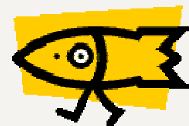
The Common Test Framework

ERICSSON

TOPICS

1. Overview – what is Common Test
2. Test suites and test cases
3. Configuration files
4. Test results and logs
5. Common Test i/f- and library modules
6. Test execution
7. Code coverage analysis
8. Test specifications
9. Large Scale Testing
10. Event handling
11. Test case groups
12. In the pipeline
13. Documentation



ERICSSON

The Common Test Framework

1. Overview – what is Common Test?

ERICSSON

Common Test

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

• A portable test server for black-box testing (function and system testing) target nodes of any type.

• A practical tool for white-box testing OTP applications and Erlang programs.

Common Test provides:

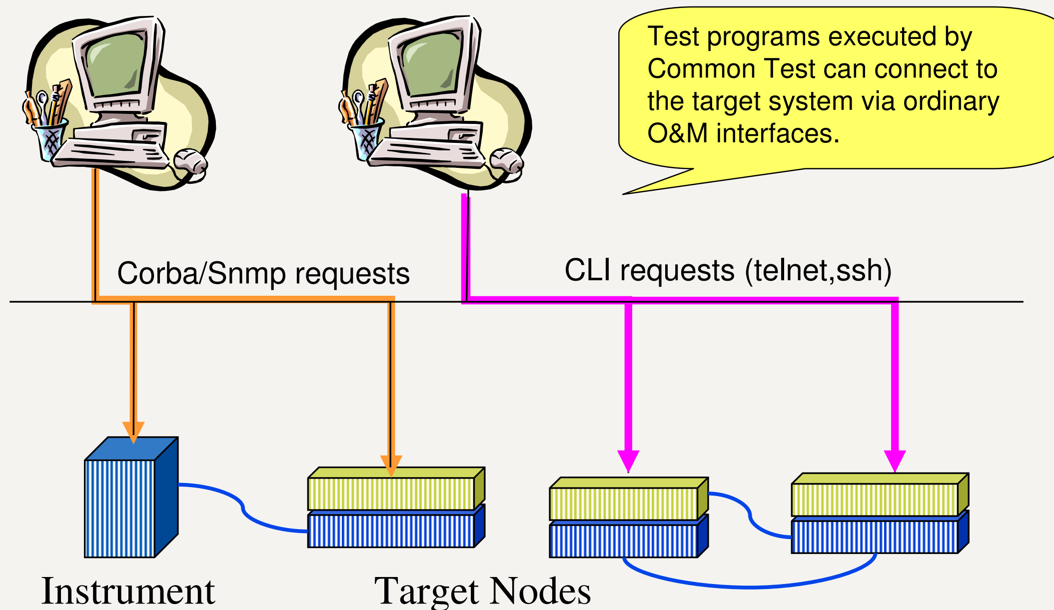
- Possibility to run test suites automatically on local and remote targets.
- HTML progress and result logs.
- Test suite templates and support libraries.
- Support for large scale testing.
- Event handler interface for integration with other programs.

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.

Testing through O&M interfaces



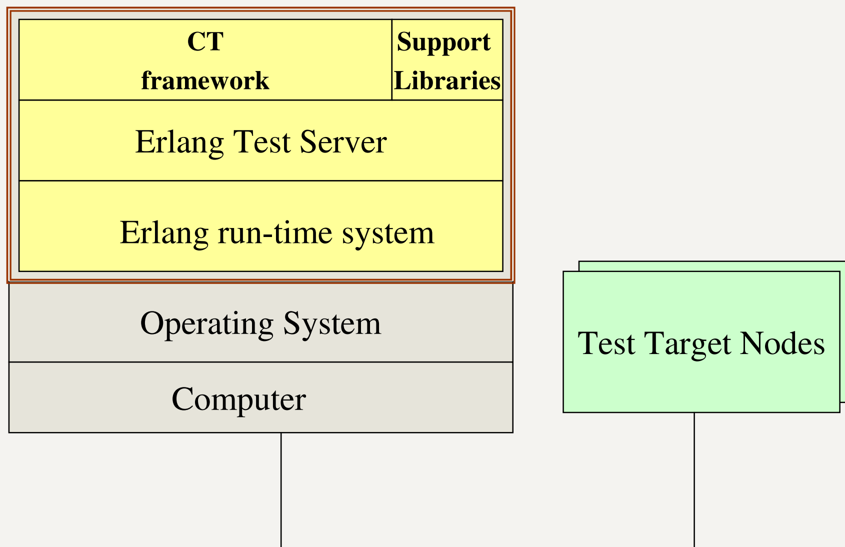
ERICSSON

Parallel connections

The possibility to handle parallel connections - many of them if necessary - is an important strength of Common Test!

