

MIN Test Framework

Self Learning Material

Sampo Saaristo

Copyright © 2008 Nokia.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

What is MIN

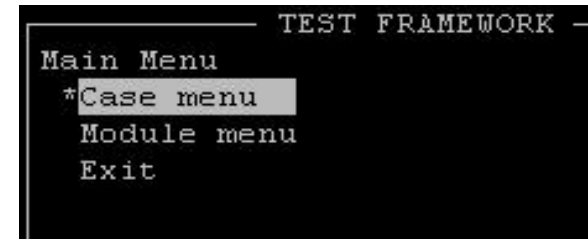
- **Advanced Test Harness for testing Linux non-UI components**
- **Toolkit for test case implementation and test cases execution**
- **Framework for implementing test cases in separated test modules**
- **Not an automatic test case creator but allows to concentrate to the actual test case implementation**
- **MIN separates test case implementation and test environment**
 - **Test cases implemented to separated test module**
 - **Test module implements common test module API**
 - **Test modules are running independently in own processes**

What is MIN (continue)

- Easy to use
 - Usability has been one of the key considerations
 - Simple and flexible interface to test modules
 - Plenty of optional advanced features though
- Write once, test everywhere
 - E.g. test cases made during development phase can be used in system testing, automatic release testing, etc

What MIN provides

- User Interfaces for test cases execution
 - MIN ConsoleUI
 - MIN Command line interface
 - (MIN External Interface)
- Test engine for test module management and generating test reports.
- Common API for test modules
- Tools e.g. MIN Parser, MIN Logger and Test Module Wizard for helping test case implementation

A screenshot of a terminal window showing the 'TEST FRAMEWORK' main menu. The menu options are 'Main Menu', '*Case menu', 'Module menu', and 'Exit'. The '*Case menu' option is highlighted with a grey background.

```
— TEST FRAMEWORK —  
Main Menu  
*Case menu  
Module menu  
Exit
```

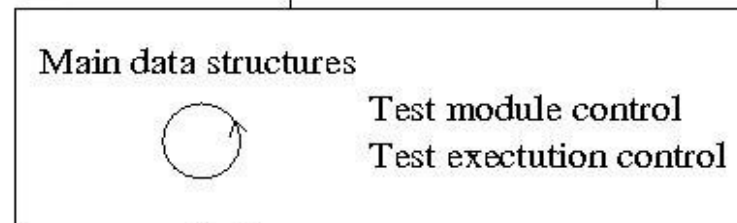
MIN Architecture

CONSOLEUI



MIN ENGINE

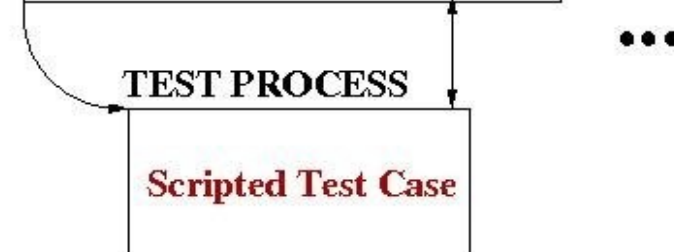
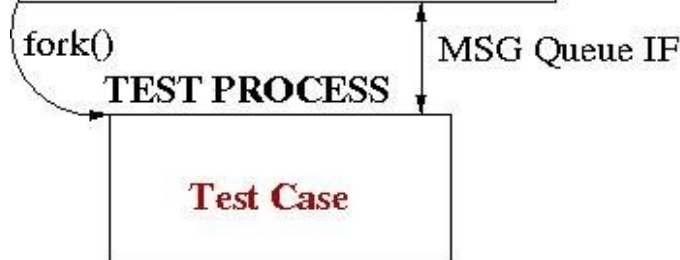
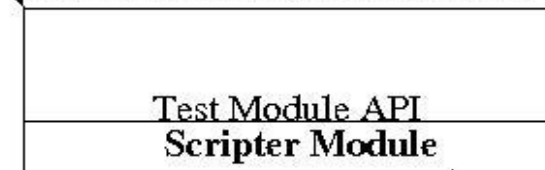
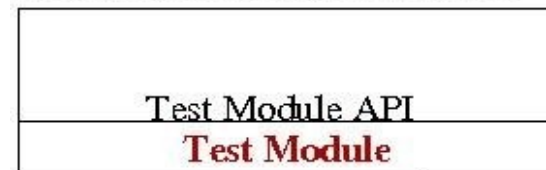
External interface to test automation



