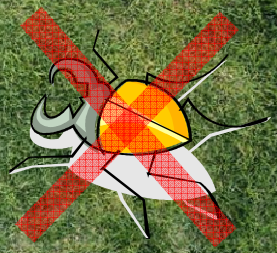
Erlang Factory, London, June 10-11th 2010

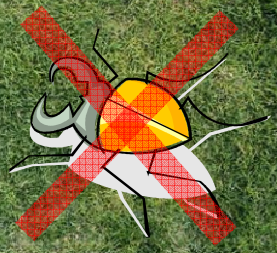
Common Test



101

Erlang Factory, London, June 10-11th 2010

CCTCC



The Compact Common Test Crash Course

ERICSSON

Peter Andersson
Ericsson AB, Erlang/OTP
Stockholm, Sweden

Email: peppe@erlang.org

contents

The Common Test Framework

1. Overview – what is Common Test?

Common Test

What is the **C**ommon **T**est framework?

- A portable test server for black-box testing (function and system testing) target systems of any type.
- A practical tool for white-box testing OTP applications and Erlang programs.

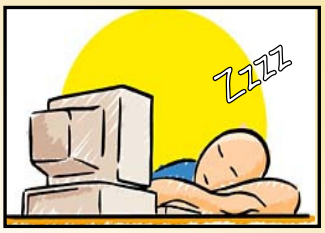
ERICSSON

Common Test provides:

- Possibility to run test suites automatically
- HTML progress and result logs
- Test suite templates and support libraries
- Support for running multiple test sessions in parallel on local or remote nodes
- Flexible test specification
- Flexible configuration
- Event handler interface for integration with other programs

ERICSSON

Regression testing

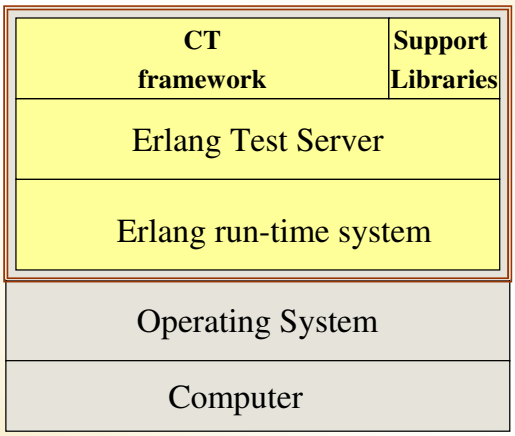


Common Test is suitable for ***regression testing***:

- Automated execution of test suite programs (no operator interaction required during test).
- Test progress and result logs are printed to file (on HTML format).
- Flexible test specification.
- Support for running multiple independent test sessions in parallel.