ERICSSON

Common Test implementation structure



ERICSSON

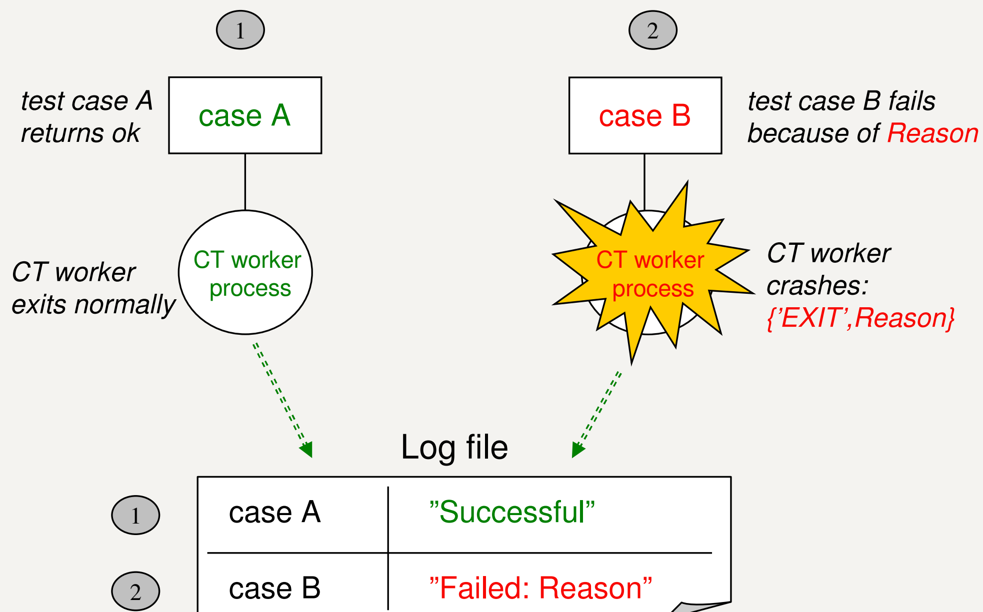
Support libraries

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).

ERICSSON

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 (cont.)

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

```
?SUCCESS = perform_operation(Conn, Op)
```

- **Erlang, a small general-purpose language:**
  - = Short time to learn enough to start working with test suite development and maintenance (especially with existing programming-knowledge and experience).
  - = Great flexibility when writing test programs.

Parallel execution

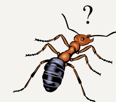
A powerful feature of Common Test is its support for (and use of) parallel execution and communication.

This is a very important and useful test server characteristic, and is something Common Test "gets for free", being an Erlang application.

This provides for, e.g:

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

Parallel execution (cont.)



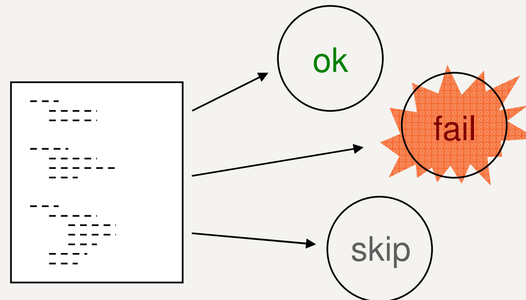
Parallel execution of code is used in many different scenarios in the Common Test based framework:

- Open and control arbitrary numbers of connections to arbitrary numbers of SUTs and instruments and handle traffic on these connections concurrently (asynchronously, or synchronously without blocking traffic on other connections).
- Use concurrency in test cases at will to solve any parallel problems, or to increase performance of operations.
- Parallel execution of independent test cases (to save time or to verify that parallel stimuli is handled correctly by the SUT).
- Common Test relies on concurrency (and error handling mechanisms) for various operations such as:
  - Running test cases on dedicated processes to avoid dependencies.
  - Setting timers and generating timeouts for executing or hanging test cases.
  - Determining the success or failure of test cases.