MSG Queue IF

TEST MODULE CONTROLLER

TEST MODULE CONTROLLER



MIN Features

- Error and crash handling
- Concurrent test case execution
- Test module template wizard (createtestmodule)
- File parsing with MIN Parser
- Logging facilities for test modules with MIN Logger
- Scripted test cases with Test Scripter
- Test case combining with Test Scripter
- xUnit style test cases with MINUnit
- Test case synchronization with Event System
- Test case grouping by creating test sets with MIN Console UI
- Support for test cases written in Lua
- Support for test cases written in Python
- Support for stress/interference testing
- Support "make check" target for compiling/executing MIN tests

MIN requirements and building

- Pre requirements
 - Linux operating system
 - MIN release
 - Test Module(s)
- MIN build process
 - Decompress MIN release
 - Use dpkg-buildpackage, which generates min and min-dev Debian packages.
 - Use dpkg --install command to install the package(s)
 - OR
 - Decompress MIN release
 - Use build.sh script to compile MIN
 - Use 'make install' to install MIN

Task: Setting Up & Demomodule (cont')

- MIN start-up
 - sminDemoModule is installed as default, so it's loaded automatically
 - Start MIN:
 - \$ min
 - Check Modules menu to see that minDemoModule is loaded
 - Start test case Demo_1 (Case menu->Start new case->Demo_1)
 - Start multiple test cases
 - See execution results from Passed, Failed and Aborted/Crashed cases

ConsoleUI

- ConsoleUI is text based UI running in shell
- Provides good test case controlling and monitoring possibilities
 - Test case starting, pausing, resuming and aborting
 - Output from test cases
 - Executed test cases are listed in different menus according to their execution status and result
 - Allows forming a group of test cases and running them either sequentially or in parallel

```
— TEST FRAMEWORK —
Main Menu
*Case menu
Module menu
Exit
```

```
— TEST FRAMEWORK —
Start new case
*io o&c
iowr2
iowr&rd
iodeltwice
Test Case 1
```

```
— TEST FRAMEWORK —
io o&c
*View output

Result info: Passed
Started: 12:05:56 PM
Completed: 12:05:56 PM
```

Command line Interface

- MIN command line interface can be used for running test cases when UI is not needed or it cannot be used
- When starting MIN without the console UI all test cases in all test modules which are configured to MIN initialization file (min.conf) are executed sequentially
- MIN can be started in cuiless mode by command:
- *\$ min -console*
- MIN can also made to omit the module definitions in min.conf(s) and execute the cases from a test module specified on the command line with switch *-execute* (or *-x*).
- *\$ min -console -execute:/path/to/testmodule.so[:/path/to/configuration_file.cfg]*

MIN Engine

- Test Engine component manages the test case execution
- Test Engine uses an initialization file (min.conf), which defines e.g. Test Module Controller and test modules location.
- User can define ModSearchPath(s) for MIN.
- User can define test module absolute path, or if path is configured to ModSearchPath only module name is needed.

```
[Engine_Defaults]
TmcBinPath=/usr/bin/tmc
ModSearchPath=/usr/lib
ModSearchPath=$HOME/.min
[End_Defaults]
[New_Module]
ModuleName=/usr/lib/minDemoModule.so
[End_Module]
```

MIN test module controller and test modules

- Test layer consists of following components:
 - Test Module Controller(s)
 - Test Module(s)
 - MIN Scripter
 - Tools
 - MIN Parser
 - MIN Logger
 - MIN Event System

Test Module Controller

- Test Module Controller = TMC
- Provides “sand box” for test module
- For each test module, there exists one TMC
- TMC gets requests from the Test Engine
- TMC handles requests itself or calls the corresponding function from the test module