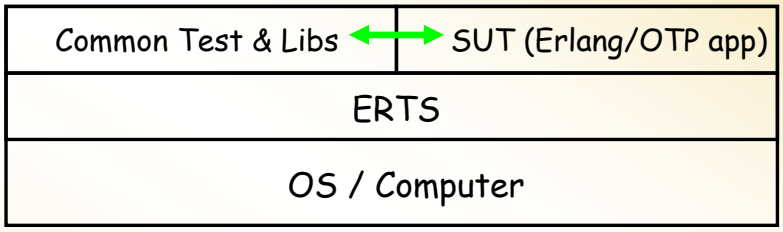
ERICSSON

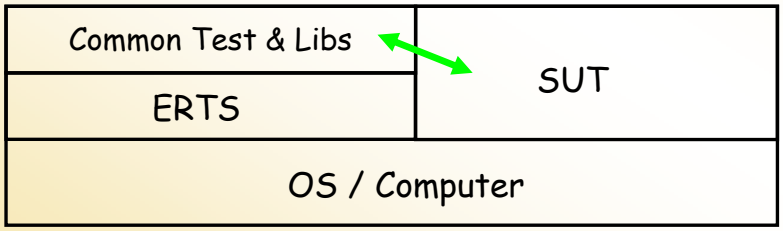
Common Test implementation structure



ERICSSON

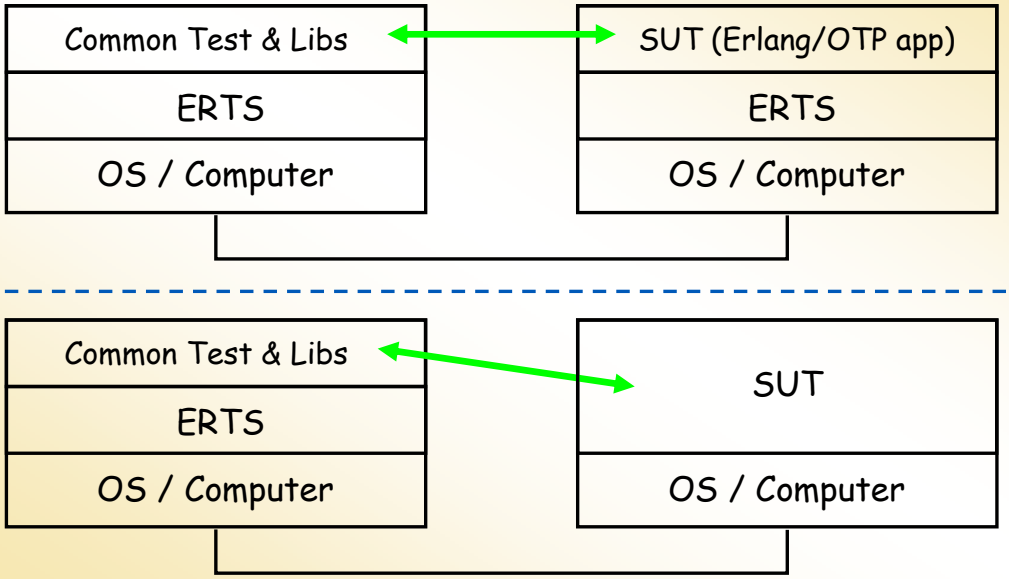
Testing scenarios





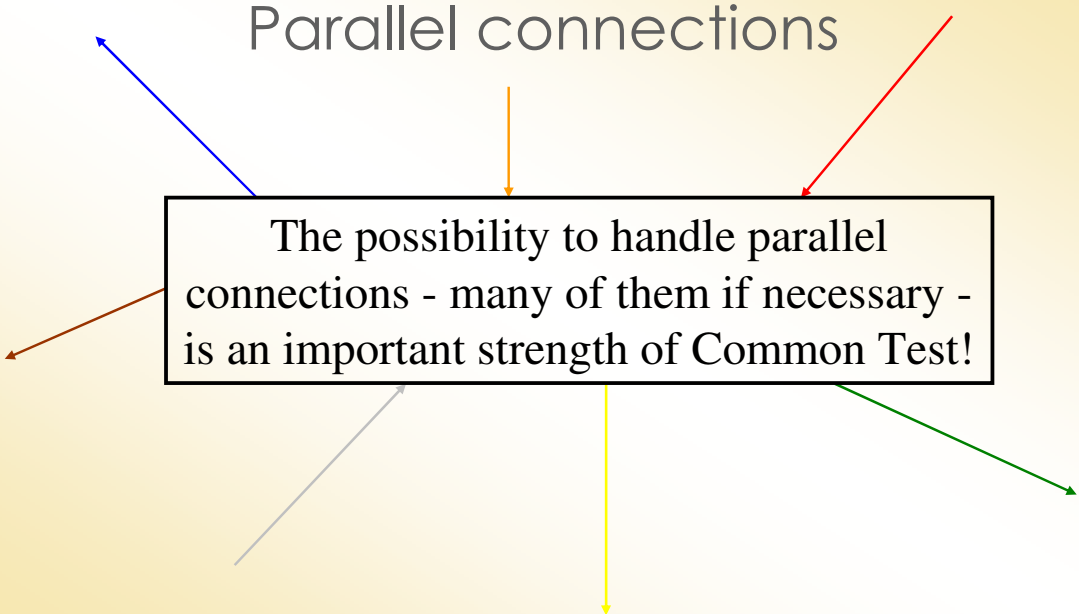
ERICSSON

Testing scenarios



ERICSSON

Parallel connections

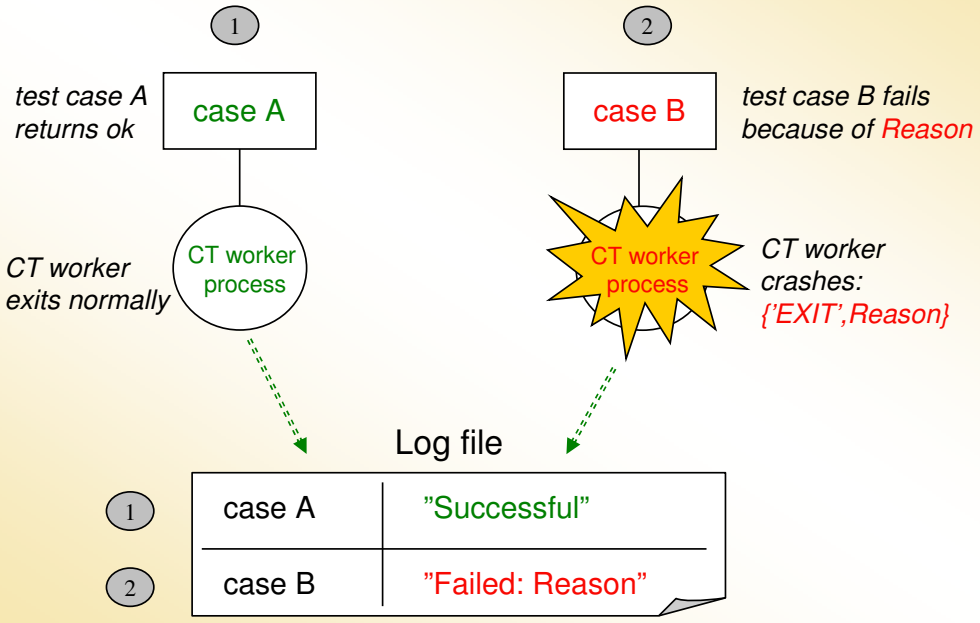


Support libraries

Any of the following libraries may be used (in any combination):

- CT support libraries for general protocols (e.g. FTP, SNMP, etc).
- Erlang/OTP libraries.
- Specific test object libraries (test ports).

Test case execution



ERICSSON

Erlang as language for test programs

- **Declarative, high-level, language with dynamic type system:**
  - = Short, concise, test code.
  - = Quick to implement, very little overhead.
  - = Easy to read and maintain.
- **Dynamic code loading (no static linking of modules) + Common Test *auto-compilation* feature:**
  - = Simple compilation and loading of test suites and other support library modules.
- **Support for concurrent programming built into the language and the runtime system.**
  - = Handle parallel connections.
  - = Scalability.

ERICSSON

Erlang as language for test programs

- ***Pattern matching* expressions:**

  = Tests and pre/post-conditions can be expressed as simple declarative one-liners. Example:

  ?SUCCESS = perform_operation(Conn, Op)

- **Erlang, a compact general-purpose language:**

  = Short time to learn enough to start working with test suite development and maintenance

  = Great flexibility when writing test programs.

ERICSSON

Parallel execution

A powerful feature of Common Test is its support for (and use of) ***parallel execution and communication.***

Important and useful test server characteristic that Common Test "gets for free", being an Erlang application.

ERICSSON

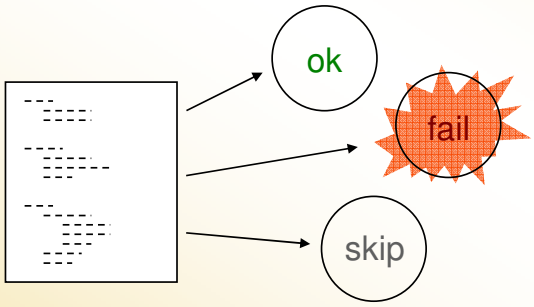
Parallel execution

Parallel execution provides for:

- Relevant testing of SUT that should be able to handle communication and events on **multiple interfaces in parallel**.
- Implementation of test programs where the possible numbers of available connections (and traffic) used in test can be **extended easily** as the SUTs mature and more SUTs and instruments are added over time.

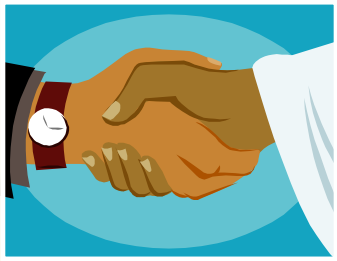
The Common Test Framework

2. Test suites, cases and groups



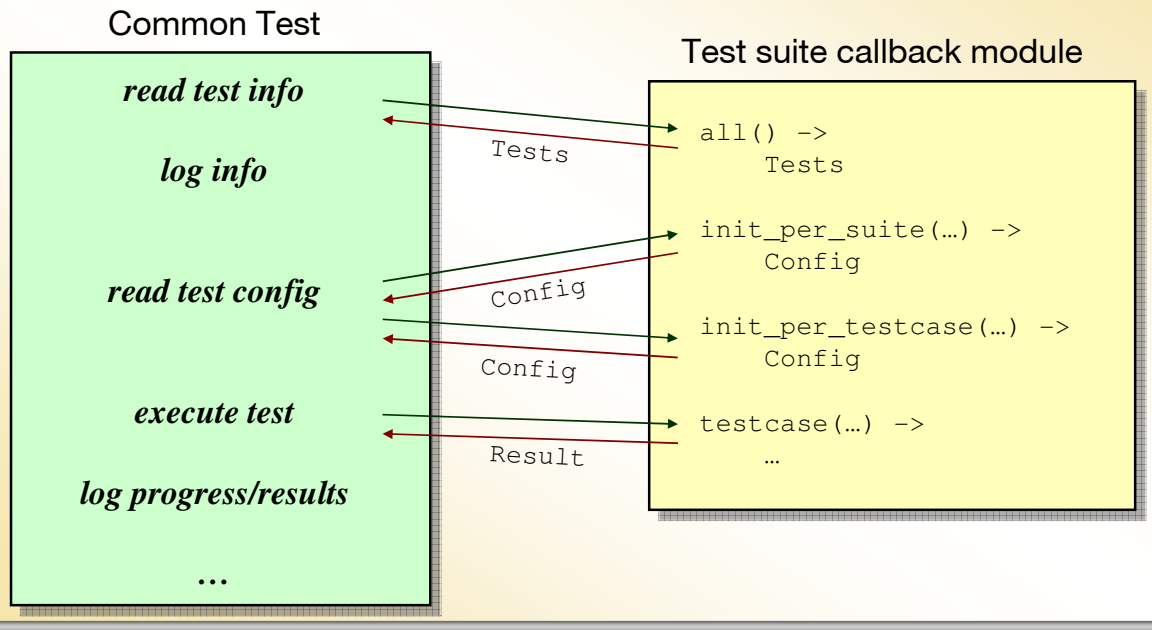
The test suite

The test suite is a callback module (similar to an OTP behaviour) that must comply with a defined test server interface.



