - Install a systems equivalence file in the `tet` home directory
- Arrange to start `tccd`
  - **rc** version
  - **inittab** version
  - **inetd** version

THE Open GROUP

Feb-05 TETware training course 2-41

Create a new user called `tet`

- Add an entry to the password database
- Create a home directory

THE Open GROUP

Feb-05 TETware training course 2-42

Add a `tcc` entry in the services database

- This entry defines the well-known port number that will be used by `tccd`
- It must be the same on all systems
- For example:

```
tcc     1234/tcp
```

- Usually the services database is in `/etc/services`
- Consult your system administrator if your system uses NIS

Install a systems equivalence file in the `tet` home directory

- Create a file called `systems.equiv` in the home directory of the user `tet` that you have just created
- Edit the file to contain the host names of systems that are permitted to connect to `tccd`, one per line
- If your system uses DNS, each host name should be fully qualified

Arrange to start `tccd`

- How you start `tccd` depends on which version you built:
  - **rc** version
  - **inittab** version
  - **inetd** version
- Follow the instructions in section 4.4 of the TETware Installation Guide for UNIX Operating Systems

Windows NT

- Add a `tcc` entry in the services database
- Install a systems equivalence file in your home directory
- Arrange to start `tccd`

Add a `tcc` entry in the services database

- This entry defines the well-known port number that will be used by `tccd`
- It must be the same on all systems
- For example:

```
tcc     1234/tcp
```

- Usually the services database is in `c:/winnt/system32/drivers/etc/services`

Install a systems equivalence file in your home directory

- Create a file called `systems.equiv` in your home directory
- Edit the file to contain the host names of systems that are permitted to connect to `tccd`, one per line
- If your system uses DNS, each host name may need to be fully qualified

Arrange to start `tccd`

- Open a new Korn Shell window
- Invoke the `tccd` bootstrap program, thus:

```
tccdstart
```


Exercise 2

Module 3

The Test Case Controller

Module 3 - The Test Case Controller

- 3.1 Modes of operation
- 3.2 Directory structure
- 3.3 Environment variables
- 3.4 Configuration variables
- 3.5 Test case processing
- 3.6 Saved files processing
- 3.7 Execution results processing
- 3.8 Rerun and Resume processing
- 3.9 Invocation
- 3.10 Test session interruption

3.1 Modes of operation

- `tcc` processes the scenario in one or more **modes of operation** as follows:
  - **build** mode
  - **execute** mode
  - **clean** mode
- Any combination of these modes can be specified when `tcc` is invoked

Build mode

- `tcc` builds each test case using the build tool
- The build tool is specified by `TET_BUILD_TOOL` in the build mode configuration
- The build tool should build the test case
- If an alternate execution directory is to be used, the build tool should also install the test case below there
- Often, `make` is used as the build tool

Execute mode

- `tcc` executes each test case
- Usually, `TET_EXEC_TOOL` is not specified, so the test case is executed directly

Clean mode

- `tcc` cleans up each test case using the clean tool
- The clean tool is specified by `TET_CLEAN_TOOL` in the clean mode configuration
- The clean tool should restore the file system to the state it was in before the test case was built
- Often, this is done by invoking `rm -f` or `make clobber`

3.2 Directory structure

- We have already learned about the directory structure in Module 1:
  - the **tet root** directory
  - the **test suite root** directory
  - the **temporary** directory
  - the **results** directory
  - the optional **alternate execution** directory
  - the optional **runtime** directory

The tet root directory

- This is the top of the TETware directory tree
- It must be specified by the TET_ROOT environment variable

The test suite root directory

- All the test suite files are located below this directory
- Usually it is located immediately below the **tet root** directory
- This is the place from which you invoke `tcc`

Feb-05 TETware training course THE Open GROUP 3-9

The temporary directory

- `tcc` executes test cases below this directory when `TET_EXEC_IN_PLACE` is false
- `tcc` creates this directory when necessary

Feb-05 TETware training course THE Open GROUP 3-10

The results directory

- This directory contains the journal files for each test run
- Each time `tcc` is invoked, it creates a new directory below here called *`NNNN`*`[bec]`
  - *`NNNN`* is an ascending sequence number
  - followed by one or more of b, e and c, depending on which modes of operation were selected

Feb-05 TETware training course THE Open GROUP 3-11

The optional alternate execution directory

- This directory can be specified if required
- When it is specified, test cases are executed in their location below this directory instead of below the test suite root directory
- Most of The Open Group's test suites use an alternate execution directory

Feb-05 TETware training course THE Open GROUP 3-12

The optional runtime directory

- When this directory is specified, `tcc` copies the entire test suite directory hierarchy to the runtime directory and processes it there
- This feature can be used when the test suite is contained on a read-only file system

3.3 Environment variables

- When `tcc` starts up, it reads in certain environment variables:
  - Required:
    - `TET_ROOT`
  - Optional:
    - `TET_EXECUTE`
    - `TET_SUITE_ROOT`
    - `TET_TMP_DIR`
    - `TET_RUN`

TET_ROOT

- This environment variable defines the location of the **tet root** directory
- This variable must be in the environment when `tcc` is invoked

THE Open GROUP

Feb-05 TETware training course 3-15

TET_EXECUTE

- If this environment variable is defined, it specifies the location of the **alternate execution** directory

THE Open GROUP

Feb-05 TETware training course 3-16

TET_SUITE_ROOT

- This environment variable may be used to specify the name of the directory containing the **test suite root** directory
- It can be used when the test suite root directory is not below the **tet root** directory

TET_TMP_DIR

- This environment variable can be used to specify an alternative location for the temporary directory
- If this variable is not defined, the temporary directory defaults to `*tet-root*/tet_tmp_dir`

TET_RUN

- If this environment variable is defined, it specifies the location of the runtime directory
  - `tcc` copies the test suite root directory hierarchy to the runtime directory and processes the test suite there instead
- This feature enables `tcc` to process a test suite which resides on a read-only file system

Feb-05 TETware training course THE Open GROUP 3-19

3.4 Configuration variables

- `tcc` uses certain configuration variables to determine how to process test cases
- A variable can be:
  - a Boolean variable
    - is either **true** or **false**
    - always has a default value (**false**)
  - a string variable
    - may have any value, or may be undefined
    - does not usually have a default value
- Configuration variables are specified in files - they are **not** environment variables

Feb-05 TETware training course THE Open GROUP 3-20

Categories of configuration variables

- There are variables to specify:
  - Tools and instruction files
  - Compatibility mode
  - Execution directory
  - API-conforming or non API-conforming
  - Saved files processing
  - Result code file name
  - Whether or not configuration variable expansion should be performed

THE Open GROUP

Tools and instruction files

- TET_PREBUILD_TOOL
- TET_PREBUILD_FILE
- TET_BUILD_TOOL
- TET_BUILD_FILE
- TET_BUILD_FAIL_TOOL
- TET_BUILD_FAIL_FILE
- TET_EXEC_TOOL
- TET_EXEC_FILE
- TET_CLEAN_TOOL
- TET_CLEAN_FILE

THE Open GROUP

Tools and instruction files (cont'd)

- `TET_*_TOOL` specifies a tool to use when processing a test case
- `TET_*_FILE` specifies an optional argument to pass to the tool
- Default: undefined
- Each tool only needs to be defined in the configuration for the mode that uses it
  - for example: `TET_BUILD_TOOL` should be defined in the build mode configuration

THE Open GROUP

Feb-05 TETware training course 3-23

Compatibility mode - TET_COMPAT

- Specifies the compatibility mode to use when interpreting scenarios
- Possible values: `dtet2` or `etet`
- Default: undefined
- If defined, must have the same value in all the per-mode configurations

THE Open GROUP

Feb-05 TETware training course 3-24

Execution directory - TET_EXEC_IN_PLACE

- Specifies whether test cases should be executed “in place” or in a temporary execution directory
- Default: `false`
- Only relevant in the execute mode configuration

API-conforming or non API-conforming

- `TET_OUTPUT_CAPTURE`
  - Specifies whether or not a test case or tool should be executed with output capture mode enabled
- `TET_API_COMPLIANT`
  - Specifies whether or not test cases or tools use the API
- `TET_PASS_TC_NAME`
  - Specifies whether or not the test case name should be passed to a build or clean tool
- Normally you only need to specify `TET_OUTPUT_CAPTURE` - appropriate defaults for the other two variables are taken from this one

TET_OUTPUT_CAPTURE

- Specifies whether or not a test case or tool should be executed with output capture mode enabled
- Default: `false`
- For convenience, provides default values for `TET_API_COMPLIANT` and `TET_PASS_TC_NAME`:
  - For an API-conforming test case or tool, set `TET_OUTPUT_CAPTURE` to `false`
  - For a non API-conforming test case or tool, set `TET_OUTPUT_CAPTURE` to `true`

TET_API_COMPLIANT

- Specifies whether or not test cases or tools use the API
- Default: inverse of `TET_OUTPUT_CAPTURE`

TET_PASS_TC_NAME

- Specifies whether or not the test case name should be passed to a build or clean tool
- Default: same as TET_OUTPUT_CAPTURE
- Only relevant in the build and clean mode configurations

Saved files processing

- TET_SAVE_FILES
  - Specifies which files to save after each test case is executed
- TET_TRANSFER_SAVE_FILES
  - In Distributed TETware, specifies whether or not saved files should be transferred to the local system and saved there instead

TET_SAVE_FILES

- A comma-separated list of files and/or directories to be copied to the saved files directory after test cases are executed
- Shell pattern-matching characters may be used
- Default: undefined
- Only relevant in the execute mode configuration

THE Open GROUP

Feb-05 TETware training course 3-31

TET_TRANSFER_SAVE_FILES

- Only in Distributed TETware
- If false, files specified by `TET_SAVE_FILES` on a remote system are saved on that system
- If true, files specified by `TET_SAVE_FILES` on a remote system are saved on the local system
- Default: `false`
- Only relevant in the execute mode configuration

THE Open GROUP

Feb-05 TETware training course 3-32

Result code file name - TET_RESCODES_FILE

- Specifies the name of a file containing result code definitions
- Default: `tet_code`
- If defined, must have the same value in all the per-mode configurations
- The file is searched for below the tet root and test suite root directories, so should be just a file name - not a path name

TET_EXPAND_CONF_VARS

- Specifies whether or not configuration variable expansion should be enabled in the per-mode configurations
- Default: `false`
- Configuration variable expansion may be performed on the assignments for:
  - All the `TET_*_TOOL` and `TET_*_FILE` variables
  - User-defined variables

Configuration variable expansion

- When TET_EXPAND_CONF_VARS is true, the value of one configuration variable may be interpolated in the assignment for another variable
- For example:

```
TET_EXPAND_CONF_VARS=true
PRESENTER=Andrew
MESSAGE=This course is being presented by ${PRESENTER}
```