ERICSSON

The Common Test Framework

2. Test suites and Test cases



ERICSSON

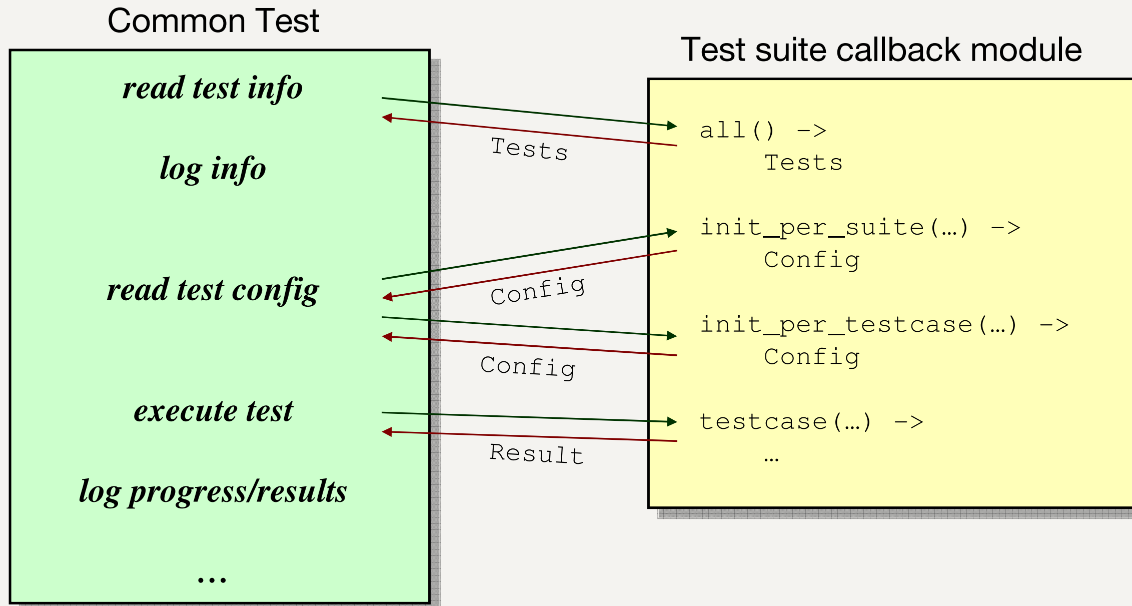
The test suite

The test suite is a callback module that must comply with a defined test server interface. This is documented in the `common_test` part of the Common Test Reference Manual.

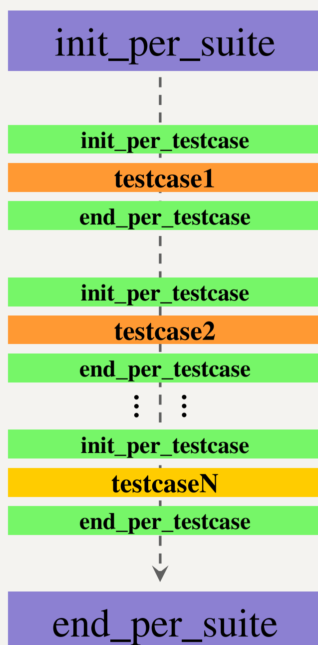
For example, a test suite must export the function `all/0` which returns a list of all test cases in the module.

The header file `ct.hrl` must be included in all test suite files. (This header file may also be included in library modules that the test suites use).

The test suite callback module



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

- The test case function takes one argument, `Config`, which contains **runtime configuration data** (such as `priv_dir` and `data_dir`).

```
testcase_name(Config) ->
  …
  ok.
```

- In the functions `init_per_suite` and `init_per_testcase` it is possible to add your own configuration data (`{Key,Value}` tuples) to `Config`.

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

- All config items can be extracted using the **`?config`** macro, e.g:

```
PrivDir = ?config(priv_dir, Config)
```

Test suite info function

The *test suite info function*, `suite/0`, can be used in a test suite module to set the default values for various properties and perform initial assertions.

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

Note: Property values set by `suite/0` can be overridden by individual testcases if necessary.

`timetrap` specifies how long a test case may run before it's aborted by the test server (the default time limit is 30 min). The value `infinity` disables the timetrap.

`require` is used to check data in configuration files (details later).

Other options that may be specified are: `userdata`, `stylesheet`, `silent_connections`. (See the CT User's Guide for info).

Test case info function

For each test case function there can be an additional *test case info function* which has the **same name** as the test case, but no arguments.

The test case info function returns key-value tuples that specify various properties regarding the test case or perform assertions.

Can be used to override properties set by the `suite/0` function.

Predefined attributes are: `timetrap`, `require`, `userdata`, `stylesheet` and `silent_connections`. Other optional `{Key,Value}` tuples (where `Key` is always an atom) may be added later.

```
testcase_name() ->
       [ % Max time for test case execution before abortion.
         {timetrap, {seconds,60}},
         % Configuration variables required by the test case
         {require, myvariable}
       ].
```