Test Module

- Test module implements test cases
- Test module implements Test Module API
 - No other requirements or restrictions
 - Allows to use old code as basis and write complex test cases
- Usually there is one module that tests one component or feature
 - BT test module
 - Camera test module
 - Audio test module
 - Networking test module
- Test modules are libraries loaded by TMC
- Currently four different test module types are supported in MIN:
 - Hardcoded
 - normal
 - MINUnit
 - Test class (test scripter)

Test Module Configuration File

- Optional
- Delivered to test module during test module initialization
- Set via MIN initialization file (min.conf)
- Test module may use initialization file to get some common, test case independent, information
 - E.g. debug level, log file name, dial-up phone number, hardware/component/environment initialization settings, priority of the test cases run, etc.
- Content is determined by test module, i.e. MIN just passes the filename to test module

Test case

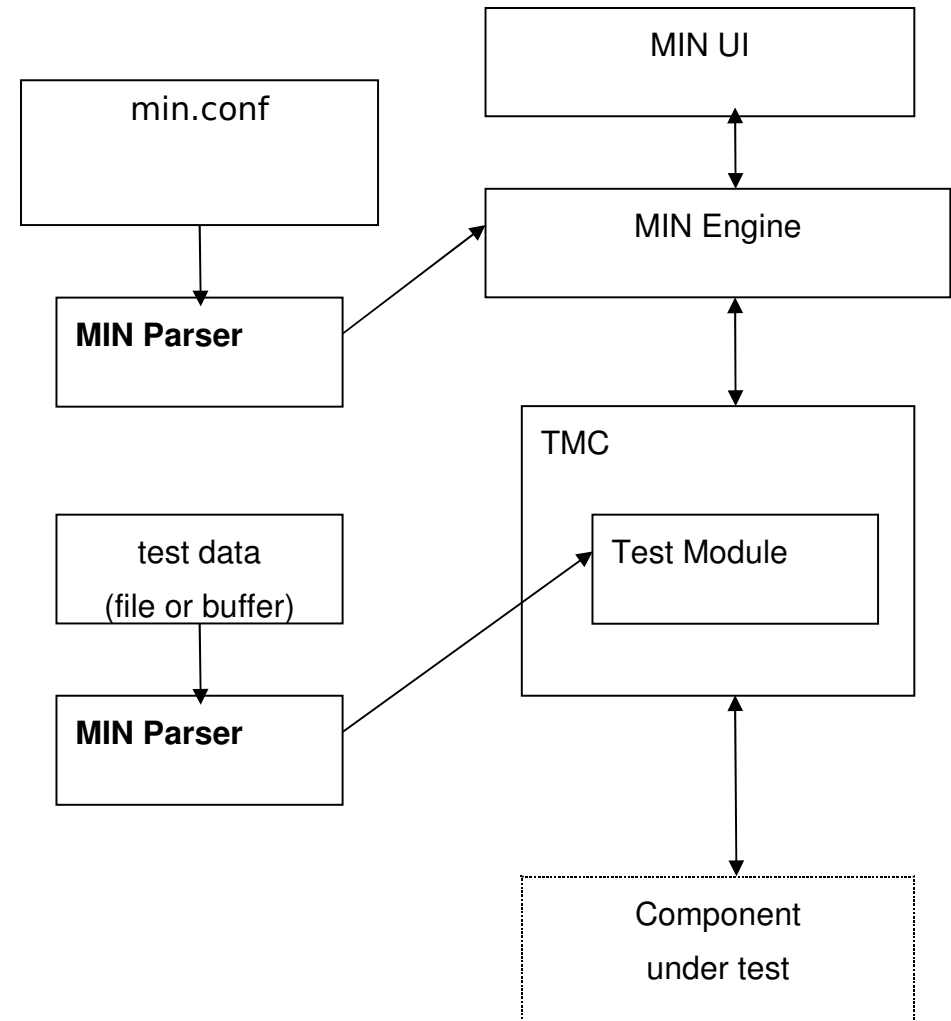
- Test case is a single test that tries to verify that something operates as specified
- Test case returns a result after completion
 - Result code
 - TP_CRASHED
 - TP_TIMEOUTED
 - TP_PASSED
 - TP_FAILED
 - TP_NC
 - TP_LEAVE
 - String containing additional description about failure
 - Shown on consoleUI