Special variables defined by TETware

- It is also possible to interpolate the values of some special variables which are defined internally
- The special variables include:

  ${TET_ROOT} (the location of the **tet root** directory)
  ${TET_TSROOT} (the location of the **test suite root** directory)

- For example:

```
TET_EXPAND_CONF_VARS=true
TET_BUILD_TOOL=${TET_TSROOT}/bin/mybuildtool
```

- When you do this, you avoid the need to specify hard-wired path names in a configuration file

`tcc` variables in Distributed TETware

- In Distributed TETware, `tcc` reads some variables from the master configuration and some from the per-system configurations
- Variables read from the master configuration:
  - are specified on the local system
  - do not have a `TET_REM`*nnn*`_` prefix
- Variables read from a per-system configuration:
  - may be specified on the local system, or on the related remote system
  - may have a `TET_REM`*nnn*`_` prefix

THE Open GROUP

Feb-05 TETware training course 3-37

Variables read from the master configuration

- `TET_OUTPUT_CAPTURE`
  - `TET_API_COMPLIANT`
  - `TET_PASS_TC_NAME`
- `TET_COMPAT`
- `TET_EXEC_IN_PLACE`
- `TET_RESCODES_FILE`

THE Open GROUP

Feb-05 TETware training course 3-38

Variables read from the per-system configurations

- All the TET_*_TOOL and TET_*_FILE variables
- TET_EXPAND_CONF_VARS
- TET_SAVE_FILES
- TET_TRANSFER_SAVE_FILES

Feb-05 TETware training course THE Open GROUP 3-39

3.5 Test case processing

- This section describes how tcc processes test cases in:
  - build mode
  - execute mode
  - clean mode
- This processing is affected by the settings of several configuration variables
  - mainly TET_EXEC_IN_PLACE and TET_OUTPUT_CAPTURE

Feb-05 TETware training course THE Open GROUP 3-40

Common processing for test cases and tools

- This processing is performed in all the modes of operation
- If `TET_OUTPUT_CAPTURE` is true:
  - the test case or tool is executed with standard output and standard error redirected to a file
  - when the process exits, the captured output is copied to the journal

Common processing for test cases and tools (cont'd)

- If `TET_API_COMPLIANT` is false:
  - `tcc` generates the execution results lines that would be output by an API-conforming test case
- If `TET_API_COMPLIANT` is true:
  - `tcc` copies the execution results file generated by the API to the journal
  - For a non-distributed test case, test case information lines are reordered so that lines with the same context and block numbers are kept together in the journal

Build mode processing

- This processing is performed in the test case source directory
- Obtain exclusive locks
- Execute the optional prebuild tool
  - If it fails, skip the build stage and execute the optional build fail tool
- Execute the build tool
  - If it fails, execute the optional build fail tool
- Remove locks

THE Open GROUP

Feb-05 TETware training course 3-43

Execute mode processing (1)

- Obtain locks in:
  - the source directory (always)
  - the execution directory (if specified)
    - shared locks if `TET_EXEC_IN_PLACE` is false
    - exclusive locks if `TET_EXEC_IN_PLACE` is true
- If `TET_EXEC_IN_PLACE` is false:
  - copy the directory containing the test case (execution or source) to the temporary directory
  - remove the locks

THE Open GROUP

Feb-05 TETware training course 3-44

Execute mode processing (2)

- Execute the test case (or exec tool if one has been specified)
  - If `TET_EXEC_IN_PLACE` is false:
    - below the temporary directory
  - If `TET_EXEC_IN_PLACE` is true:
    - below the alternate execution directory if one is specified
    - otherwise in the test case source directory

THE Open GROUP

Feb-05 TETware training course 3-45

Execute mode processing (3)

- Perform **saved files** processing
- If `TET_EXEC_IN_PLACE` is false:
  - remove the temporary directory subtree created earlier
- If `TET_EXEC_IN_PLACE` is true:
  - remove locks

THE Open GROUP

Feb-05 TETware training course 3-46

Clean mode processing

- This processing is performed in the test case source directory
- Obtain exclusive locks
- Execute the clean tool
- Remove locks

Feb-05 TETware training course THE Open GROUP 3-47

3.6 Saved files processing

- This processing is performed after `tcc` executes a test case
- If `TET_EXEC_IN_PLACE` is false, files generated by a test case are lost when `tcc` removes the temporary directory subtree
- If it is required to keep any of these files, `tcc` can be instructed to save the files before the temporary directory is removed

Feb-05 TETware training course THE Open GROUP 3-48

TET_SAVE_FILES

- The TET_SAVE_FILES variable in the execute mode configuration tells tcc which files to save
- It should be set to a comma-separated list of file and directory names
- Shell file name syntax can be used
- If a directory is specified, it is saved recursively
- For example:

```
TET_SAVE_FILES=core,tmp[0-9]*
```

THE Open GROUP

Feb-05 TETware training course 3-49

The saved files directory

- Files are saved in a directory subtree below the results directory:

  test-suite-root/results/*NNNN*[bec]/...

THE Open GROUP

Feb-05 TETware training course 3-50

Saved files processing in Distributed TETware

- Files specified by `TET_SAVE_FILES` on a remote system can be:
  - saved below
    `test-suite-root``/results/`*`NNNN`*[bec]
    on the remote system
  - transferred to the local system and saved below
    `test-suite-root``/results/`*`NNNN`*[bec]`/REMOTE`*`nnn`*
- The value of `TET_TRANSFER_SAVE_FILES` determines which option is performed

THE Open GROUP

Feb-05 TETware training course 3-51

3.7 Execution results processing

- API-conforming test cases and tools
- Non API-conforming test cases and tools

- TETware-Lite
- Distributed TETware

THE Open GROUP

Feb-05 TETware training course 3-52

Execution results from an API-conforming test case

- The API writes execution results to a file called `tet_xres` in the test case execution directory
- `tcc` copies the contents of this file to the journal
- When copying test case information lines, `tcc` reorders them so that lines with the same context and block number are kept together
  - this is done so that output from a child process is kept separate from output from the parent

Execution results from a non API-conforming test case

- A non API-conforming test case or tool should be executed with output capture enabled
- `tcc` copies the captured output to the journal
- When processing a non API-conforming test case in execute mode, `tcc` generates the result code from the process exit code:
  - zero = PASS
  - non-zero = FAIL

Execution results processing in Distributed TETware

- The Distributed C, C++ and Java APIs send execution results to `tetxresd`
- `tcc` checks to see if `tetxresd` has been used:
  - if it has, `tcc` copies lines from `tetxresd` to the journal
  - otherwise, `tcc` copies lines from the `tet_xres` file to the journal
- `tcc` does not reorder information lines from a distributed test case

THE Open GROUP

Feb-05 TETware training course 3-55

3.8 Rerun and Resume processing

- You can ask `tcc` to rerun selected test cases from a previous test run
- You can ask `tcc` to resume processing of a previous test run from a certain point
- When `tcc` does this it uses:
  - the journal file from the previous run
  - a (comma-separated) list of test case selectors that you specify

THE Open GROUP

Feb-05 TETware training course 3-56

Test case selectors

- The test case selector is one or more of:
  - result code names - `PASS`, `FAIL`, `UNRESOLVED` etc.
  - modes of operation - b, e and c
- In Rerun mode only test cases identified by the test case selector is processed
- In Resume mode processing of the scenario is resumed from the first test case identified by the test case selector

Feb-05 TETware training course THE Open GROUP 3-57

Test case selectors (cont'd)

- When a result code name is specified, it matches a Test Purpose Result containing the corresponding result code number
  - a result code name does not match a result generated by an API-conforming build or clean tool
- When a mode of operation is specified, it matches a Result line containing a non-zero result code or an End line containing a non-zero completion status

Feb-05 TETware training course THE Open GROUP 3-58

Rerun example

- To rebuild all test cases that previously failed to build:

  ```
  tcc -b -r b ... old-journal-file ...
  ```

rerun mode (-r)

select test cases that failed in this mode (b)

Resume example

- To resume executing a scenario at the point where the first test purpose failed or was unresolved:

  ```
  tcc -e -m FAIL,UNRESOLVED ... old-journal-file ...
  ```

resume mode (-m)

select first test case with these result codes (FAIL,UNRESOLVED)

3.9 Invocation

- There are lots of options that can be specified on the `tcc` command line
  - see the `tcc` manual page at the back of the TETware User Guide for details
- You must use at least one of `-b`, `-e` and `-c` to specify which mode(s) of operation you want
- You can also specify a test suite name and a scenario name; for example:

```
tcc -b vsx4 miniscen
```

THE Open GROUP

Feb-05 TETware training course 3-61

Test suite name

- The test suite name is the name of the test suite root directory
- If you don't specify a test suite name, `tcc` takes the current directory and identifies the first component below `$TET_ROOT`
- If the current directory is not below `$TET_ROOT`, `tcc` can't determine the test suite name

THE Open GROUP

Feb-05 TETware training course 3-62

Scenario name

- If you specify a test suite name, you can also specify a scenario name
- If you don't specify a scenario name, it defaults to `all`

Some useful command-line options

- `-p` reports progress - useful to see what is happening during a long test run
- You can specify configuration variables with `-v`
- You can select or reject particular test cases from the scenario with `-y` and `-n`
- You can specify a simple scenario with `-l`
- You can specify a journal file with `-j`

3.10 Test session interruption

- `tcc` interprets signals as follows:
  - `SIGINT`
    - — interrupts the currently executing test case or tool
  - `SIGQUIT` (on UNIX systems)
  - `SIGBREAK` (on Win32 systems)
    - — interrupts the whole test run
- These signals can be generated from the keyboard

THE Open GROUP

Feb-05 TETware training course 3-65

Keyboard interrupts

- `SIGINT`
  - usually Control-C on UNIX systems
  - always Control-C on Win32 systems
- `SIGQUIT`
  - usually Control-\ on UNIX systems
- `SIGBREAK`
  - always Control-BREAK on Win32 systems

THE Open GROUP

Feb-05 TETware training course 3-66

Warning for Win32 systems

- On a Win32 system use of keyboard interrupts may result in unpredictable system behaviour
  - depending on what the test case is doing, you may need to reboot the machine!
- So only use this facility as a last resort on Win32 systems

Exercise 3

Module 4

Customising TETware

Module 4 - Customising TETware

- 4.1 Processing single test cases during the test development cycle
- 4.2 ETET and dTET2 compatibility
- 4.3 Interpreting TETware diagnostics
- 4.4 Using debugging tools
- 4.5 TETware trace debugging
- 4.6 Handling unexpected events
- 4.7 Interacting with test cases in Distributed TETware

4.1 Processing single test cases during the test development cycle

- When test cases are being developed, it is useful to be able to process a single test case
- There are several ways to instruct `tcc` to execute just one test case; for example:
  - specify a mini-scenario using `-l`
  - select or reject scenario lines using `-y` and `-n`