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.
```

Skipping test cases

It is possible to skip certain test cases, for example if you know beforehand that a specific test case fails. This might be functionality which isn't yet implemented, a bug that is known but not yet fixed or e.g. some functionality which isn't applicable for a specific version of the target software.

There are several different ways to state that a test case should be skipped:

• Returning `{skip,Reason}` from the `init_per_testcase/2` or `init_per_suite/1` functions.

• Returning `{skip,Reason}` from the test case function (which means the function is called and that the author needs to make sure the actual test is not executed).

When a test case is skipped, it will be noted as SKIPPED in the HTML log.

Skipping test cases (cont.)

Test cases can also be skipped because something went wrong:

• If `init_per_suite` fails, all test cases in the test suite will be skipped, including `end_per_suite`.

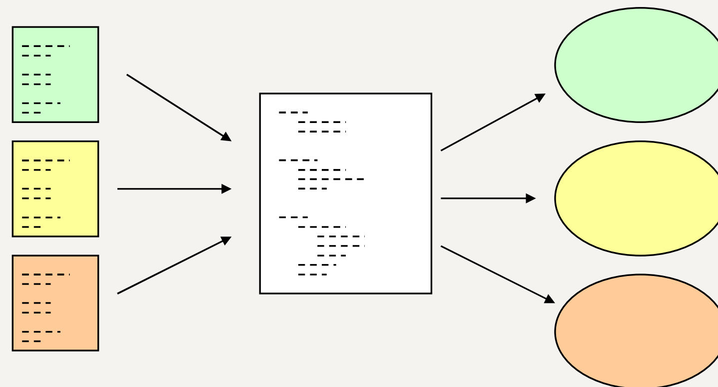
• If `init_per_testcase` crashes, the test case itself is skipped including `end_per_testcase`.

• If *required* configuration variables are not found in any of the configuration files, the test case is skipped (only true for tests performed in *suite/0* or the *test case info* function).

ERICSSON

The Common Test Framework

3. Configuration files



ERICSSON

Configuration files

The Common Test framework uses configuration files to describe data related to a **test** or a **system under test**.

Test/system configuration data makes it possible to change properties without having to modify test suites. Examples of test/system configuration data:

- Addresses to the test plant or other instruments
- Identities
- Names of files needed by the test
- Names of programs that should be executed by the test

ERICSSON

Configuration files (cont.)

A configuration file can contain any number of elements on the form:

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

where

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

(`Key` is the name of the *configuration variable*).

ERICSSON

Configuration variables and require

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

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 case of the first two, the test suite or test case is aborted if the require statement fails. `ct:require/[1,2]`, however, **returns** `ok` or `{error,Reason}` and does not automatically abort the 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), …
```


ERICSSON

Using configuration data for opening connections

There are two different methods for opening a connection using the support modules in Common Test (e.g. `ct_ftp`, `ct_ssh` or `ct_telnet`):

1. Using a configuration target name (an alias).

   When a target name is used for referencing the configuration data for the connection, the same name may be used as connection reference in the subsequent calls (also for closing the connection). It's only possible to have one open connection for each name.

2. Using the configuration variable name (the key).

   In this case the returned handle must be used as reference in all subsequent calls (also for closing the connection). With this method it is possible to open multiple connections identified by the same configuration data.

The Common Test Framework

4. 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.

An HTML file (`index.html` in the `ct_run` directory), shows you a summary after every test run. From this page you can access an HTML file (`suite.log.html`) that shows a status overview of the test suites and test cases. The latter file has a link to every test case log file.

After every test run, a link to the test summary file is stored in a history file (`all_runs.html` in the working directory).