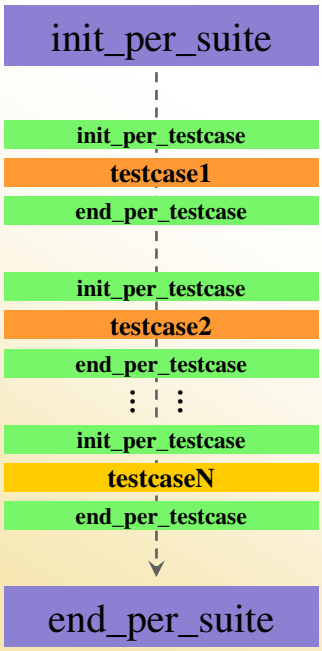
ERICSSON

The test suite callback module



ERICSSON

Simple test suite execution



init_per_suite/1 is called as the first test case in the suite. It typically contains initializations common for all test cases in the suite (operations that should only be done once).

init_per_testcase/2 is called before each test case in the suite. It typically contains initializations which must be done for each test case.

end_per_testcase/2 is called after each test case is completed, giving a possibility to clean up.

end_per_suite/1 is called as the last test case in the suite. This function should clean up after init_per_suite.

ERICSSON

Test case function

For each test case in the list returned from `all/0`, the test server calls a function with the same name and with one argument:

```
TestCaseName(Config)
```

- `Config` is the *runtime configuration data*.
- A test case is considered successful if it returns to the caller.
- A failed test case is one that crashes (or exits on purpose).

ERICSSON

Test case configuration

- Init configuration functions perform initialization (open connections, set up state, etc).
- End configuration functions clean up.
- Runtime configuration data is passed from the init config functions to the test cases and end config functions.

ERICSSON

Test case configuration

Example:

```
init_per_testcase(test_that_thing, Config) ->
  {ok,Handle} = ct_telnet:open(unix_telnet, telnet),
  [{telnet_handle,Handle} | Config].   % key = atom()

test_that_thing(Config) ->
  Handle = proplists:get_value(telnet_handle, Config),
  ?START_RESULT = ct_telnet:cmd(Handle, ?START_OP),
  …

end_per_testcase(test_that_thing, Config) ->
  Handle = proplists:get_value(telnet_handle, Config),
  ct_telnet:close(Handle).
```


ERICSSON

Info functions

Info functions are used to set values for various test properties in Common Test (e.g. a timetrap timeout value). They may also be used to perform initial assertions and *configuration data variable* bindings.

- The ***test suite info*** function sets property values for all test cases in the suite. Example:

```
suite() ->
   [{timetrap,{minutes,8}}, {require,{node,[name]}}].
```

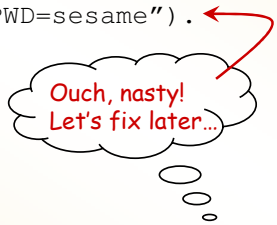
- A ***test case info*** function sets property values for a particular test case (and can override values set by `suite/0`). Example:

```
my_testcase() ->
   [{timetrap,{seconds,1},
    {userdata, {doc,"this test case tests stuff"}}].
```

ERICSSON

Test suite example

```
-module(db_data_type_SUITE).
-include("ct.hrl").
%% Test server callbacks
-export([suite/0, all/0, init_per_suite/1, end_per_suite/1,
         init_per_testcase/2, end_per_testcase/2]).
%% Test cases
-export([string/1, integer/1]).
-define(CONNECT_STR, "DSN=sqlserver;UID=alladin;PWD=sesame").
suite() ->
  [{timetrap,{minutes,1}}].
all() ->
  [string, integer].
```



ERICSSON

```
init_per_suite(Config) ->
   {ok, Ref} = db:connect(?CONNECT_STR, []),
   TableName = db_lib:unique_table_name(),
   [{con_ref, Ref },{table_name, TableName}| Config].

end_per_suite(Config) ->
   Ref = ?config(con_ref, Config),
   db:disconnect(Ref),
   ok.
```

ERICSSON

```
init_per_testcase(Case, Config) ->
    Ref = ?config(con_ref, Config),
    TableName = ?config(table_name, Config),
    ok = db:create_table(Ref, TableName, table_type(Case)),
    Config.

end_per_testcase(_Case, Config) ->
    Ref = ?config(con_ref, Config),
    TableName = ?config(table_name, Config),
    ok = db:delete_table(Ref, TableName),
    ok.
```


ERICSSON

```
string(Config) ->
    insert_and_lookup(dummy_key, "Dummy string", Config).

integer(Config) ->
    insert_and_lookup(dummy_key, 42, Config).

insert_and_lookup(Key, Value, Config) ->
    Ref = ?config(con_ref, Config),
    TableName = ?config(table_name, Config),
    ok = db:insert(Ref, TableName, Key, Value),
    [Value] = db:lookup(Ref, TableName, Key),
    ok = db:delete(Ref, TableName, Key),
    [] = db:lookup(Ref, TableName, Key),
    ok.
```

ERICSSON

Skipping test cases

It is possible to skip execution of test cases by:

- Returning {skip,Reason} from `init_per_testcase/2` or `init_per_suite/1`.
- Returning `{skip,Reason}` from the test case function (means the function is called and the author needs to make sure the actual test is not executed).

When an init configuration function fails, Common Test will automatically skip the associated test case(s).

ERICSSON

Test case groups

- **Sets of test cases**.
- Possibility to nest groups (and pass *Config* to sub-groups).
- Group execution properties (possibly combined):

  parallel - parallel execution of test cases

  sequence - sequence of test cases

  shuffle | ***{shuffle,Seed}*** - random order of test cases

  {Repeat,N} - repetition of group N times or until success or failure of any case or all cases

  ```
  (Repeat = repeat | repeat_until_all_ok | repeat_until_all_fail |
            repeat_until_any_ok | repeat_until_any_fail)
  ```

- Calls to configuration functions `init_per_group/2` and `end_per_group/2` before and after execution of each group.

ERICSSON

Groups are declared with function `groups/0`:

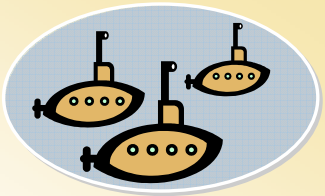
```
groups() ->
  [{NameOfGroup1, Properties1, TestCasesAndSubGroups1},
   {NameOfGroup2, Properties2, TestCasesAndSubGroups2},
    …
   {NameOfGroupN, PropertiesN, TestCasesAndSubGroupsN}].
```

Groups are ordered with test cases in the `all/0` list:

```
all() -> TestCasesAndGroups

%% TestCasesAndGroups = [TestCaseOrGroup, …]
%% TestCaseOrGroup = TestCase | {group,NameOfGroup}
%% TestCase = NameOfGroup = atom()
```


ERICSSON

Declaration of sub-groups:



- **Alt 1**

```
groups() ->
  [{Group1, Props1, [{Group11, Props11, TCsAndGroups11},
                     {Group12, Props12, TCsAndGroups12}, …]},
   …].
```

- **Alt 2**

```
groups() ->
  [{Group1, Props1, [{group,Group11}, {group,Group12}, …]},
   …,
   {Group11, Props11, TCsAndGroups11},
   {Group12, Props12, TCsAndGroups12},
   …].
```

ERICSSON

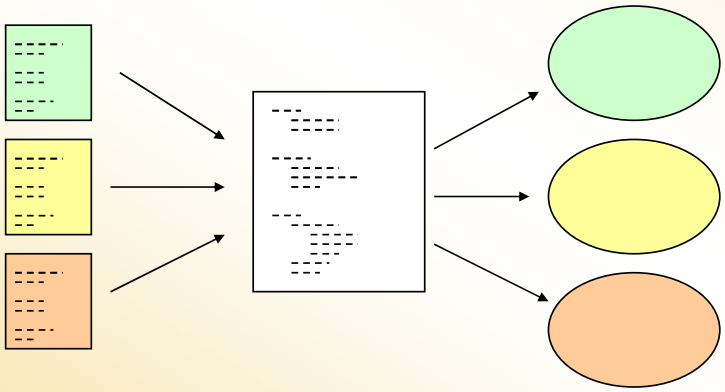
Test case groups example:

```
...
init_per_group(group1, Config) ->
       init_group1(Config);
init_per_group(group2, Config) ->
       init_group2(Config);
init_per_group(_GroupName, Config) ->
       Config.
end_per_group(_GroupName, _Config) ->
       ok.

groups() -> [{group1, [parallel], [tc11,t12,tc13]},
             {group2, [], [tc21,{group,group3},tc22,{group,group4}]},
             {group3, [sequence,shuffle], [tc31,tc32,tc33]},
             {group4, [{repeat,10},shuffle], [tc41,tc42,tc43]}].

all() -> [tc1, {group,group1}, tc2, tc3, {group,group2}].
...
```


ERICSSON

The Common Test Framework

3. External configuration data



ERICSSON

External configuration data

The Common Test user may specify data related to the test, or the system under test, by means of **configuration files or strings**.