THE Open GROUP

Specify a mini-scenario using `-l`

- For example:

```
tcc -e -l /ts/tc1 ...
```

scenario line (points to: `-l /ts/tc1`)

- When this is done, the default scenario file is not used
- You can specify more than one `-l` option if necessary

THE Open GROUP

Select or reject scenario lines using -y and -n

- For example, to process only the test case whose name contains a particular string:

  `tcc -e -y print ...`

  select a string (→ `-y print`)

- Only scenario lines containing the specified string are processed

Example shell script to execute a single test case

```
# execute a test case called /ts/foo/T.foo in the vsx4 test suite
# invoke in the test case source directory

TESTNAME=`echo $PWD | sed -e "s^${TET_ROOT:?}/vsx4^^"`/T.`basename $PWD`
ICLIST=
DBUG_ARG=

while test $# -gt 0
do
     case "$1" in
     -d)
         DBUG_OUT=${PWD}/dbug.out
         DBUG_OUT_L=${PWD}/dbug.out_l
         DBUG_ARG="-v VSX_DBUG_FLAGS=d:f:l,2:P:p:t:F:L \
              -v VSX_DBUG_FILE=$DBUG_OUT \
              -v TET_REM002_VSX_DBUG_FILE=$DBUG_OUT_L"
         rm -f $DBUG_OUT $DBUG_OUT_L
         ;;
     -*)
         echo $0: unknown option: $1 1>&2
         exit 2
         ;;
```

Example shell script (cont'd)

```
     [0-9]*)
         if test -z "$ICLIST"
         then
              ICLIST=$1
         elif test "$ICLIST" != all
         then
              ICLIST="$ICLIST,$1"
         fi
         ;;
     all)
         ICLIST=$1
         ;;
     *)
         echo $0: ignored \"$1\" 1>&2
         ;;
     esac
     shift
done
```


Example shell script (cont'd 2)

```
if test -z "$ICLIST"
then
     ICLIST=all
fi

echo $TESTNAME

rm -f j

tcc -ep -j j -l :remote,000,001,002:@$TESTNAME{$ICLIST} $DBUG_ARG
```

4.2 ETET and dTET2 compatibility

- Previous TET implementations have interpreted the `parallel` directive in different ways
- When the TETware `tcc` needs to know how it should interpret this directive, it uses the value of the `TET_COMPAT` configuration variable

Feb-05 TETware training course THE Open GROUP 4-9

TET_COMPAT

- Possible values are:
  - `etet` to select ETET behaviour
  - `dtet2` to select dTET2 behaviour
- There is no default value
- When defined, `TET_COMPAT` must have the same value in the configurations for each of the chosen modes of operation

Feb-05 TETware training course THE Open GROUP 4-10

4.3 Interpreting TETware diagnostics

- Diagnostic messages may be generated by:
  - `tcc`
  - The TCM/API
  - Distributed TETware servers:
    - `tccd`
    - `tetsyncd`
    - `tetxresd`

THE Open GROUP

Feb-05 TETware training course 4-11

What does a diagnostic message look like ?

- A free-format text message from `tcc`
  - user-level errors which report things like scenario syntax errors and configuration errors
- A structured message
  - system-level errors which report things like missing files and unexpected events
  - a structured message includes:
    - source file name and line number
    - message text
    - system error message (from `errno`) or server reply code

THE Open GROUP

Feb-05 TETware training course 4-12

A free-format text message from `tcc`

- For example:

```
tcc: unknown/unsupported directive
 remote,001 at line 20 in file
 c:/tet3lite/tests/bad_scen
tcc: giving up after finding 1 scenario
 error
```

Feb-05 TETware training course THE Open GROUP 4-13

Diagnostic with system error message

```
(tcfexec.c, 466): spawn failed,
    path = c:/tet3lite/tests/ts/tc21:
    No such file or directory
```

source file name and line number

system error message

message text

Feb-05 TETware training course THE Open GROUP 4-14

Diagnostic with server reply code

```
(exec.c, 129): can't exec
      c:/tet3lite/tests/ts/tc21,
      reply code = ER_NOENT
```

source file name and line number

server reply code

message text

Where do diagnostic messages appear?

- Diagnostics generated by `tcc`:
  - on the standard error stream
  - in the journal
    - — as Test Case Controller messages (code 50)
- Diagnostics generated by the TCM/API:
  - in the journal
    - — as Test Case Manager messages (code 510)

Where do diagnostic messages appear? (cont'd)

- Diagnostics generated by Distributed TETware servers:
  - `tetsyncd` and `tetxresd`
    - on the standard error stream
    - (inherited from `tcc`)
  - `tccd`
    - in the TCCD log file; defaults are:
      - `/tmp/tccdlog` (on UNIX systems)
      - `c:/tmp/tccdlog` (on Win32 systems)
    - on the console if the log file cannot be opened

THE Open GROUP

Feb-05 TETware training course 4-17

Identifying the source of a diagnostic

- In the journal
  - the journal line type identifies the source of the diagnostic
    - `tcc` generates line type 50
    - the TCM/API generates line type 510
- On the standard error stream
  - the diagnostic is preceded by the name of the program that generates it

THE Open GROUP

Feb-05 TETware training course 4-18

Server reply codes

- Distributed TETware uses a client-server architecture
- If a server request fails, the reason for the failure is returned in the **server reply code**
- A list of these codes is presented in Appendix E of the TETware User Guide
- Although TETware-Lite does not use a client-server architecture, many of these codes are also used in TETware-Lite diagnostics

ER_ERR - the general error code

- If a server reply code is ER_ERR, a more detailed message is printed by one of the lower software layers
- In Distributed TETware, the more detailed message is usually printed in the TCCD log file

4.5 Using debugging tools

- Using a debugger
- Preserving core files

Using a debugger

- In TETware, support for executing test cases under the control of a debugger is provided through the exec tool
- Normally a test suite does not specify an exec tool
  - in this case, `tcc` executes each test case directly
- When an exec tool is specified, `tcc` executes the exec tool each time a test case is to be processed in execute mode

The exec tool and exec file

- You can specify an exec tool using the TET_EXEC_TOOL variable in the execute mode configuration
- You can also specify the optional TET_EXEC_FILE variable in the execute mode configuration
- When an exec tool is specified, tcc executes the TET_EXEC_TOOL with TET_EXEC_FILE, the test case name and optional IC list as arguments
- How you use this feature depends on which debugger you want to use

THE Open GROUP

Feb-05 TETware training course 4-23

Preserving core files

- If a test case dumps core and TET_EXEC_IN_PLACE is false, the core file will be lost when the temporary directory is removed
- You can overcome this problem by including the name of the core file in TET_SAVE_FILES

THE Open GROUP

Feb-05 TETware training course 4-24

4.5 TETware trace debugging

- The TETware C source code includes extensive facilities for tracing the operation of TETware itself
- These facilities are provided mainly for use during TETware development
- They may also be of help when an experienced user needs to trace TETware operation
- Further details in Appendix G of the TETware User Guide

THE Open GROUP

Feb-05 TETware training course 4-25

Subsystems and trace flags

- Internally, TETware programs consist of one or more subsystems
- Each subsystem has a trace flag associated with it
- Each trace flag can be set to a value
  - more trace detail is reported for higher flag values
- Each trace flag may be propagated to other TETware processes

THE Open GROUP

Feb-05 TETware training course 4-26

Subsystems, flag names and indicators

Subsystem	Flag name	Flag indicator
Memory allocation	`tet_Tbuf`	`b`
`tcc` execution engine	`tet_Texec`	`g`
`tcc` scenario parser	`tet_Tscen`	`p`
Generic `tcc` operation	`tet_Ttcc`	`m`
TCM functions	`tet_Ttcm`	`c`
Trace subsystem	`tet_Ttrace`	`t`
DTET message i/o	`tet_Tio`	`i`
DTET client & server loops	`tet_Tloop`	`l`
DTET generic server	`tet_Tserv`	`s`
`tetsyncd` operation	`tet_Tsyncd`	`y`
`tccd` operation	`tet_Ttccd`	`s`
`tetxresd` operation	`tet_Txresd`	`x`

Process indicators

- You can use a process indicator to restrict tracing to certain processes
  - this facility is most useful when tracing Distributed TETware

Process indicator	Process description
`M`	`tcc`
`S`	`tccd`
`C`	master TCM/API
`D`	slave TCM/API
`X`	`tetxresd`
`Y`	`tetsyncd`
`T`	stand-alone programs

Setting trace flags

- Trace flags are specified using the `-T` command-line option
- They are usually set on the `tcc` command line
- They can also be set on the `tccd` command line if required
- If no process indicators are specified, the flag is set to the value in all processes
- If the flag indicator is `all`, all flags are set to the value

THE Open GROUP

Feb-05 TETware training course 4-29

Examples

- To trace processing of sync requests in `tetsyncd`:

```
tcc -TY,y8 ...
```

- To trace the operation of the scenario parser:

```
tcc -Tp6 ...
```

- To set all flags in all processes to the maximum value (definitely not recommended!):

```
tcc -Tall10 ...
```

THE Open GROUP

Feb-05 TETware training course 4-30

Some tips

- Be prepared to handle huge volumes of output
- Whenever possible, trace an example scenario rather than a fully-featured test suite
- Only turn on the smallest set of trace options
  - look at the source code to see what you get with each flag and value
- The trace output should be interpreted in conjunction with the TETware source code

4.6 Handling unexpected events

- On Unix systems, each TCM installs standard handlers for all the signals that can be caught before each test purpose is executed
- If a signal is caught during test purpose execution, control is returned to the TCM and an `UNRESOLVED` result is reported to the journal
- You can modify this behaviour by setting the `TET_SIG_IGN` and `TET_SIG_LEAVE` variables in the execute mode configuration
- The TCM does not permit this behaviour to be changed for POSIX signals

TET_SIG_IGN - list of signals to ignore

- You can set this variable to a (comma-separated) list of signals to be ignored
- The TCM will set these signals to be ignored instead of installing the standard handler
- For example, you would make the following assignment in order to ignore SIGXCPU on an SVR4 system:

```
TET_SIG_IGN=30
```


TET_SIG_LEAVE - list of signals to leave alone

- You can set this variable to a (comma-separated) list of signals to be left alone
- The TCM will not install handlers for these signals, so the default behaviour of these signals is left unchanged
- For example, you would make the following assignment in order to leave SIGTSTP and SIGCONT alone on an SVR4 system:

```
TET_SIG_LEAVE=24,25
```

4.7 Interacting with test cases in Distributed TETware

- In Distributed TETware, a test case is not a child of `tcc`; it does not have a controlling terminal and you can't interact with it
- If you need to interact with a test case you can use `tet_start` to run a test case in its own terminal window
  - On a UNIX system `tet_start` creates a new `xterm` window
  - On a Win32 system `tet_start` uses the MKS `start` command to create the new window