ERICSSON

Test Results - Windows Internet Explorer

C:\CT\examples\index.html

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**			

Copyright © 2009 Open Telecom Platform
Updated: Thu Nov 26 2009 17:20:07

ERICSSON

All test runs in current directory - Windows Internet Explorer

C:\CT\examples\all_runs.html

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

Copyright © 2009 Open Telecom Platform
Updated: Thu Nov 26 2009 17:20:07

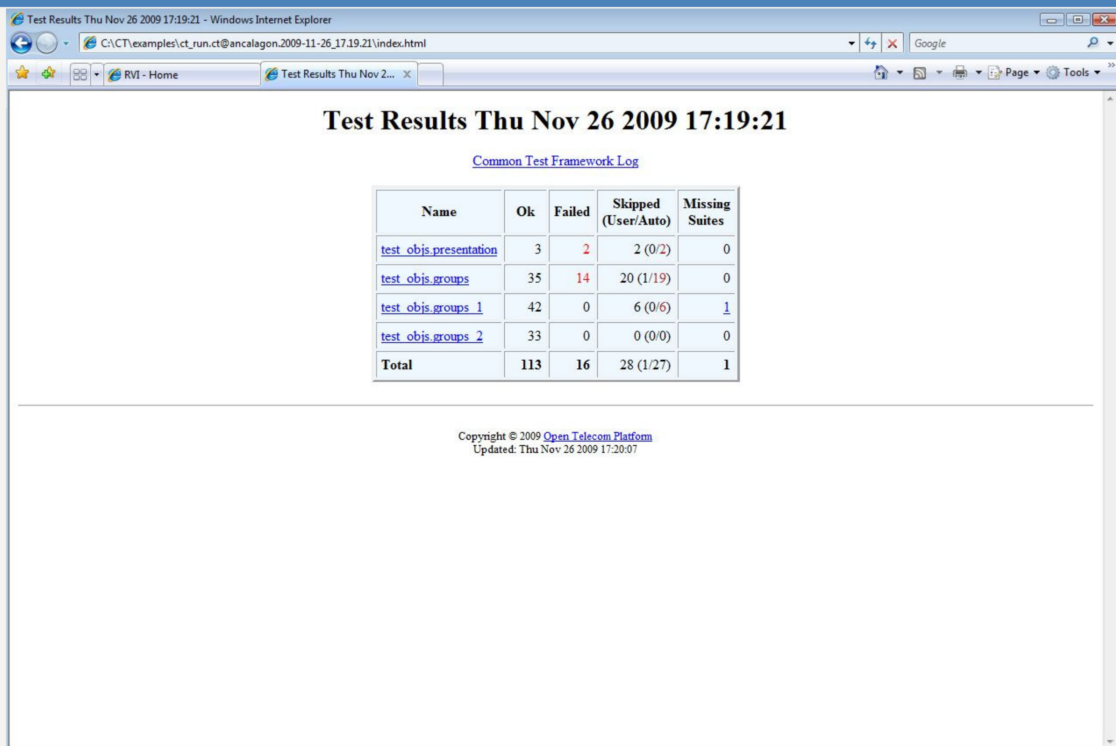
ERICSSON

The test result index page

This file gives a short overview of all individual tests performed in the same test run. The test names follows this convention:

TopLevelDir.TestDir (all suites in TestDir executed)
TopLevelDir.TestDir:suites (specific suites were executed)
TopLevelDir.TestDir.Suite (all cases in Suite executed)
TopLevelDir.TestDir.Suite:cases (specific test cases were executed)
TopLevelDir.TestDir.Suite.Case (only Case was executed)

ERICSSON



Test Results Thu Nov 26 2009 17:19:21

Common Test Framework Log

Name	Ok	Failed	Skipped (User/Auto)	Missing Suites
test_objs.presentation	3	2	2 (0/2)	0
test_objs.groups	35	14	20 (1/19)	0
test_objs.groups_1	42	0	6 (0/6)	1
test_objs.groups_2	33	0	0 (0/0)	0
Total	**113**	**16**	**28 (1/27)**	**1**

Copyright © 2009 Open Telecom Platform
Updated: Thu Nov 26 2009 17:20:07

ERICSSON

Test "test_objs.presentation" results - Windows Internet Explorer

C:\CT\examples\ct_run.ct@ancalagon.2009-11-26_17.19.21\test_objs.presentation.logs\run.2009-11-26_17.19.23\suite.log.html

Results from test "test_objs.presentation"

Test started at 2009-11-26 17:19:23

Host:
Run by peppe on ancalagon
Used Erlang 5.7.4 in `/usr/local/otp/releases/sles10_64_R13B03_patched`.

Full textual log
Coverage log

Suite contains 7 test cases.

Num	Module	Case	Log	Time	Result	Comment
	example_SUITE	init_per_suite	< >	0.000s	Ok	
1	example_SUITE	t1	< >	0.000s	Ok	
2	example_SUITE	t2	< >	1.000s	FAILED	{timetrap_timeout,{example_SUITE,t2,120}} This test just might hang...
	example_SUITE	init_per_group	< >	0.000s	Ok	parallel group starts
3	example_SUITE	pt1	< >	3.005s	Ok	
4	example_SUITE	pt2	< >	3.003s	Ok	
	example_SUITE	end_per_group	< >	0.000s	Ok	parallel group ends
	example_SUITE	init_per_group	< >	0.000s	Ok	sequence group starts
5	example_SUITE	st1	< >	0.000s	FAILED	{example_SUITE,st1,143} kaboom
6	example_SUITE	st2	< >	0.000s	SKIPPED	{failed,{example_SUITE,st1}}
	example_SUITE	end_per_group	< >	0.000s	Ok	sequence group ends
7	example_SUITE	t3	< >	0.000s	SKIPPED	{require_failed,{not_available,some_variable}}
	example_SUITE	end_per_suite	< >	0.000s	Ok	
	TOTAL			4.290s	FAILED	3 Ok, 2 Failed of 5

ERICSSON

example_SUITE - Windows Internet Explorer

C:\CT\examples\ct_run.ct@ancalagon.2009-11-26_17.19.21\test_objs.presentation.logs\run.2009-11-26_17.19.23\example_suite.t2.html

=== source code for example_SUITE:t2/1

```
===
=== Test case started with:
example_SUITE:t2([{watchdog,<0.168.0>},
                  {tc_group_properties,[]},
                  {data_dir,"/home/peppe/ct_and_rbs/ct_test/test_objs/presentation/test/example_SUITE_data/"},
                  {priv_dir,"/ldisk/ct_test/examples/ct_run.ct@ancalagon.2009-11-26_17.19.21/test_objs.presentation.logs/run.2009-11-26_17.19.23/log_privat

=== Current directory is "/ldisk/ct_test/examples/ct_run.ct@ancalagon.2009-11-26_17.19.21"

=== Started at 2009-11-26 17:19:23


*** User  17:19:23 ***
This test just might hang...

===
=== Ended at 2009-11-26 17:19:24
=== location {example_SUITE,t2,120}
=== reason = timetrap timeout
```