Configuration data makes it possible to change properties without having to modify test suites. Examples:

- Addresses to the SUT or instruments
- Identities
- Authentication data
- Names of files and programs needed by the test

ERICSSON

config files

Common Test can read configuration data elements on **Erlang tuple form** from file, and from **XML specifications**.

The Erlang tuple representation has the form:

```
{Key,Value}.
```

where

```
Key = atom()
Value = term() | [{Key,Value}]
```

`Key` is the name of the *configuration variable*

userconfig

The Common Test user may read configuration data on **arbitrary form** by providing a callback module to handle the data. Useful for:

- Reading data from other types of files than Erlang- or XML-files.
- Reading strings that may be used for configuration or for obtaining configuration data from somewhere else.

Example:

```
$ run_test -userconfig db_login "testuser lETmEiN"
```

will result in call to:

```
db_login:check_parameter(Str="testuser lETmEiN") -> {ok, Str}
```

and:

```
db_login:read_config("testuser lETmEiN") ->
     {ok, [{login, [{user,"testuser"}, {pwd,"lETmEiN"}]}]}
```

The config terms will be loaded by Common Test and can be used in test cases by means of *require* and *get_config*.

ERICSSON

Configuration variables and require

One can **within a test suite** *require* (i.e. assert) that a variable exists in loaded configuration data.

There are 3 ways to require a variable:

- Specify a require tuple in the `suite/0` return list.
- Return a require tuple from the test case info function
- Call `ct:require/[1,2]` from a test case.

In the test case, the value of a variable may be read using the function: `ct:get_config/1/2/3`.

ERICSSON

Configuration file example

Example of a configuration file:

```
{ftp, [{host,"134.138.177.105"},
       {user,"testuser"},
       {password,"123"}]}.
{url, "http://134.138.177.105:8888/"}.
{install_script, unix_ws_install}.
```

Example of how to access configuration data inside a test case:

```
FtpAddr = ct:get_config({ftp,host}),
URL = ct:get_config(url), …
```

The Common Test Framework

4. Test execution

Test execution

The program ***run_test*** can be used for starting tests from the **OS command line**. Examples:

```
$ run_test -config <configfilenames> -dir <dirs>

$ run_test -config <configfilenames> -suite <suiteswithfullpath>

$ run_test -config <configfilenames> -suite <suitewithfullpath>
           -case <casenames>
```

Examples:

```
$ run_test -config node.cfg -dir objX_test/

$ run_test -suite objX_test/objX_setup_SUITE objY_test/objY_setup_SUITE
```

ERICSSON

Starting from the OS command line

Examples of other useful run_test flags:

- `-logdir <dir>`, specifies where the HTML log files are to be written.
- `-refresh_logs`, refreshes the top level HTML index files.
- `-stylesheet`, for installing a CSS file.
- `-cover`, for performing code coverage tests.
- `-include`, to add include directories for test suite compilation.
- `-no_auto_compile`, for running tests from pre-compiled (and possibly pre-loaded) suites.

For documentation about start flags, see the *run_test* reference manual page and the *Running Test Suites* chapter in the User's Guide.

ERICSSON

Starting from an Erlang shell prompt

The test execution function:

```
ct:run_test(Opts)
```

takes the same input options as the ***run_test*** script, but as **tuples in a list**.

Example:

```
1> ct:run_test([{logdir,"/ldisk/logdir"},
                {dir,"my_test_obj"}]).
```

ERICSSON

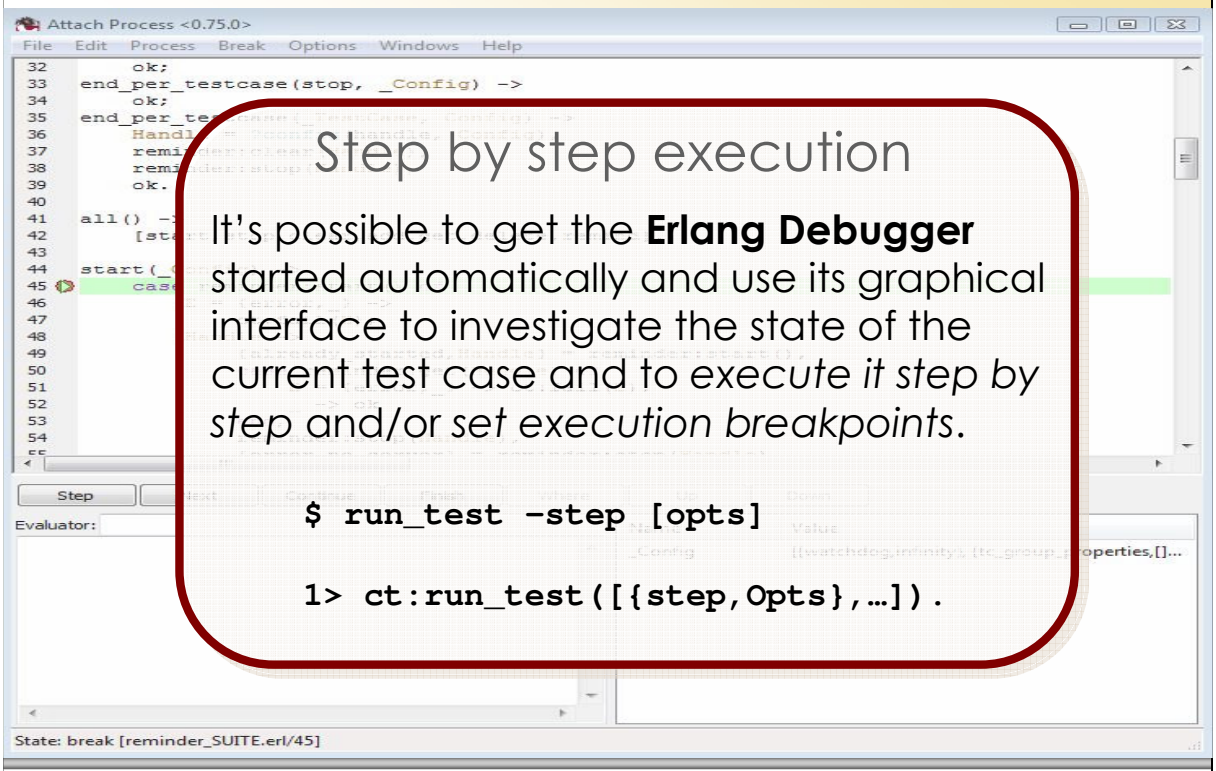
Starting `ct` in *interactive shell mode*

You can start Common Test in an *interactive shell mode*. Features:

- Evaluate test expressions in an Erlang shell instead of having CT execute tests automatically.
- Configuration data can be installed and used (required and retrieved) in the shell just like in a test suite.

```
$> run_test -shell

$> run_test -shell -config <configfilenames>

$> run_test -shell -userconfig <userconfigdata>
```


ERICSSON



Step by step execution

It's possible to get the **Erlang Debugger** started automatically and use its graphical interface to investigate the state of the current test case and to *execute it step by step* and/or *set execution breakpoints*.

```
$ run_test -step [opts]

1> ct:run_test([{step,Opts},…]).
```