Feb-05 TETware training course THE Open GROUP 4-35

Using `tet_start`

- `tet_start` is an exec tool
- For example:

```
TET_EXPAND_CONF_VARS=true
TET_EXEC_TOOL=${TET_ROOT}/bin/tet_start
```

- You can use configuration variables to customise the behaviour of `tet_start`
- Instructions on how to use `tet_start` are presented in Chapter 8 of the TETware User Guide

Feb-05 TETware training course THE Open GROUP 4-36

Exercise 4

Module 5

The C API

Module 5 - The C API

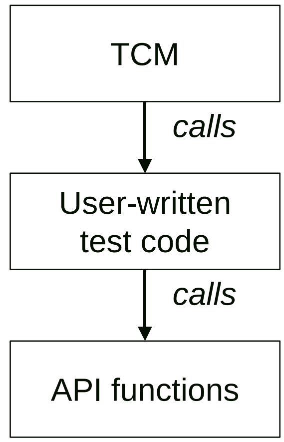
5.1 - Test case structure revisited

- We have already learned that:
  - each **test case** is an executable program
  - each test case contains one or more **invocable components**
  - each invocable component contains one or more **test purposes**
  - each test purpose tests a single item of functionality and generates a **result**
    - the result indicates whether or not the item conforms to the specification being tested

Test case structure (cont'd)

- We have already learned that when a test case is built, it is linked with:
  - the Test Case Manager (TCM)
  - the Application Program Interface (API) library

Relationship between TCM, user-written code and the API



THE Open GROUP

Feb-05 TETware training course 5-5

What is provided by TETware

- The Test Case Manager (TCM)
- The API library

THE Open GROUP

Feb-05 TETware training course 5-6

What the test case author must provide

- Required:
  - one or more test purpose functions
- Optional:
  - test case startup function
  - test case cleanup function

Test purpose functions

- Each test purpose function:
  - tests an item of functionality
  - should generate a result
- The TCM calls each function in turn

Startup and cleanup functions

- These functions:
  - are optional
  - should not perform any testing operations
  - should not generate results
- If you define a startup function, the TCM calls the function before it calls the first test purpose function
- If you define a cleanup function, the TCM calls the function after it calls the last test purpose function

5.2 - The `tet_api.h` file

- The functions, data items and constants that make up the C API are declared in the file *`tet-root`*`/inc/tet3/tet_api.h`
- You should include this header file in each test case source file

5.3 - Interface to user-written test code

- You can use one of two interfaces between the TCM and your test code:
  - The static interface
  - The dynamic interface
- Most test cases will use the static interface, so that is what we will describe here

THE Open GROUP

Feb-05 TETware training course 5-11

The static test case interface

- You tell the TCM the names of your test purpose function by defining an array called `tet_testlist`
- You tell the TCM the names of your startup and cleanup functions by defining variables called `tet_startup` and `tet_cleanup`
- These are the only symbols that are required to exist in your code

THE Open GROUP

Feb-05 TETware training course 5-12

The `tet_testlist[]` array

- This is an array of `tet_testlist` structures
- Each element describes a test purpose function that is to be called by the TCM
- The structure of each element is defined as follows:

```
struct tet_testlist {
  void (*testfunc)(void);  /* ptr to TP function */
  int icref;               /* IC number */
};
```

- You should set `testfunc` to the address of the test purpose function that the TCM should call
  - the array is terminated by a NULL value for `testfunc`
- You should set `icref` to the IC number that this test purpose belongs to
  - invocable components are numbered sequentially from 1

THE Open GROUP

Feb-05 TETware training course 5-13

`(*tet_startup)()` and `(*tet_cleanup)()`

- These variables are defined as follows:

```
void (*tet_startup)(void);
void (*tet_cleanup)(void);
```

- If your test case contains a startup and/or cleanup function, you should set these variables accordingly
- If your test case doesn't contain one of these functions, the corresponding variable should be set to NULL

THE Open GROUP

Feb-05 TETware training course 5-14

Example test case structure

- This example test case contains three test purpose functions, a startup function but no cleanup function:

```
#include <stdio.h>
#include "tet_api.h"

static void tp1(), tp2(), tp3();
static void prepare_tests();

struct tet_testlist tet_testlist[] = {
   { tp1, 1 },
   { tp2, 2 },
   { tp3, 3 },
   { NULL, 0 }
};

void (*tet_startup)() = prepare_tests;
void (*tet_cleanup)() = NULL;
```

THE Open GROUP

Feb-05 TETware training course 5-15

5.4 - API overview

- The API provides some functions and some global variables
- All the global symbols in the TCM and API start with the prefix `tet_`
  - so names starting with this prefix are reserved
- All the defined constants in `tet_api.h` start with the prefix TET_
  - so names starting with this prefix are reserved as well

THE Open GROUP

Feb-05 TETware training course 5-16

API functions

- There are API functions to:
  - make journal entries
  - cancel test purposes
  - access configuration variables
  - execute child processes
- In Distributed TETware the API also includes functions to:
  - synchronise parts of a distributed test case
  - obtain remote system information
  - execute remote processes

THE Open GROUP

Feb-05 TETware training course 5-17

API global variables

- `char *tet_pname;`
  - the test case name
- `int tet_thistest;`
  - the current test purpose number
- `int tet_nosigreset;`
  - enables you to alter the TCM's default signal handling
- `pid_t tet_child;`
  - child process ID after a fork
- `int tet_errno;`
  - last API error code
- `char *tet_errlist[];`
  - list of API error strings
- `int tet_nerr;`
  - number of strings in `tet_errlist[]`

THE Open GROUP

Feb-05 TETware training course 5-18

5.5 - API functions

- Making journal entries
- Canceling test purposes
- Accessing configuration variables
- Generating and executing processes

THE Open GROUP

Feb-05 TETware training course 5-19

Making journal entries

- Writing test case information lines
- Generating a test purpose result
- Changing the journal context and block number

THE Open GROUP

Feb-05 TETware training course 5-20

Writing test case information lines

- `tet_infoline(char *line)`
  - write a single test case information line
- `tet_minfoline(char **lines, int nlines)`
  - write multiple test case information lines
    - mainly used in distributed test cases
- `tet_printf(char *format, ...)`
- `tet_vprintf(char *format, va_list ap)`
  - write a formatted test case information line

THE Open GROUP

Feb-05 TETware training course 5-21

Writing test case information lines (cont'd)

- For example:

```
tet_infoline("an error has occurred");
```

- or:

```
tet_printf("can't open %s", fname);
```

THE Open GROUP

Feb-05 TETware training course 5-22

Generating a test purpose result

- `tet_result(int result)`
  - writes a result to the journal
- The standard result codes are defined in `tet_api.h`
- For example:

```
tet_result(TET_PASS);
```


Changing the journal context and block number

- `tet_setcontext(void)`
  - sets a new infoline context in the journal
- `tet_setblock(void)`
  - starts a new block of infolines in the journal
- The API calls these functions for you as required
- You don't normally need to call these functions yourself

Canceling test purposes

- `tet_delete(int testno, char *reason)`
  - cancels or reactivates a test purpose
    - — the TCM does not call a canceled test purpose function
    - — to cancel a test purpose, you specify a test number and a string
      - - the string should describe why the test purpose is to be cancelled
    - — to reactivate a canceled test purpose, you specify a test number and a NULL reason string
- `char *tet_reason(int testno)`
  - returns the reason string which describes why the test purpose has been canceled
    - — or returns NULL if the test purpose is active

Accessing configuration variables

- `char *tet_getvar(char *name)`
  - returns the value of a variable in the configuration for the current mode of operation
  - returns NULL if the variable has not been defined

Generating and executing processes

- Only on UNIX systems:
  - `tet_fork()` and `tet_exec()`
- On both UNIX and Win32 systems:
  - `tet_spawn()`, `tet_wait()` and `tet_kill()`

`tet_fork()`

- ```
  int tet_fork(
    void (*childproc)(void),
    void (*parentproc)(void),
    int waittime, int validresults)
  ```
- The API forks and calls the `childproc` function in the child process
- If `parentproc` is non-NULL, the API calls the `parentproc` function in the parent process
- Then the API waits for the child process to exit
  ```

`tet_fork()` - child exit status processing

- If the child process exits normally, the API clears bits in the child's exit status that are set in `validresults`
  - that is: `tmp = (`*`child-exit-status`*` & ~validresults);`
- If the result of this operation is zero, `tet_fork()` returns the child's exit status
- If the result of this operation is non-zero or the child process is terminated by a signal:
  - the API prints a message to the journal and generates a result of `UNRESOLVED`
  - `tet_fork()` returns -1

`tet_fork()` - timing out a child process

- If `waittime` is +ve, the API waits for up to `waittime` seconds for the child process to exit after the `parentproc` function returns
  - if the timeout expires, the API terminates the child process
- If `waittime` is zero, the API waits indefinitely for the child process to exit
- If `waittime` is -ve, the API ignores `validresults` and does not wait for the child process at all
  - in this case it is the responsibility of the `parentproc` function to wait for the child process to exit and interpret the child's exit status

`tet_exec()`

- `int tet_exec(char *file,`
  `char *argv[], char *envp[])`
- This function executes a subprogram that will use the API
  - the subprogram must be linked with a child process controller (e.g.: `tcmchild.o`)
- This function may be called from the `childproc` function in the child process that is generated by a call to `tet_fork()`

THE Open GROUP

Feb-05 TETware training course 5-31

`tet_spawn()`, `tet_wait()` and `tet_kill()`

- `pid_t tet_spawn(char *file,`
  `char *argv[], char *envp[])`
  - execute a subprogram that will use the API
- `tet_wait(pid_t pid, int *statloc)`
  - wait for a subprogram started by `tet_spawn()` to terminate
- `tet_kill(pid_t pid, int sig)`
  - terminate a subprogram started by `tet_spawn()`

THE Open GROUP

Feb-05 TETware training course 5-32

5.6 - Child processes and subprograms

- A subprogram started by `tet_exec()` or `tet_spawn()` must be linked with the child process controller (e.g.: `tcmchild.o`)
- The child process controller calls a function called `tet_main()` which you must provide

THE Open GROUP

Feb-05 TETware training course 5-33

Child processes and subprograms (cont'd)

- A subprogram:
  - can call most of the API functions
  - can generate a test purpose result
- The API sets the following global variables in a subprogram:
  - `int tet_thistest;`
    - the current test number
  - `char *tet_pname;`
    - the name of the subprogram

THE Open GROUP

Feb-05 TETware training course 5-34

`tet_main()` - the subprogram entry point

- `int tet_main(int argc, char *argv[])`
  - this is a function that you must supply
  - when the child process controller starts up, it calls this function
  - `argc` and `argv` refer to the arguments that you passed to the `tet_exec()` or `tet_spawn()` call that started this subprogram
  - if your `tet_main()` function returns, its return value provides the subprogram's exit status

Exiting from a child process or subprogram

- `tet_exit(int status)`
  - you call this function to exit from a child process or subprogram that uses the API

5.7 - Error reporting

- If an API function returns an error value, it sets the global variable `tet_errno` to indicate the cause of the error
- The file `tet_api.h` contains definitions for all the TETware error codes

Feb-05 TETware training course THE Open GROUP 5-37

5.8 - The Thread-safe API

- If your test case uses threads, it must use the Thread-safe version of the API
- The API needs to know what threads are running in the test case, so you must create a new thread by using the appropriate API function
- There are some issues that you should be aware of when writing a test case that uses threads
  - see Sections 10.6 through 10.9 of the TETware Programmers Guide for details

Feb-05 TETware training course THE Open GROUP 5-38

Creating a new thread

- UNIX systems:
  - Unix International threads (UI threads)
    - `tet_thr_create()`
  - POSIX threads (pthreads)
    - `tet_pthread_create()`
- Win32 systems
    - `tet_beginthreadex()`

Feb-05 TETware training course THE Open GROUP 5-39

Waiting for a thread to exit

- UNIX systems:
  - Unix International threads (UI threads)
    - `tet_thr_join()`
  - POSIX threads (pthreads)
    - `tet_pthread_join()`
- Win32 systems
    - no API function - use `WaitForSingleObject()` and `CloseHandle()` instead

Feb-05 TETware training course THE Open GROUP 5-40

Detaching a thread

- UNIX systems:
  - Unix International threads (UI threads)
    - no functionality to do this
  - POSIX threads (pthreads)
    - `tet_pthread_detach()`
- Win32 systems
    - no functionality to do this

THE Open GROUP

Compiling a test case which uses the thread-safe C API

- UNIX systems:
  - UI threads:
    - `cc -DTET_THREADS` ...
  - POSIX threads:
    - `cc -DTET_POSIX_THREADS` ...
- Win32 systems (using the defined build environment):
    - `cc -MD -DTET_THREADS` ...

THE Open GROUP

5.9 - Linking a test case with the C API

- The “standard” C API:
  - main programs
    - `tcm.o` and `libapi.a`
  - subprograms
    - `tcmchild.o` and `libapi.a`
- The Thread-safe C API:
  - main programs
    - `thrtcm.o` and `libthrapi.a`
  - subprograms
    - `thrtcmchild.o` and `libthrapi.a`

THE Open GROUP

Feb-05 TETware training course 5-43

5.10 - Example C test case