ERICSSON

/home/peppe/ct_and_rbs/ct_test/test_objs/presentation/test/example_SUITE.erl - Windows Internet Explorer

file:///C:/CT/examples/ct_run.ct@ancalagon.2009-11-26_17.19.21/test_objs.presentation.logs/run.2009-11-26_17.19.23/example_suite.src.html#120

```
102:
103:###-----------------------------------------------------------------
104:### TEST CASES
105:###-----------------------------------------------------------------
106:
107:t1(_Config) ->
108:    done.
109:
110:###-----------------------------------------------------------------
111:
112:t2() ->
113:    [{timetrap,{seconds,1}}].
114:
115:t2(_Config) ->
116:    Info = "This test just might hang...",
117:    ct:log(Info, []),
118:    ct:comment(Info),
119:    self() ! hi,
120:    receive hello -> ok end.
121:
122:###-----------------------------------------------------------------
123:
124:t3() ->
125:    [{require,some_variable}].
126:
127:t3(_Config) ->
128:    exit("Should already have been skipped").
129:
130:###-----------------------------------------------------------------
131:
132:pt1(_Config) ->
133:    timer:sleep(3000),
134:    ok.
135:
136:pt2(_Config) ->
137:    timer:sleep(3000),
138:    ok.
139:
140:###-----------------------------------------------------------------
141:
142:st1(_Config) ->
143:    exit(kaboom).
144:
145:st2(_Config) ->
146:    exit("Should have been skipped").
```

The transformation of this file (147 lines) took 0.00 seconds

ERICSSON

Printouts from test cases

CT provides the following functions for printing information from a **test case**:

`ct:comment/1` % print string in comment field in HTML log file

`ct:log/[1,2,3]` % print to test case log file

`ct:print/[1,2,3]` % print to console

`ct:pal/[1,2,3]` % print both to log and console

Note that printouts to stdout, e.g. with io:format/2, are directed to the test case log file during the test execution.

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

5. Common Test i/f- and library modules

Common Test modules

Common Test consists of the following interface 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

from v1.4
(OTP R13A)

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 reading configuration data.
- Functions for terminating a test case with error reason.
- Functions for adding comments to the HTML overview page.

ERICSSON

The Common Test Framework

6. Test execution

ERICSSON

Test execution

The script *run_test* can be used for starting tests from a **Unix command line**:

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

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

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

Examples:

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

$ run_test -config $CFGDIR/node.cfg -suite $TESTDIR/objY_test/objY_setup_SUITE
```

ERICSSON

Running from the Unix 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.
- `-silent_connections [conn_types]`, tells CT to suppress printouts for specified connections (e.g. telnet).
- `-stylesheet`, for installing a CSS file.
- `-cover`, for performing code coverage tests.
- `-include`, to add include directories for test suite compilation.

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

ERICSSON

Running 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"}]).
```

Other (less flexible) test execution functions:

- `ct:run(Dir)`
- `ct:run(Dir, Suite)`
- `ct:run(Dir, Suite, Cases)`
- `ct:step(Dir, Suite, Case)`