The Common Test Framework

5. Test results and logs


Test results and logs

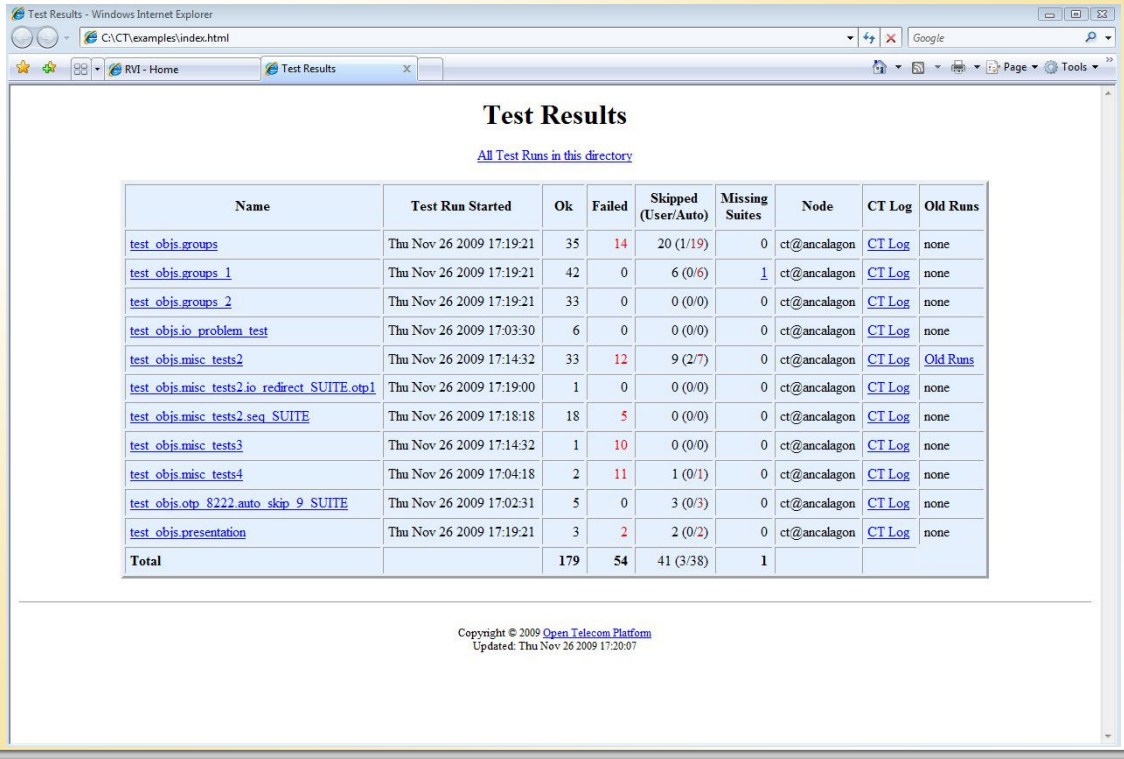
During the execution of a test suite, all information (incl. printouts to stdout) is recorded in log files, stored in a unique directory:

```
<log_dir>/ct_run.<node>.<date>_<time>.
```

- The result from each test case is printed to an individual test case log file.

- You get summary files generated during test runs. These give you a status overview and provide links to every test case log file.

ERICSSON



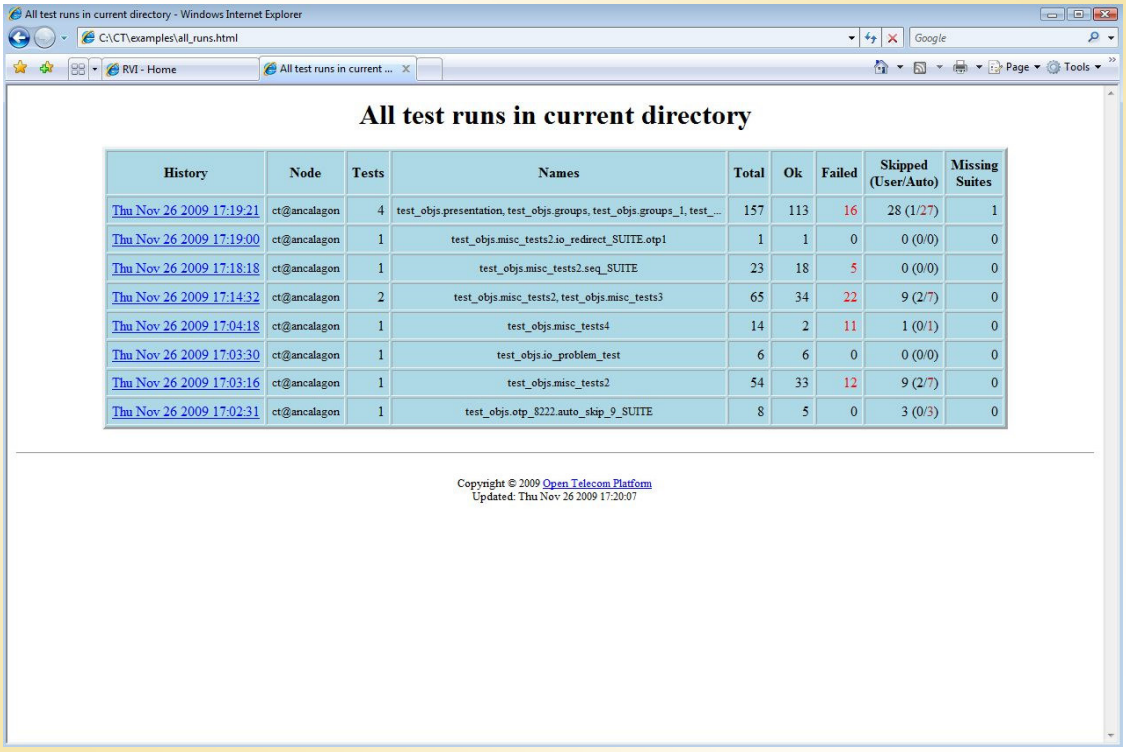
Test Results

All Test Runs in this directory

Name	Test Run Started	Ok	Failed	Skipped (User/Auto)	Missing Suites	Node	CT Log	Old Runs
test_objs.groups	Thu Nov 26 2009 17:19:21	35	14	20 (1/19)	0	ct@ancalagon	CT Log	none
test_objs.groups_1	Thu Nov 26 2009 17:19:21	42	0	6 (0/6)	1	ct@ancalagon	CT Log	none
test_objs.groups_2	Thu Nov 26 2009 17:19:21	33	0	0 (0/0)	0	ct@ancalagon	CT Log	none
test_objs.io_problem_test	Thu Nov 26 2009 17:03:30	6	0	0 (0/0)	0	ct@ancalagon	CT Log	none
test_objs.misc_tests2	Thu Nov 26 2009 17:14:32	33	12	9 (2/7)	0	ct@ancalagon	CT Log	Old Runs
test_objs.misc_tests2.io_redirect_SUITE.otp1	Thu Nov 26 2009 17:19:00	1	0	0 (0/0)	0	ct@ancalagon	CT Log	none
test_objs.misc_tests2.seq_SUITE	Thu Nov 26 2009 17:18:18	18	5	0 (0/0)	0	ct@ancalagon	CT Log	none
test_objs.misc_tests3	Thu Nov 26 2009 17:14:32	1	10	0 (0/0)	0	ct@ancalagon	CT Log	none
test_objs.misc_tests4	Thu Nov 26 2009 17:04:18	2	11	1 (0/1)	0	ct@ancalagon	CT Log	none
test_objs.otp_8222.auto_skip_9_SUITE	Thu Nov 26 2009 17:02:31	5	0	3 (0/3)	0	ct@ancalagon	CT Log	none
test_objs.presentation	Thu Nov 26 2009 17:19:21	3	2	2 (0/2)	0	ct@ancalagon	CT Log	none
Total		**179**	**54**	41 (3/38)	**1**			