```
#include <stdlib.h>
#include "tet_api.h"

void (*tet_startup)() = NULL, (*tet_cleanup)() = NULL;
void tp1();

struct tet_testlist tet_testlist[] = { {tp1,1}, {NULL,0} };

void tp1()
{
        tet_infoline("This is the first test case (tc1)");
        tet_result(TET_PASS);
}
```

THE Open GROUP

Feb-05 TETware training course 5-44

Exercise 5

Module 6

Test case development techniques

Module 6 - Test case development techniques

- 6.1 - Introduction
- 6.2 - Test suite structure
- 6.3 - Some useful guidelines
- 6.4 - Result codes
- 6.5 - Use of configuration variables
- 6.6 - Testing of optional features
- 6.7 - Reporting results
- 6.8 - Child processes and subprograms

6.1 - Introduction

- In this module we will discuss some of the things that you will need to consider when writing a test suite
- This module is not intended to be a comprehensive tutorial on how to write tests
    - this would be the subject of a course in itself!

6.2 - Test suite structure

- We have already learned that:
    - a test suite includes:
        - test case files
        - configuration files
        - a scenario file
        - build and clean tools
            - if the test suite is to be supplied in source form
    - all the files in a test case reside below the **test suite root** directory

Directory layout

- Set up a directory sub-tree which contains all the test case files
  - for example, in VSX all the test case files are below a directory called `tset`
  - in a large test suite, group the test cases by area
- Put the files for each test case in their own directory
  - source files
  - makefile
  - any data files that are used by the test case

THE Open GROUP

Feb-05 TETware training course 6-5

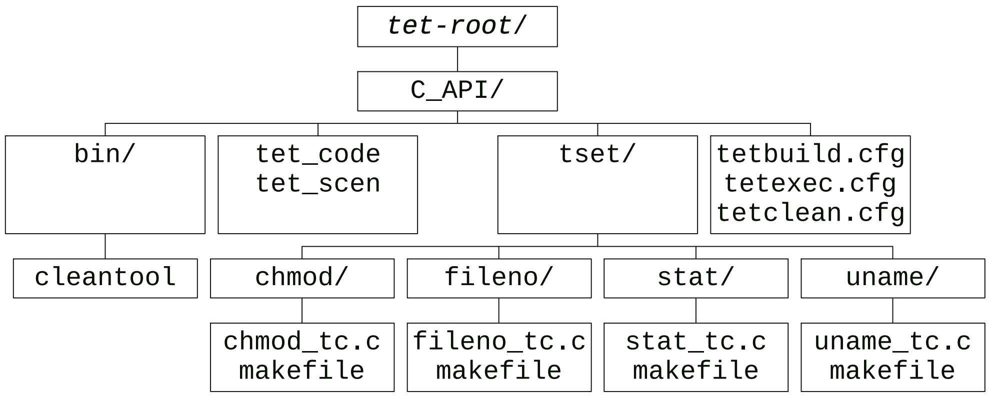
Test case layout

- Put all the tests for a particular functional element in a single test case
  - positive (or compliance) tests
  - negative (or deviance) tests
- For example, in VSX all the tests for the `fclose()` function are in a single test case

THE Open GROUP

Feb-05 TETware training course 6-6

Structure of the example C-API test suite



6.3 - Some useful guidelines

- Each test case should be self-contained
  - it should not rely on being executed before or after another test case
- Each test purpose should be self-contained
  - it should not rely on a setup operation having been performed by a previous test purpose
  - if you **must** rely on a previous test purpose having been executed, group your test purposes together into a single invocable component

Some useful guidelines (cont'd)

- Each test purpose should leave the system in the state it was in before the test started
  - for example: if you create a file, you should remove it before the test purpose ends
- Create any required files in the test case execution directory if possible
- Don't assume that a previous instance of a test purpose cleaned up successfully
  - for example: if you create a file, you should unlink an existing file first

6.4 - Result codes

- We have already learned about the standard result codes:
  - `PASS`
  - `FAIL`
  - `UNRESOLVED`
  - `NOTINUSE`
  - `UNSUPPORTED`
  - `UNTESTED`
  - `UNINITIATED`
  - `NORESULT`

Standard result codes - PASS and FAIL

- PASS
  - The element under test behaved in the way required by the specification
- FAIL
  - The element under test did not behave in the way required by the specification

Standard result code - UNRESOLVED

- Some problem occurred when preparing to test the element
  - for example: in a test for `read()`, the file must be opened before it can be read
  - if the `open()` fails, the result is `UNRESOLVED`
- This result code can also be used to indicate a test suite configuration problem
  - for example, if a test uses a configuration variable and the variable is not defined, the result is `UNRESOLVED`

Standard result code - `NOTINUSE`

- Used when a test is not to be performed for some reason
  - for example: when VSX is run in XPG3 mode, tests that are only applicable to XPG4 are reported as `NOTINUSE`

Feb-05 TETware training course THE Open GROUP 6-13

Standard result code - `UNSUPPORTED`

- Used when an optional feature is not supported by the system under test
  - when POSIX-style assertions are being tested, this corresponds to an **If** clause in the assertion
  - for example:

    If modem control is supported:
    A call to `open()` on a terminal device does not return until the carrier detect line is asserted
- In this case:
  - support for modem control would be indicated by a configuration variable
  - if the variable is `True`, the test is performed
  - if the variable is `False`, the test is not performed but reports `UNSUPPORTED` instead

Feb-05 TETware training course THE Open GROUP 6-14

Standard result code - UNTESTED

- Used to indicated that an extended assertion could not be tested fully
  - for example: if there is no practical limit to the number of files that a process can open, the `open()` test for the `EMFILE` error would report `UNTESTED`
  - often, it is appropriate to use `UNTESTED` instead of `PASS` when testing an extended assertion

Standard result codes - UNINITIATED and NORESULT

- These codes are provided for use by TETware
- `UNINITIATED`
  - used to indicate that a test purpose has been canceled by a previous call to `tet_delete()`
- `NORESULT`
  - used to indicate that a test purpose did not generate a result

6.5 - Use of configuration variables

- You can use configuration variables to pass parameters to a test case
- Often they are used to:
  - specify a system-dependent quantity
  - say whether or not the system under test supports a particular optional feature

THE Open GROUP

Feb-05 TETware training course 6-17

6.6 - Testing of optional features

- Sometimes a specification defines an optional feature
- The specification does not **require** the feature to be implemented
  - but if it **is** implemented it must conform to the specification
- In this case you should define a configuration variable which says whether or not the system supports the optional feature
  - the variable is set as appropriate on each system under test
  - the test reports `PASS` or `FAIL` if the feature is supported, or `UNSUPPORTED` if the feature is not supported

THE Open GROUP

Feb-05 TETware training course 6-18

Example of testing an optional feature

```
if ((val = tet_getvar("TS_MODEM_CONTROL_SUPPORTED")) == (char *) 0) {
   tet_infoline("parameter TS_MODEM_CONTROL_SUPPORTED is not set");
   tet_result(TET_UNRESOLVED);
   return;
}

switch (*val) {
case 'Y':
case 'y':
   break;
case 'N':
case 'n':
   tet_infoline("modem control is not supported");
   tet_result(TET_UNSUPPORTED);
   return;
default:
   tet_infoline("parameter TS_MODEM_CONTROL_SUPPORTED has an invalid value");
   tet_result(TET_UNRESOLVED);
   return;
}

/* rest of test ... */
```


6.7 - Reporting results

- When a test reports `FAIL` or `UNRESOLVED`, it should provide information describing what went wrong
- The message should provide as much useful information as possible
- If a test fails, you should say what you **expected** and what you **observed**
- Take time to get this right - a test suite is only as useful as the information it generates!

Some unhelpful messages

- `an error has occurred`
- `can't create file`
- `setup failed`

Some useful messages

- When a test reports `UNRESOLVED`:
  - `can't open ./t31file for reading: errno = ENOENT (No such file or directory)`
  - `caught unexpected signal 11 (SIGSEGV)`
- When a test reports `FAIL`:
  - `read() did not return expected values: expected 256, observed -1 with errno set to EINTR`

Results that must be verified by the user

- Some results can't easily be verified by a test purpose function
  - for example: the host name and OS version number returned by `uname()`
- In this case you should:
  - print the information to be verified in the journal
  - use a user-defined result code which means “this information must be inspected by a person”

6.8 - Child processes and subprograms

- Child process
  - generated by a call to `tet_fork()` on UNIX systems
  - there is no equivalent on Win32 systems
- Subprogram
  - a separate program that uses the API
    - so must be linked with the child process controller
  - on UNIX systems
    - started by a call to `tet_exec()` from a child process
  - on UNIX and Win32 systems
    - started by a call to `tet_spawn()`

Child processes

- The program environment includes things like:
  - current working directory
  - `umask`, `ulimit`, `nice` value, etc.
  - process group ID
  - disposition of signals
  - environment string values
- If a test purpose function modifies the program environment, it should be put in a child process
  - it is easy to do this by calling `tet_fork()` with no `parentproc` function

THE Open GROUP

Feb-05 TETware training course 6-25

Simple `tet_fork()` example