Running `ct` in *interactive shell mode*

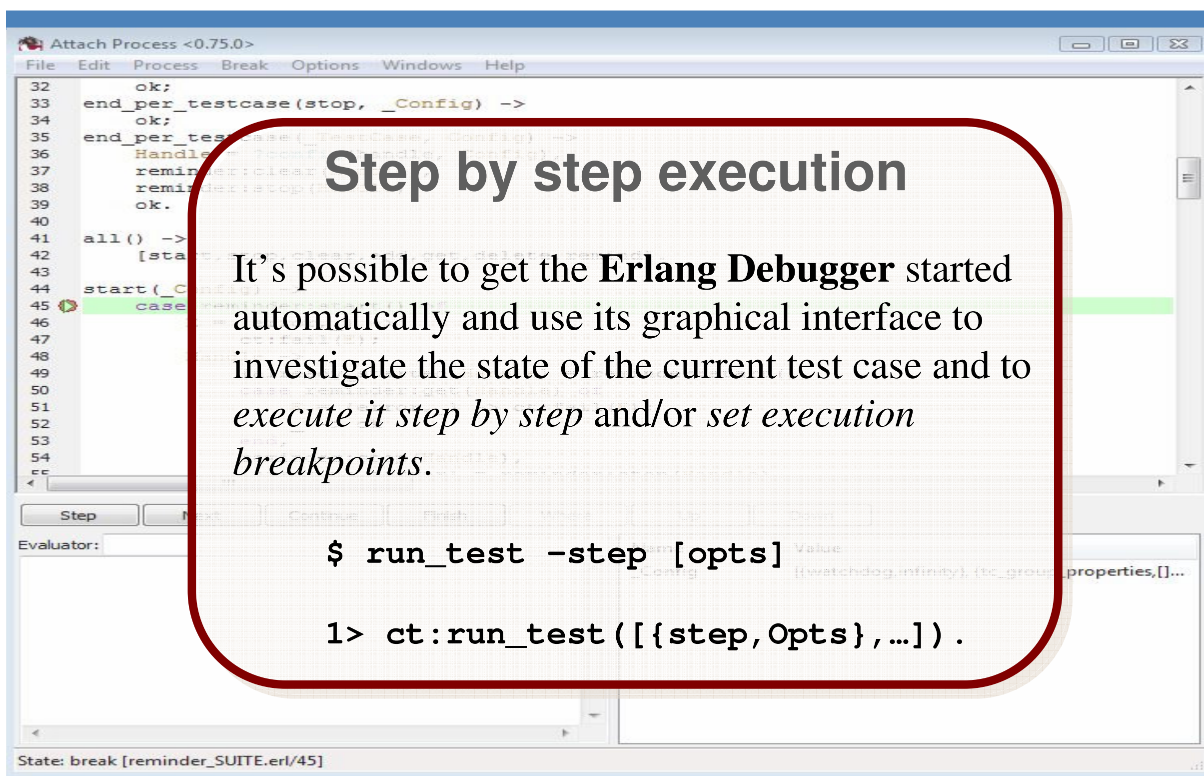
You can start Common Test in an interactive shell mode. In this mode it is possible to evaluate test expressions in an Erlang shell instead of having CT execute tests automatically. Configuration data can be installed and used (required and retrieved) in the shell just like in a test suite.

```
$> run_test -shell

$> run_test -shell -config <configfilenames>
```

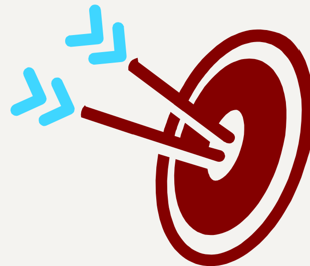
If no config file is specified by means of "*run_test –config*", you must explicitly run `ct:install/1` to install the config data you need for your tests (if any).

In an Erlang shell, the interactive CT shell mode can be toggled on/off by means of calling the `ct:start_interactive/0` and `ct:stop_interactive/0` functions. You can not start normal automated tests in a shell that has the interactive CT mode enabled!



The Common Test Framework

7. Code coverage analysis



Code coverage analysis

- Measure code coverage when testing Erlang applications.
- 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 (cont.)

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:

- `{node, NodeAlias, Node}`
- `{config, ConfigFiles}`
- `{config, NodeRef, ConfigFiles}`
- `{alias, DirAlias, Dir}`
- `{logdir, LogDir}`
- `{logdir, NodeRef, LogDir}`
- `{event_handler, EventHandlers}`
- `{event_handler, NodeRef, EventHandlers}`
- `{cover, CoverSpecFile}`
- `{cover, NodeRef, CoverSpecFile}`

ERICSSON

Test specification syntax (cont.)

Test terms:

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

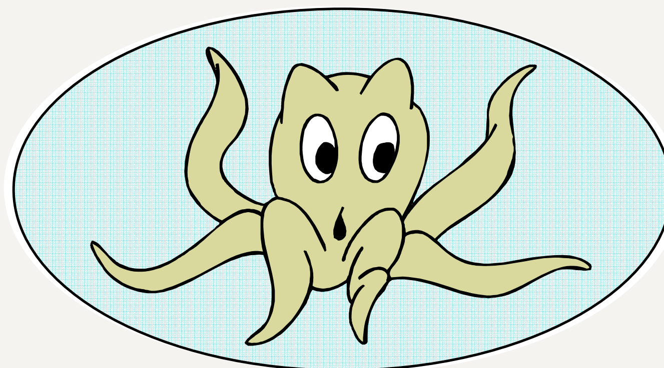
In an Erlang shell, use:

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

ERICSSON

The Common Test Framework

9. Large Scale Testing

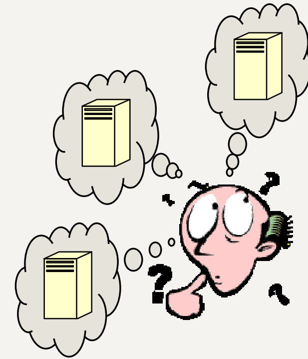


Using Common Test for Large Scale Testing

Large scale automated testing requires running multiple independent test sessions in parallell.

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.