Updated: Thu Nov 26 2009 17:20:07

ERICSSON

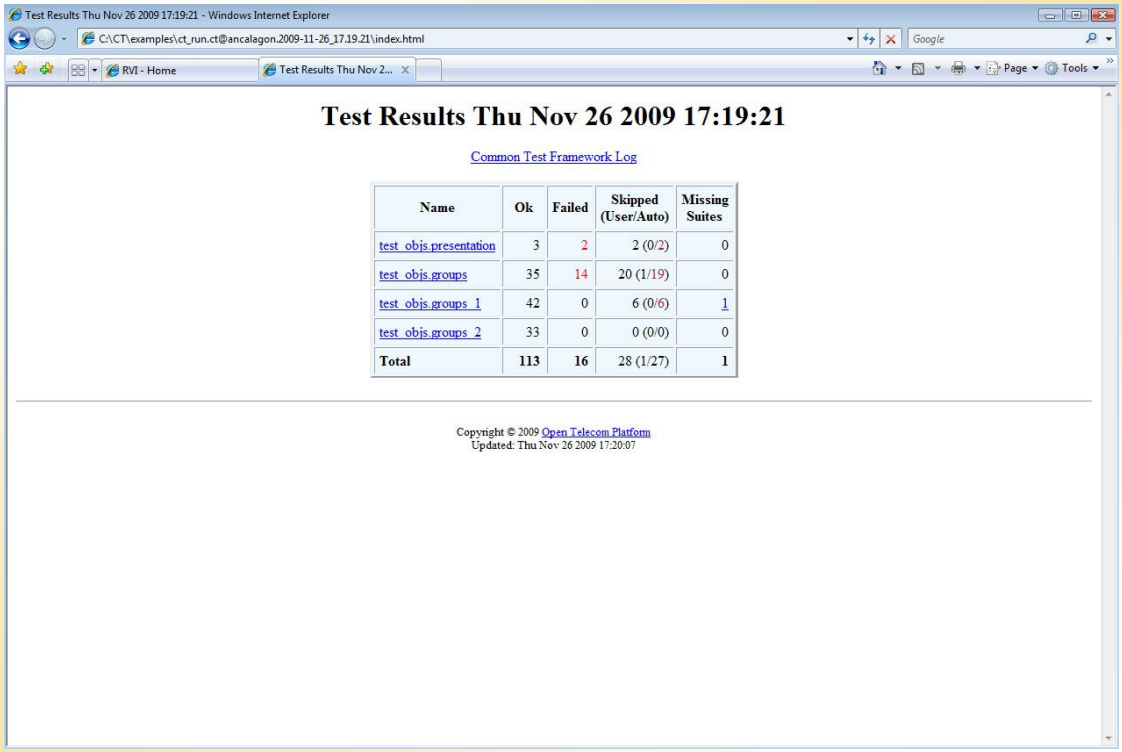


All test runs in current directory

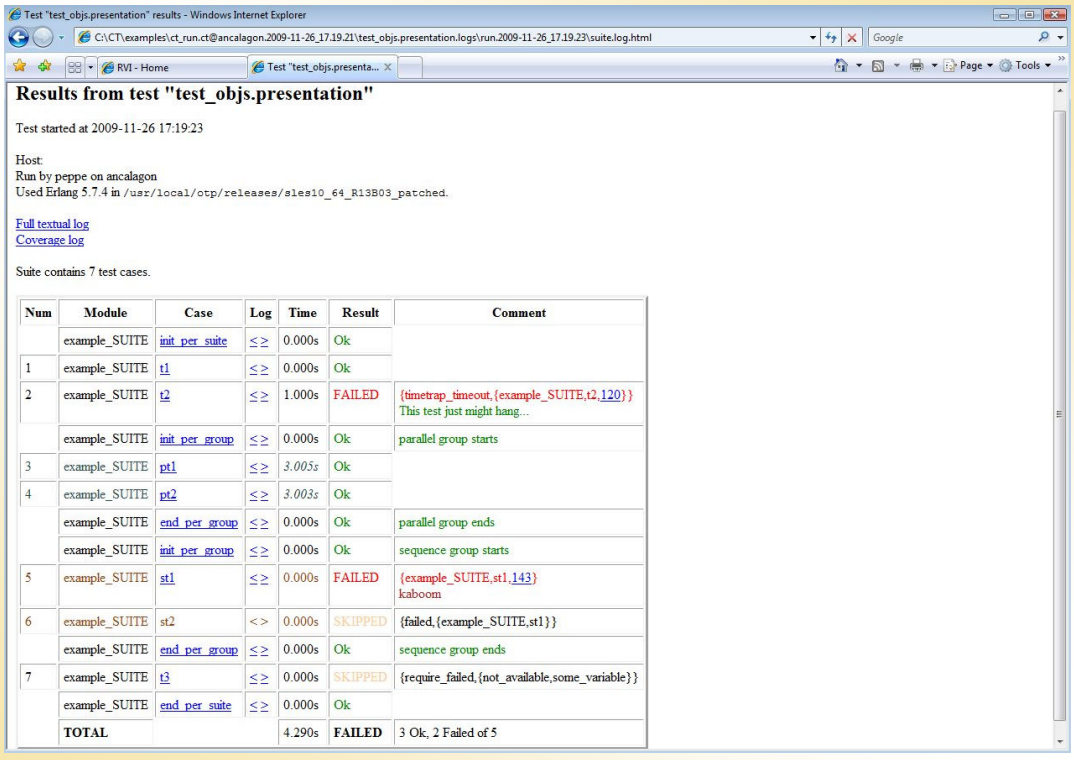
History	Node	Tests	Names	Total	Ok	Failed	Skipped (User/Auto)	Missing Suites
Thu Nov 26 2009 17:19:21	ct@ancalagon	4	test_objs.presentation, test_objs.groups, test_objs.groups_1, test_...	157	113	16	28 (1/27)	1
Thu Nov 26 2009 17:19:00	ct@ancalagon	1	test_objs.misc_tests2.io_redirect_SUITE.otp1	1	1	0	0 (0/0)	0
Thu Nov 26 2009 17:18:18	ct@ancalagon	1	test_objs.misc_tests2.seq_SUITE	23	18	5	0 (0/0)	0
Thu Nov 26 2009 17:14:32	ct@ancalagon	2	test_objs.misc_tests2, test_objs.misc_tests3	65	34	22	9 (2/7)	0
Thu Nov 26 2009 17:04:18	ct@ancalagon	1	test_objs.misc_tests4	14	2	11	1 (0/1)	0
Thu Nov 26 2009 17:03:30	ct@ancalagon	1	test_objs.io_problem_test	6	6	0	0 (0/0)	0
Thu Nov 26 2009 17:03:16	ct@ancalagon	1	test_objs.misc_tests2	54	33	12	9 (2/7)	0
Thu Nov 26 2009 17:02:31	ct@ancalagon	1	test_objs.otp_8222.auto_skip_9_SUITE	8	5	0	3 (0/3)	0