```
static void test3()
{
   (void) tet_fork(ch_t3, TET_NULLFP, 30, 0)
}

static void ch_t3()
{
   /* perform test here and generate a result */
}
```

THE Open GROUP

Feb-05 TETware training course 6-26

Subprograms

- If a test case needs to run with a different user ID, it must be put in a subprogram which has the set-UID bit set
  - you can launch a subprogram from a child process using `tet_exec()`, or make a call to `tet_spawn()` followed by a call to `tet_wait()`
- The subprogram must contain a `tet_main()` function and be linked with the child process controller (e.g.: `tcmchild.o`)

THE Open GROUP

Example test case using `tet_fork()` and `tet_exec()`

```
#include "tet_api.h"
static void test5()
{
    (void) tet_fork(ch_t5, TET_NULLFP, 30, 0);
}

static void ch_t5()
{
    static char *argv[] = { "./foo-t5", (char *) 0) };

    (void) tet_exec(argv[0], argv, environ);
    tet_printf("tet_exec(%s) failed, tet_errno = %d", argv[0],
      tet_errno);
    tet_result(TET_UNRESOLVED);
}
```

THE Open GROUP

Example test case using tet_spawn() and tet_wait()

```
#include "tet_api.h"
static void test5()
{
      static char *argv[] = { "./foo-t5", (char *) 0) };
      pid_t pid;
      int status = 0;

      if ((pid = tet_spawn(argv[0], argv, environ)) == -1) {
          tet_printf("tet_spawn(%s) failed, tet_errno = %d", argv[0], tet_errno);
          tet_result(TET_UNRESOLVED);
          return;
      }

      if (tet_wait(pid, &status) == -1) {
          tet_printf("tet_wait() failed, tet_errno = %d", tet_errno);
          tet_result(TET_UNRESOLVED);
      }
      else if (status != 0) {
          tet_printf("%s terminated with unexpected status %d", argv[0], status);
          tet_result(TET_UNRESOLVED);
      }
}
```


Example subprogram

```
#include "tet_api.h"

int tet_main(argc, argv)
int argc;
char **argv;
{
      int euid = geteuid();

      if (euid != expected-euid) {
          tet_printf("expected effective UID %d, observed %d", expected-
              euid, euid);
          tet_result(TET_UNRESOLVED);
          return(0);
      }

      /* perform test here and generate a result */

      return(0);
}
```

Exercise 6

Module 7

Other APIs

Module 7 - Other APIs

- 7.1 - The C++ API
- 7.2 - The Shell , POSIX Shell and Korn Shell APIs
- 7.3 - The Perl API

7.1 - The C++ API

- The C++ API is a “lightweight” API because it mostly uses the C API
  - only the TCM is different
  - the API library is the same
- There is also a Thread-safe version of the C++ API

Interface to the user-written test code

- This is the same as when the C API is used
- However, the interface variables must be enclosed in an `extern "C"` block
- For example:

```
extern "C" {
    struct tet_testlist tet_testlist[] = {
        ...
    };
    void (*tet_startup)() = ...
    void (*tet_cleanup() = ...
}
```

The `tet_api.h` file

- As with the C API, you should include the `tet_api.h` header file in each test case source file
- When this file is compiled by a C++ compiler, the contents are placed within an `extern "C"` block

Feb-05 TETware training course THE Open GROUP 7-5

Linking a test case with the C++ API

- The “standard” C++ API:
  - main programs
    - `Ctcm.o` and `libapi.a`
  - subprograms
    - `Ctcmchild.o` and `libapi.a`
- The Thread-safe C++ API:
  - main programs
    - `Cthrtcm.o` and `libthrapi.a`
  - subprograms
    - `Cthrtcmchild.o` and `libthrapi.a`

Feb-05 TETware training course THE Open GROUP 7-6

Example C++ test case

```
#include <iostream.h>
#include <stdlib.h>
#include "tet_api.h"


static void startup()
{
// startup routine called before the test purposes

        cout << "This is from the startup routine for tp1\n";
        cout << "Its interesting to note that C++ output\n";
        cout << "using cout does not get journaled.\n";
        cout << "Test 2 will explain why!.\n";
}

void tp1();

extern "C" {
        void (*tet_startup)() = startup, (*tet_cleanup)() = NULL;
        struct tet_testlist tet_testlist[] = { {tp1,1}, {NULL,0} };
}

void tp1()
{
        cout << "This output comes from test purpose tp1\n";
        tet_infoline("This is the first test case (tc1)");
        tet_result(TET_PASS);
}
```


7.2 - The Shell, POSIX Shell and Korn Shell APIs

- Structure of a shell test case
- Interface to the user-written test code
- Description of API functions
- API library files
- Child processes and subprograms

Structure of a shell test case

- A Shell test case consists of a set of shell functions which you provide
- You should put the code for each test purpose in a separate function
- Your test case should only contain functions - not directly executed commands

Feb-05 TETware training course 7-9

Interface to the user-written test code

- You tell the TCM about the invocable component names in your test case by defining a variable called `iclist`
- You tell the TCM the names of your startup and cleanup functions by defining variables called `tet_startup` and `tet_cleanup`

Feb-05 TETware training course 7-10

`iclist` - list of invocable component names

- You should set `iclist` to a (blank-separated) list of invocable component names
- Each of these names should have the prefix `ic` followed by the invocable component number
- For example:

```
iclist="ic1 ic2 ic3"
```

- Then you should define a variable for each invocable component and set it to the names of the IC's test purpose functions
- For example:

```
ic1="test1"
ic2="test2"
ic3="test3 test4"
```

THE Open GROUP

Feb-05 TETware training course 7-11

`tet_startup` and `tet_cleanup`

- You should set these variables to the names of your startup and cleanup functions
- If your test case doesn't use one of these functions you should set the corresponding variable to an empty string

THE Open GROUP

Feb-05 TETware training course 7-12

Description of API functions

- These functions are equivalent to the corresponding functions in the C API:
  - making journal entries
  - canceling test purposes
- You can use Shell mechanisms for:
  - accessing configuration variables
  - generating and executing processes

Making journal entries

- Writing test case information lines
- Generating a test purpose result
- Changing the journal context and block number

Writing test case information lines

- `tet_infoline` *`data`* `...`
  - writes a test case information line to the journal

Generating a test purpose result

- `tet_result` *`result-name`*
  - writes a result to the journal
  - *`result-name`* should be the name of a result code; for example PASS or `FAIL`

Changing the journal context and block number

- `tet_setcontext`
  - establishes a new journal context for test case information lines
- `tet_setblock`
  - starts a new block of test case information lines

- You need to call these functions if you start a subshell or execute a shell script that uses the API

THE Open GROUP

Feb-05 TETware training course 7-17

Canceling test purposes

- `tet_delete` ***`test-name reason`***
  - cancels or reactivates a test purpose
    - the TCM does not call a canceled test purpose function
    - to cancel a test purpose, you specify the name of the test purpose function and a string
      - the string should describe why the test purpose is to be cancelled
    - to reactivate a canceled test purpose, you specify the name of the test purpose function and an empty reason string
- `tet_reason` ***`test-name`***
  - prints the reason string on the standard output
    - or prints an empty string if the test purpose is active

THE Open GROUP

Feb-05 TETware training course 7-18

Accessing configuration variables

- The API makes configuration variables available to test purpose functions as readonly Shell variables

Generating and executing processes

- The TCM executes each test purpose function in its own subshell
- You can use normal Shell syntax if you need another subshell below that
- If you create a subshell:
  - you should call `tet_setcontext` in the subshell
  - you should call `tet_setblock` in the parent shell after the subshell code

API library files

- XPG3 Shell API:
  - TCM
    - `$TET_ROOT/lib/xpg3sh/tcm.sh`
  - API library
    - `$TET_ROOT/lib/xpg3sh/tetapi.sh`
- Korn Shell API:
  - TCM
    - `$TET_ROOT/lib/ksh/tcm.ksh`
  - API library
    - `$TET_ROOT/lib/ksh/tetapi.ksh`

THE Open GROUP

Feb-05 TETware training course 7-21

API library files (cont'd)

- The Shell TCM must be *sourced* into the test case script by using the **.** (dot) command
- This command should be the last line in the file
- For example, to use the XPG3 Shell API:
  - `. ${TET_ROOT:?}/lib/xpg3sh/tcm.sh`
- or, to use the Korn Shell API:
  - `. ${TET_ROOT:?}/lib/ksh/tcm.ksh`
- Sourcing the TCM automatically sources the API as well

THE Open GROUP

Feb-05 TETware training course 7-22

Child processes and subprograms

- You can execute another shell script from your test purpose function
- If this script uses the API, it must *source* the API library near the top of the script using the `.` (dot) command
- For example:

```
. ${TET_ROOT:?}/lib/xpg3sh/tetapi.sh
```

- or:

```
. ${TET_ROOT:?}/lib/ksh/tetapi.ksh
```


Example Shell test case

```
:
tet_startup=""
tet_cleanup=""
iclist="ic1"
ic1="test1"

test1()
{
    tet_infoline "this is a trivial shell test case"
    tet_result PASS
}

. ${TET_ROOT:?}/lib/xpg3sh/tcm.sh
```

7.3 - The Perl API

- The Perl API is similar to the Shell APIs
- All the functions provided by the API are in the `tet` package

Interface to the user-written test code

- You tell the TCM about the invocable component names in your test case by defining an array called `iclist`
- You tell the TCM the names of your startup and cleanup functions by defining variables called `tet'startup` and `tet'cleanup`

`iclist` - list of invocable component names

- You should initialise each element in the `iclist` array to the name of an invocable component
- Each of these names should have the prefix `ic` followed by the invocable component number
- For example:

```
@iclist=(ic1,ic2,ic3);
```

- You should define an array for each invocable component, then initialise each element in the array to names of that IC's test purpose functions
- For example:

```
@ic1=("test1");
@ic2=("test2");
@ic3=("test3 test4");
```

THE Open GROUP

`tet'startup` and `tet'cleanup`

- You should set these variables to the names of your startup and cleanup functions
- If your test case doesn't use one of these functions you should set the corresponding variable to an empty string

THE Open GROUP

Perl API functions and variables

- Making journal entries
  - `&tet'infoline("`*`text`*`");`
  - `&tet'result("`*`result-name`*`");`
  - `&tet'setcontext;` and `&tet'setblock;`
- Canceling test purposes
  - `&tet'delete("`*`test-name`*`"` [, `"`*`reason-string`*`"`]`);`
  - *`deletion-reason`* `= &tet'reason("`*`test-name`*`");`
- Name of the current test purpose
  - `$tet'thistest`

THE Open GROUP

Feb-05 TETware training course 7-29

Accessing configuration variables

- The API makes configuration variables available to functions as variables within the `tet` namespace
- For example, if you define a configuration variable called `MY_VAR`, you would access it in a function as `$tet'MY_VAR`

THE Open GROUP

Feb-05 TETware training course 7-30

API library files

- TCM
  - `$TET_ROOT/lib/perl/tcm.pl`
- API library
  - `$TET_ROOT/lib/perl/api.pl`
- The Perl TCM must be *sourced* into the test case script by using the `require` command
- This command should be the last line in the file
- For example:

```
require "$ENV{\"TET_ROOT\"}/lib/perl/tcm.pl"
```

- Sourcing the TCM automatically sources the API as well

Example Perl test case

```
#!/usr/bin/perl
@iclist=(ic1);
@ic1=("tp1");

sub tp1{

        &tet'infoline("This is a trivial test case");
        &tet'result("PASS");
}

require "$ENV{\"TET_ROOT\"}/lib/perl/tcm.pl";
```

Module 8

Distributed testing

Module 8 - Distributed testing

- 8.1 - What is a distributed test case ?
- 8.2 - Logical systems and physical machines
- 8.3 - Distributed configuration variables
- 8.4 - APIs that support distributed testing
- 8.5 - The Test Case Manager (TCM)
- 8.6 - API functions for use in distributed test cases
- 8.7 - Test case synchronisation

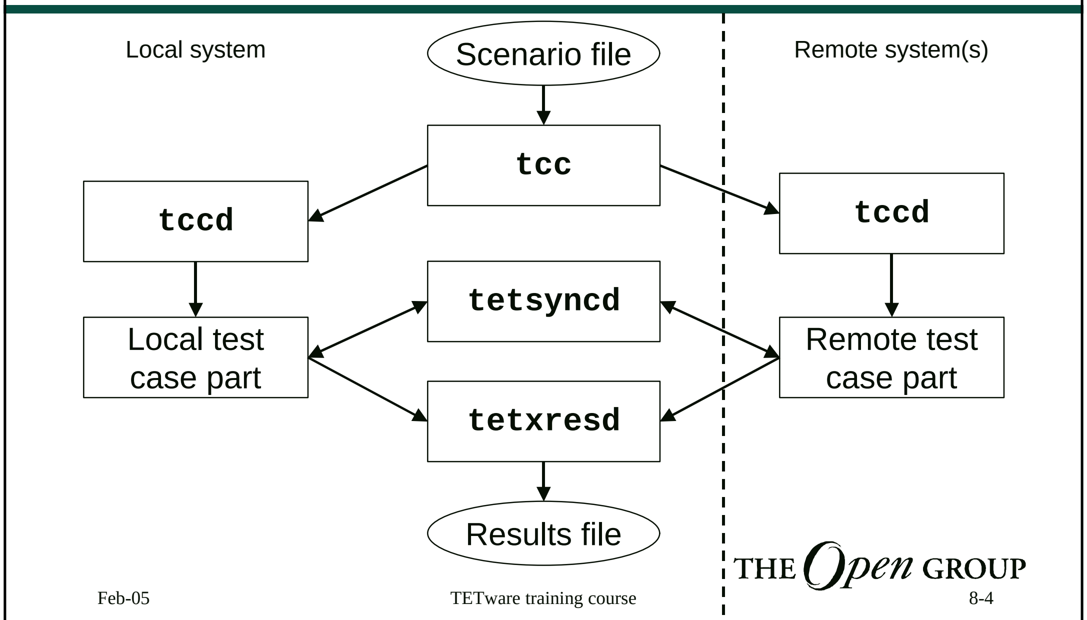
8.1 - What is a distributed test case ?

- We have already learned that:
  - a distributed test case:
    - consists of several parts which interact with each other
    - is typically used to test some kind of interaction between computer systems
  - each part is processed on a different system
  - each part contributes to a common result
- This is not the same as “running tests on several machines at once”

THE Open GROUP

Feb-05 TETware training course 8-3

Simple architecture diagram for Distributed TETware



THE Open GROUP

Feb-05 TETware training course 8-4

Specifying a distributed test case in the scenario

- You specify a distributed test case by using the `:distributed:` directive
  - the directive's parameters are the numerical IDs of the systems on which the test case parts will run
- For example:

```
:distributed,000,001,002:
    /tset/test1/tc1
    /tset/test2/tc2
    ...
:enddistributed:
```

- In this example, `tcc` processes three test case parts at the same time - one on each system

Processing a distributed test case

- A distributed test case **must** use the API
- A build or clean tool may be API-conforming or non API-conforming
- `tcc` processes each part of a distributed test case at the same time
  - they are built at the same time
  - executed at the same time
  - cleaned up at the same time

Structure of a distributed test case

- Each part of a distributed test case must have the same structure
  - same number of test purpose functions
  - same assignment of invocable component numbers
  - in other words: each part must contain an identical `tet_testlist[]` array
- Each part of a distributed test purpose must generate a result (by calling `tet_result()`)
- If there is nothing for one of the test purpose parts to do, it should just report PASS and return

8.2 - Logical systems and physical machines

- We have already learned that:
  - TETware identifies each system by a three-digit **system ID**
  - Entries in the file *`tet-root`*`/systems` map system IDs to host names (or IP addresses)
  - It is possible to map more than one logical system ID to the same physical machine
- It is best to set up separate **test suite root** directories for each system when several systems are mapped to the same machine

The `systems` file

- When the socket interface is used, fields are system ID and host name; for example:

```
000     argon
001     neon
002     89.0.0.24
```

- Each host name should be a **real** host name or IP address - not `localhost`
- The `systems` file must be provided on all participating systems and must have the same contents
  - otherwise chaos will break out when the systems try to talk to each other!

8.3 - Distributed configuration variables

- `tcc` gets information about the test suite on the local system from environment variables and the current working directory
  - in particular: the location of the **tet root** and **test suite root** directories
  - but this information probably doesn't apply to remote systems
- You specify this information about the remote parts of the test suite by using distributed configuration variables
  - these variables are defined on the local system in the file *`test-suite-root`*`/tetdist.cfg`

Distributed configuration variables (cont'd)

- Each variable starts with a `TET_REM`*nnn*`_` prefix
- *nnn* indicates to which remote system the variable refers
- A `TET_REM000_` prefix has no meaning in the distributed configuration

Required distributed configuration variables

- These variables must be defined for each remote system
- `TET_REM`*nnn*`_TET_ROOT`
  - specifies the location of the **tet root** directory on system *nnn*
- `TET_REM`*nnn*`_TET_TSROOT`
  - specifies the location of the **test suite root** directory on system *nnn*

8.4 - APIs that support distributed testing

- A distributed test case must be built using an API that supports distributed testing
  - these are: the versions of the C and C++ APIs that are supplied with Distributed TETware
- The Distributed `tcc` can process test cases that use other APIs as (non-distributed) remote test cases

Feb-05 TETware training course THE Open GROUP 8-13

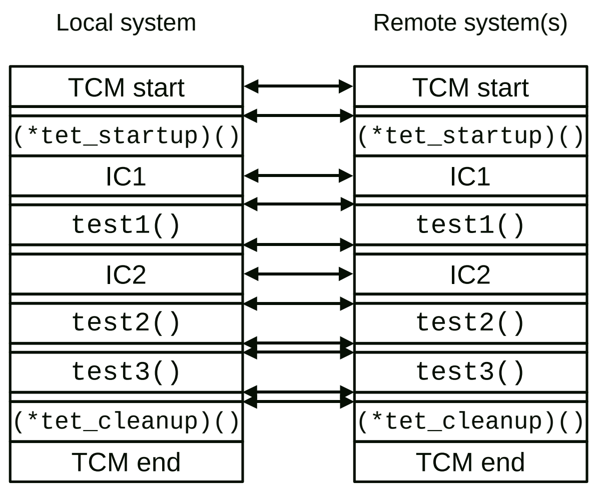
8.5 - The Test Case Manager (TCM)

- The TCMs synchronise between parts of a distributed test case, so as to keep them in step
- These synchronisation points are:
  - at TCM startup time
  - before calling a user-supplied startup function
  - at the start of each invocable component
  - before calling each test purpose function
  - after each test purpose function returns
  - before calling a user-supplied cleanup function

Feb-05 TETware training course THE Open GROUP 8-14

TCM's automatic synchronisation points

```
struct tet_testlist tet_testlist[] = {
  { test1, 1 },
  { test2, 2 },
  { test3, 2 },
  { NULL, 0 }
};
```



8.6 - API functions for use in distributed test cases

- Remote system information
- Executed process functions
- Remote process control
- Synchronisation

Remote system information

- `int tet_remgetlist(int **sysnames)`
  - returns the number of other systems in a distributed test case
  - a pointer to the list of system IDs is returned indirectly through `*sysnames`
  - the returned number of systems and system list can be passed to `tet_remsync()`
- `int tet_remgetsys(void)`
  - returns the system ID of the calling process

THE Open GROUP

Feb-05 TETware training course 8-17

Use of `tet_remgetlist()` and `tet_remgetsys()`

- For example, if parts of a distributed test case are running on systems 0, 1 and 2:
  - on system 1:

```
int sysid, *syslist, nsys;
sysid = tet_remgetsys();
nsys = tet_remgetlist(&syslist);
```

  - after these calls:
    - `sysid = 1` my system ID
    - `nsys = 2` number of other systems
    - `syslist[0] = 2` the list of other systems
    - `syslist[1] = 0`

THE Open GROUP

Feb-05 TETware training course 8-18

Remote system information (cont'd)

- `tet_getsysbyid(int sysid, struct tet_sysent *sysp)`
  - access information in the `systems` file
- `tet_remtime(int sysid, time_t *tp)`
  - get remote system time

Feb-05 TETware training course THE Open GROUP 8-19

Executed process functions

- `tet_exit(int status)`
  - log off servers and exit
- `tet_logoff(void)`
  - log off servers
  - the result of calling an API function is undefined after a call to this function

Feb-05 TETware training course THE Open GROUP 8-20

Remote process control

- Use of these functions is discouraged!
- `int tet_remexec(int sysid, char *file, char *argv[])`
  - start a subprogram on a remote system that will use the API
    - the subprogram must be linked with the remote process controller (`tcmrem.o`)
  - returns a `remoteid` which identifies the process

THE Open GROUP

Feb-05 TETware training course 8-21

Remote process control (cont'd)

- `tet_remwait(int remoteid, int waittime, int *statloc)`
  - waits for a subprogram started by `tet_remexec()` to terminate
  - the status returned indirectly through `*statloc` uses a standard encoding
- `tet_remkill(int remoteid)`
  - terminates a subprogram started by `tet_remexec()`

THE Open GROUP

Feb-05 TETware training course 8-22

Synchronisation

- Synchronisation functions
  - `tet_remsync()`
  - obsolete - provided for backward compatibility:
    - `tet_sync()`
    - `tet_msync()`
- Control over sync error reporting
  - `(*tet_syncerr)()`
  - `tet_syncreport()`

THE Open GROUP

Feb-05 TETware training course 8-23

8.7 - Test case synchronisation

- Introduction
- What can synchronise (and what can't)
- Basic concepts
- Defining sync point numbers
- Identifying sync points in a test purpose
- Sync point leapfrogging
- Using `tet_remsync()`

THE Open GROUP

Feb-05 TETware training course 8-24

Introduction

- The key to understanding distributed testing is understanding synchronisation
- It is a complicated subject which is worth taking time to get right

Feb-05 TETware training course THE Open GROUP 8-25

Introduction (cont'd)

- Synchronisation is used to make sure that things in different parts of a distributed test case happen in the correct order
- We have already learned that:
  - the TCMs use synchronisation to keep the parts of a distributed test case in step with each other
  - these are called **automatic** sync points
- In this section we will learn how to use **user-defined** sync points in a distributed test purpose function

Feb-05 TETware training course THE Open GROUP 8-26

What can synchronise (and what can't)

- Synchronisation can be performed between parts of the same distributed test case that are running on different systems
- Synchronisation can't be performed between:
  - processes running on the same system
  - processes that are not part of the same distributed test case

Basic concepts

- Defining a user sync event
- Sync point number
- Sync vote
- Sync state
- Message data
- Delegating sync authority

Defining a user sync event

- A sync event is defined by:
  - the list of systems that will participate
  - the sync point number to be used
- Synchronisation is co-operative
  - all the participating systems must be prepared to sync with each other if the event is to succeed
  - for example: if system 0 expects to sync with system 1, system 1 must also expect to sync with system 0

Sync point number

- Systems synchronise to a particular sync point number
- The value of the sync point number must increase throughout the life of the test case
  - so it is necessary to allocate separate number ranges for use by each test purpose function
- A zero value means “the next sync point”
- At least one system must specify a non-zero sync point

Sync vote

- Each system votes in a sync event
- The vote says whether or not the event should succeed
- The event is successful if everyone votes yes
- The event fails if at least one system abstains or votes no

Sync state

- The API maintains a set of **sync states** for each sync event
- There is one sync state for each system that will take part in the event
- The states are:
  - `SYNC-YES` the system has voted yes
  - `SYNC-NO` the system has voted no
  - `NOT-SYNCED` the system has not yet voted
  - `TIMED-OUT` the system has voted but then timed out
  - `DEAD` the system has disconnected from the server

Message data

- It is possible for systems to exchange message data during a sync event
- One system sends the data
- All the other systems receive the data if the sync event is successful

THE Open GROUP

Feb-05 TETware training course 8-33

Delegating sync authority

- Synchronisation is defined in terms of **systems**, not individual **processes**
- The test suite author must ensure that only one process on a particular system will attempt to take part in a particular sync event

THE Open GROUP

Feb-05 TETware training course 8-34

Defining sync point numbers

- We have already learned that:
  - sync point numbers must increase throughout the life of a test case
  - so it is necessary to allocate separate number ranges for use by each test purpose function
- Sometimes, sync point numbers are used in library functions
  - so it can also be necessary to allocate blocks of sync point numbers for use in library functions

Feb-05 TETware training course THE Open GROUP 8-35

Defining sync point numbers (cont'd)

- In XNFS we used a macro to generate sync point numbers; it is defined as follows:

```
#define MK_SPNO(n) \
    ((tet_thistest << 8) | ((n) << 4))
```

  - this macro can be used up to 16 times in any test purpose
  - the value generated by this macro can be:
    - passed directly to `tet_remsync()` in the test purpose function
    - used to generate a **base sync point number** which can be passed to a library function which performs synchronisation
      - the library function can also use up to 16 sync point numbers, starting from the base value

Feb-05 TETware training course THE Open GROUP 8-36

Identifying sync points in a test purpose

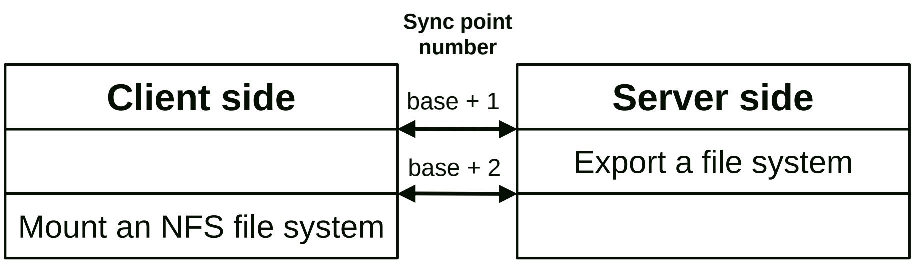
- Consider this strategy in an NFS test:
  - Server: export a file system using specified attributes
  - Client: mount the file system using NFS
  - Client: create a file on the mounted file system
  - Server: check that the file was created correctly and generate a result
  - Client: unmount the NFS file system
  - Server: unexport the file system

Identifying sync points (cont'd)

Client side	Sync point number	Server side
	1	
		Export a file system
	2	
Mount an NFS file system		
Create a file		
	3	
		Verify the file
	4	
Unmount the file system		
	5	
		Unexport the file system

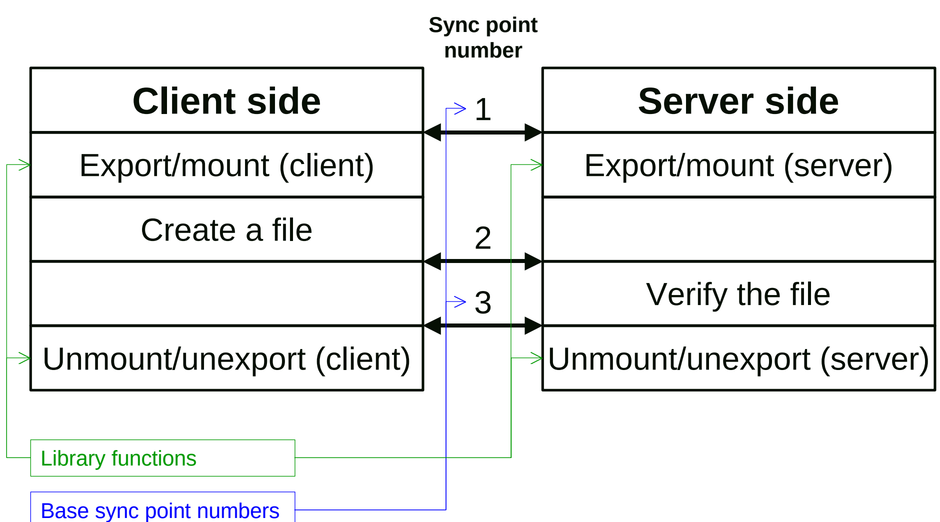
Sync points in library functions

- Sometimes, sync points are used in library functions
  - the library function has client and server parts
- You can pass a **base sync point number** to the function
- The base sync point number is generated by `MK_SPNO()`



Feb-05 TETware training course 8-39

Using library functions with sync points



Feb-05 TETware training course 8-40

Sync point leapfrogging

Client side	Sync point number	Server side
	1	
	2	Export a file system
Mount an NFS file system		
Create a file	3	
	4	Verify the file
Unmount the file system	5	
		Unexport the file system

- In this example:
  - the client does nothing between sync points 1 and 2, and between 3 and 4
  - the server does nothing between sync points 2 and 3, and between 4 and 5
- In the code for each test purpose part there would be pairs of calls to `tet_remsync()` with nothing in between them
- So in the client, sync points 1 and 3 are redundant and in the server, sync points 2 and 4 are redundant
  - the client only needs to sync to points 2, 4 and 5
  - the server only needs to sync to points 1, 3 and 5

Using `tet_remsync()`

- `tet_remsync(long syncptno, int *sysnames, int nsysname, int waittime, int vote, struct tet_synmsg *msgp)`
- `syncptno` is the sync point number
- `sysnames` points to the system list
- `nsysname` specifies the number of systems in the list
- `waittime` specifies the timeout
  - +ve value - number of seconds to wait for other systems to sync
  - zero value - return immediately without waiting
  - -ve value - wait indefinitely
- `vote` specifies the sync vote
  - TET_SV_YES or TET_SV_NO
- `msgp` points to a structure which describes message data
  - or `NULL` if no message data is to be sent or received

tet_remsync() example

```
int systems[] = { 1, 2 };
int nsys = sizeof systems / sizeof systems[0];

if (tet_remsync(MK_SPNO(1), systems, nsys, 30, TET_SV_YES, NULL) < 0) {
    switch (tet_errno) {
    case TET_ER_SYNCERR:
        /* another system didn't sync, voted NO, timed out or died */
        break;
    case TET_ER_TIMEDOUT:
        /* this system's timeout expired */
        break;
    case TET_ER_DONE:
        /* event already happened - we've missed it! */
        break;
    default:
        /* unexpected error */
        break;
    }
}
```

THE Open GROUP

Feb-05 TETware training course 8-43

Control over sync error reporting

- When a call to `tet_remsync()` is unsuccessful:
  - the API sets `tet_errno`
  - the API calls the function pointed to by `tet_syncerr`
- Initially this variable points to the API's default sync error reporting function `tet_syncreport()`
  - this function prints error messages describing the error in the journal
- You can change this if you want to do your own sync error handling

THE Open GROUP

Feb-05 TETware training course 8-44

(*tet_syncerr)()

- `(*tet_syncerr)(long syncpno,`
  `struct tet_syncstat *statp, int nstat)`
- syncpno is the number of the sync point that failed
- `statp` points to an array of `tet_syncstat` structures
  - each element describes one of the other systems
- `nstat` is the number of elements pointed to by `statp`
- The sync error handler can inspect the values in the sync status array to find out which other system(s) caused the event to fail

THE Open GROUP

Feb-05 TETware training course 8-45

tet_syncstat - per-system sync status

```
struct tet_syncstat {
    int tsy_sysid;   /* system id */
    int tsy_state;   /* system's sync state */
}
```

- Values for `tsy_state:`
  - `TET_SS_NOTSYNCED` system hasn't voted yet
  - `TET_SS_SYNCYES` system voted yes
  - `TET_SS_SYNCNO` system voted no
  - `TET_SS_TIMEDOUT` system timed out after voting
  - `TET_SS_DEAD` system disconnected after voting

THE Open GROUP

Feb-05 TETware training course 8-46

Exercise 8