Using Common Test for Large Scale Testing

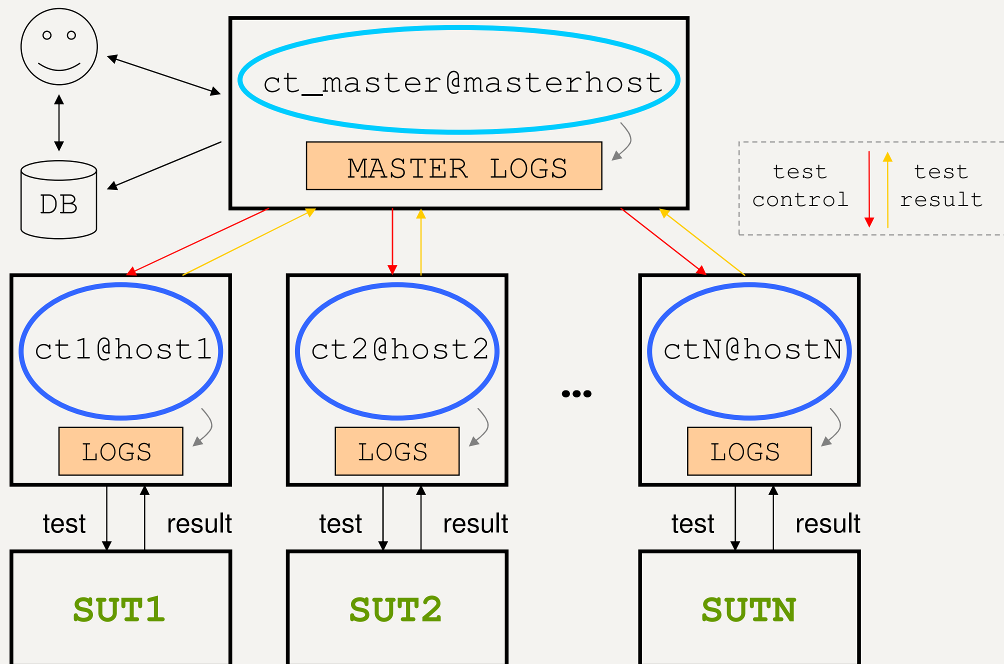
Common Test Master

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!)
- 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 (cont.)

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`.

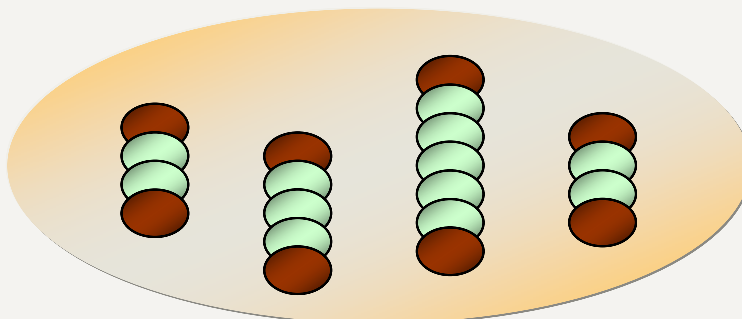
The event handler receives `#event{name, node, data}` records from the CT server. The events are documented in the *Event Handling* chapter in the User's Guide (and also in the `ct_event.erl` module).

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. Test case groups



ERICSSON

Test case groups

- Set 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

Test case groups (cont.)

Groups are declared with function `groups/0`:

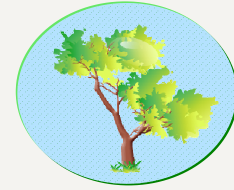
```
groups() ->
   [{NameOfGroup1, Properties1, TestCasesAndSubGroups1},
    {NameOfGroup2, Properties2, TestCasesAndSubGroups2},
     …
    {NameOfGroupN, PropertiesN, TestCasesAndSubGroupsN}].
```

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

```
all() -> TestCasesAndGroups

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

Test case groups (cont.)



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},
   …].
```


Test case groups (cont.)

Test case groups example:

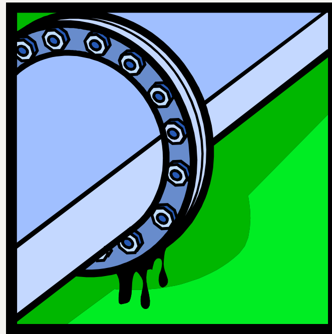
```
...
init_per_group(group1, Config) ->
        init_group1(Config);
init_per_group(group2, Config) ->
        init_group2(Config);
init_per_group(_GroupName, Config) ->
        Config.
end_per_group(_GroupName, _Config) ->
        ok.

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}].
...
```

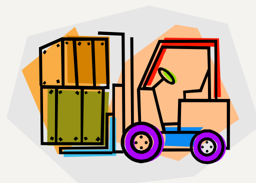
The Common Test Framework

12. In the pipeline...



In the pipeline…

- Generators and parameterized test cases
- State machines as test suites and/or test cases
- (Better) Quickcheck integration
- Events as alternative logging mechanism
- …



ERICSSON

The Common Test Framework

13. 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