Updated: Thu Nov 26 2009 17:20:07

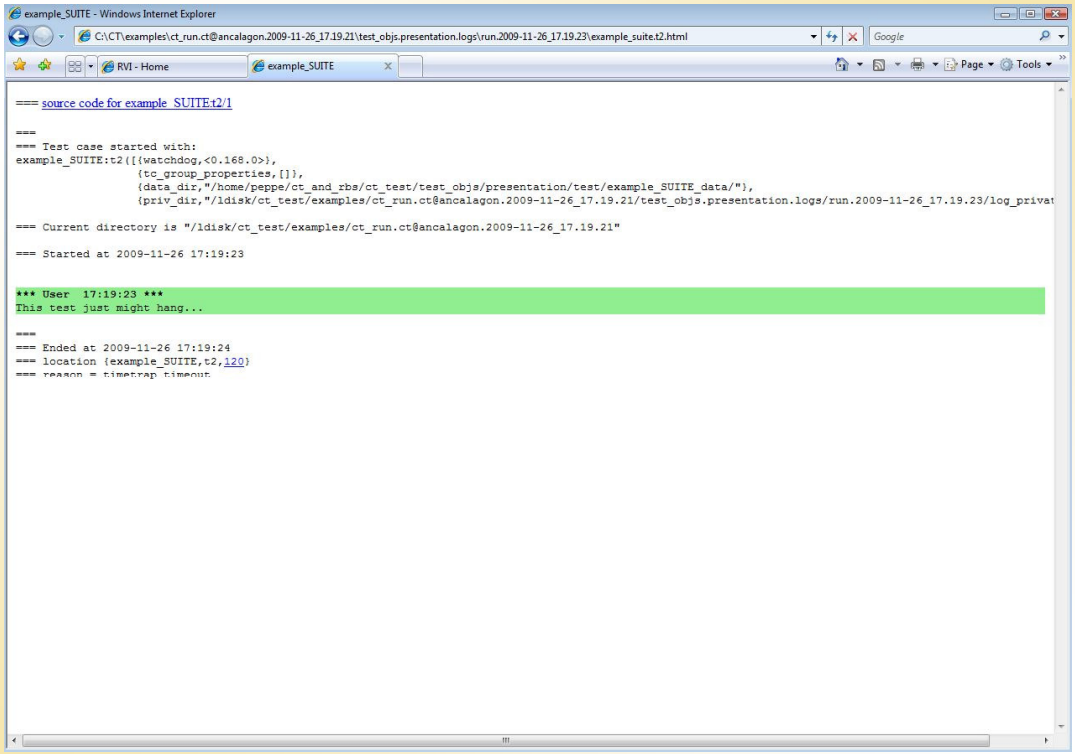
ERICSSON



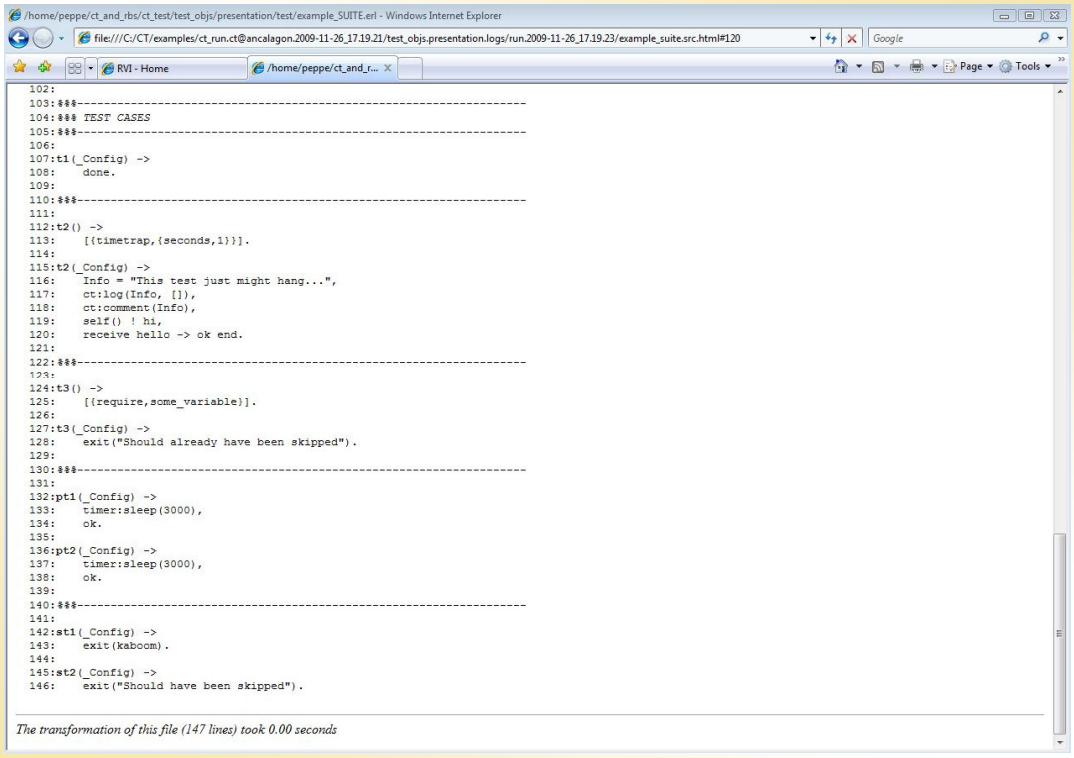
ERICSSON



ERICSSON



ERICSSON



ERICSSON

HTML stylesheets

- Optional Common Test feature
- CSS file for customizing user printouts.
- Category mapped to CSS selector.

Example of declaration:

```
<style>
div.ct_internal { background:lightgrey; color:black }
div.default     { background:lightgreen; color:black }
div.sys_config  { background:blue; color:white }
div.sys_state   { background:yellow; color:black }
div.error       { background:red; color:white }
</style>
```


ERICSSON

The Common Test Framework

6. Common Test i/f- and library modules

ERICSSON

Common Test modules

Common Test consists of the following i/f modules:

- `ct` - main user interface for the framework
- `ct_master` - support for large scale testing
- `ct_ftp` - CT interface to FTP client
- `ct_rpc` - CT interface to Erlang/OTP RPC
- `ct_telnet` - CT interface to Telnet client
- `unix_telnet` - ct_telnet callback for Unix host
- `ct_snmp` - CT interface to Erlang/OTP SNMP
- `ct_ssh` - CT interface to Erlang/OTP SSH/SFTP

ERICSSON

The `ct` module

The `ct` module provides the main interface for writing test cases. This includes e.g:

- Functions for executing test cases.
- Functions for printing & logging.
- Functions for checking and reading configuration data.
- Functions for terminating a test case with error reason.
- Functions for timetrap handling.

The Common Test Framework

7. Code coverage analysis

Code coverage analysis

- Measure code coverage when testing Erlang programs.
- Simple access to the *OTP Cover tool* - CT handles starting, compiling modules, analysing result, etc, automatically.
- Cover specification file to declare what modules to include.
- Posibility to import and export coverage data between tests.
- Code coverage results included with CT html logs.
- Start test with:

```
run_test -cover <CoverSpecFile>, or
ct:run_test([{cover,CoverSpecFile},…])
```

ERICSSON

Code coverage analysis

Example of a cover specification file:

```
{nodes, [n1@finwe,n2@aldor]}.
{import, [“cover0.data”]}.
{export, “cover1.data”}.
{level, overview}.
{incl_dirs_r, [“app1”,”app2”]}.
{excl_dirs, [“app1/priv”]}.
{excl_mods, [utils]}.
```


ERICSSON

The Common Test Framework

8. Test specifications

ERICSSON

Test specifications

- Flexible way to specify tests.
- A sequence (arbitrary number) of Erlang terms: *configuration terms* and *test specification terms*.
- Can be declared in a file or passed as a list.
- Enables skipping of test suites or cases.

ERICSSON

Test specification syntax

Config terms:

- `{init, InitOptions}`
- `{config, ConfigFiles}`
- `{userconfig, {CallbackMod, ConfigData}}`
- `{alias, DirAlias, Dir}`
- `{logdir, LogDir}`
- `{event_handler, EventHandlers[, InitArgs]}`
- `{cover, CoverSpecFile}`
- `{include, IncludeDirs}`
- `{multiply timetraps, N}`
- `{scale_timetraps, Bool}`

ERICSSON

Test specification syntax

Test terms:

- `{suites, DirRef, Suites}`
- `{cases, DirRef, Suite, Cases}`
- `{skip_suites, DirRef, Suites, Comment}`
- `{skip_cases, DirRef, Suite, Cases, Comment}`

ERICSSON

Test specification example

```
{logdir, "/home/test/logs"}.

{config, "/home/test/t1/cfg/config.cfg"}.
{config, "/home/test/t2/cfg/config.cfg"}.
{config, "/home/test/t3/cfg/config.cfg"}.

{alias, t1, "/home/test/t1"}.
{alias, t2, "/home/test/t2"}.
{alias, t3, "/home/test/t3"}.

{suites, t1, all}.
{skip_suites, t1, [t1B_SUITE,t1D_SUITE], "Not implemented"}.
{skip_cases, t1, t1A_SUITE, [test3,test4], "Irrelevant"}.
{skip_cases, t1, t1C_SUITE, [test1], "Ignore"}.
{suites, t2, [t2B_SUITE,t2C_SUITE]}.
{cases, t2, t2A_SUITE, [test4,test1,test7]}.
{skip_suites, t3, all, "Not implemented"}.
```

ERICSSON

Run using test specifications

In a UNIX shell:

```
run_test -spec <testspecs>
```

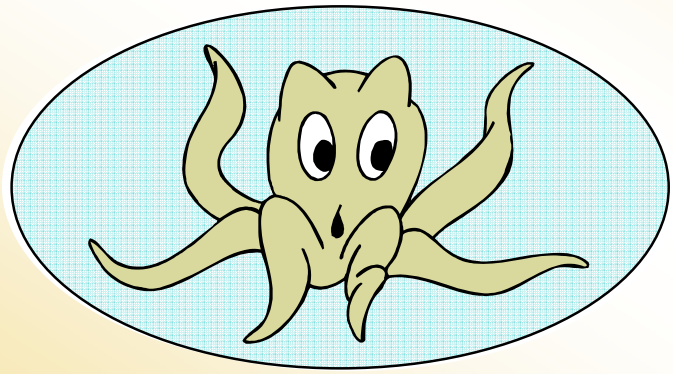
In an Erlang shell, use:

`ct:run_test/1` or `ct:run_testspec/1`.

ERICSSON

The Common Test Framework

9. Large Scale Testing



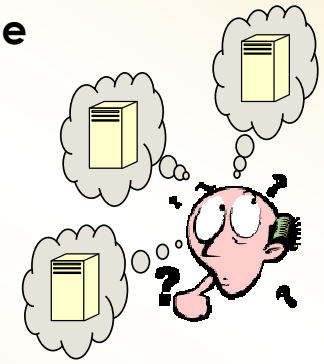
ERICSSON

Using Common Test for Large Scale Testing

Large scale automated testing typically requires running **multiple independent test sessions in parallel**.

This may be accomplished by running a number of Common Test nodes **on one or more hosts**, testing different target systems.

Configuring, starting and controlling the test nodes independently can be a cumbersome operation.



ERICSSON

Using Common Test for Large Scale Testing

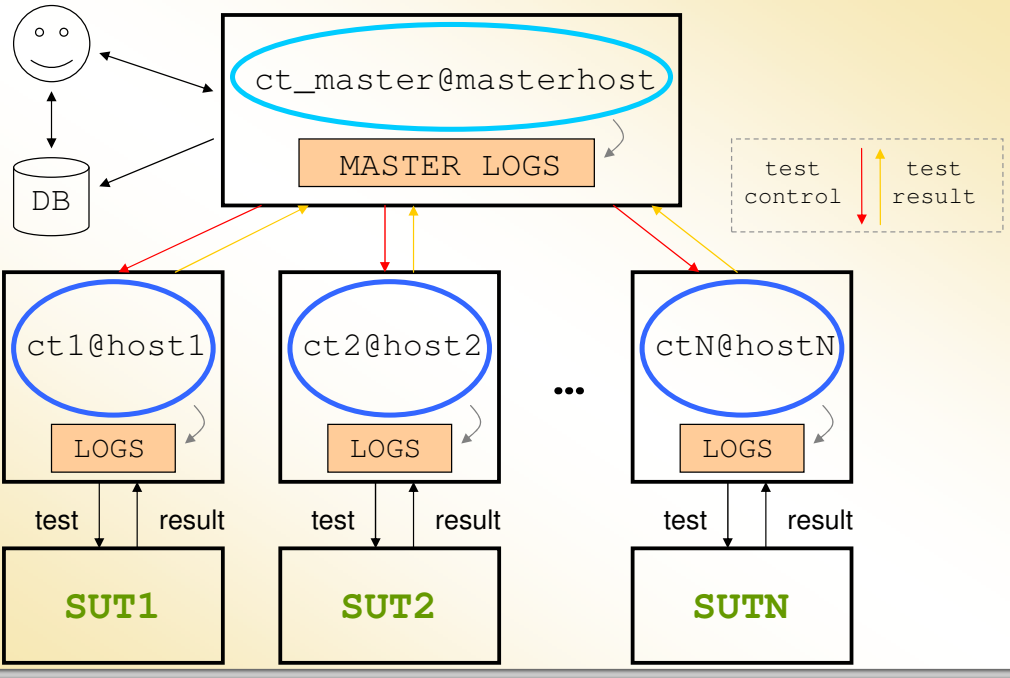
Common Test Master

A master test node component that aids automated Large Scale Testing.

CT Master handles central configuration and control in a system of distributed CT host nodes.

ERICSSON

Using Common Test for Large Scale Testing



ERICSSON

Using Common Test for Large Scale Testing

Common Test Master API module: `ct_master`

- Start tests with `ct_master:run/1` or `ct_master:run/3`.
- Use *test specifications* as input (test specifications for Large Scale Testing are compatible with specifications created for single host node tests - and the other way around!).
- Start slave nodes automatically (over ssh) on local or remote host.
- Add or remove host nodes dynamically at runtime.

ERICSSON

The Common Test Framework

10. Event handling

ERICSSON

Event handling

Plug in an *event handler* to receive event notifications continously during a test session.

Examples of notifications:

- when a test case starts and stops
- current count of succeeded, failed and skipped cases

Can be used e.g. to log progress and results on other format than HTML, implement test system supervision, or to save statistics to a database for report generation.

ERICSSON

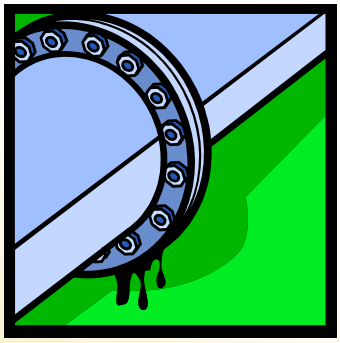
Event handling

- The CT event handler is based on the OTP event manager concept and *gen_event* behaviour. The CT user implements the event handler callback module, which should include `ct_event.hrl`.
- The event handler receives `#event{name, node, data}` records from the CT server.
- Event handlers (any number) can be plugged in on regular CT nodes as well as on a CT Master node.

ERICSSON

The Common Test Framework

11. In the pipeline...



ERICSSON

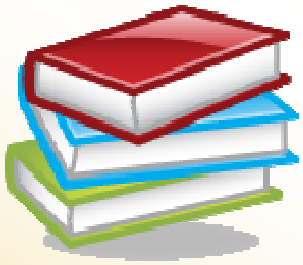
In the pipeline...

- Generators and parameterized test cases
- State machines as test suites and/or test cases
- Extended and improved test case group functionality
- New powerful Large Scale Testing features
- New APIs for integrating CT with tools and frameworks
- (Better) Quickcheck integration
- Events as alternative logging mechanism
- ...

ERICSSON

The Common Test Framework

12. Documentation



ERICSSON

Documentation

file:///<OTP_ROOT>/doc/index.html -> Tool Applications -> common_test

http://www.erlang.org/doc/index.html -> Tool Applications -> common_test

Common Test User's Guide

Common Test Reference Manual

Common Test Man pages