Test Case File

- Can be used by test module to pass test parameters to test cases
- Test case file name is delivered to test module when test cases are querieded and executed
- Multiple test case files are supported concurrently
- Set via MIN initialization file
- Content is determined by test module, i.e. MIN just passes the filename to test module
 - Existing test case definitions can be used conveniently

MIN Parser

- Easy-to-use parser for test modules
 - E.g. for initialization file and test case file parsing
- Support for hierarchical parsing
 - Section
 - Subsection
 - Line
 - String
 - Character
 - Integer



Example: MIN Parser

```
int parser_example () {
    MinSectionParser* msp = INITPTR;
    MinItemParser* mip = INITPTR;
    char* string = INITPTR;

    /* Lets create MinParser */
    MinParser* mp = mp_create( "/tmp" /* path to the file */
                              , "file.txt" /* file to be parsed */
                              , ENoComments); /* indicates how to
                                                treat comments */
    msp = mp_section( mp /* MinParser instance to be used */
                     , "[Section]" /* Start tag of the section */
                     , "[EndSection]" /* End tag of the section */
                     , section_nuber ); /* section to be read: >0 */

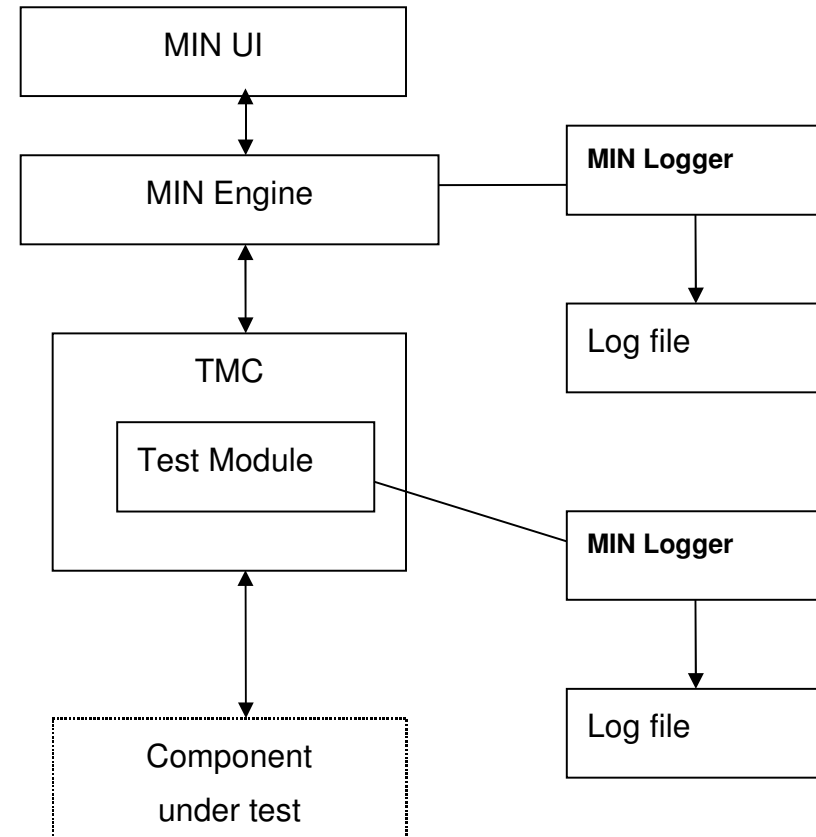
    /* Now we are going to parse file section by section */
    while( msp != INITPTR ) {
        /* Lets create MinItemParser */
        mip = msp_get_item_line( msp /* StixSelectionParser to be used */
                                , "variable" /* the line to parse */
                                , ENoTag ); /* is tag returned also */
        while( mip_get_next_string( mip, &string ) == 0 ) {

            /* Print it onto the screen: */
            printf( "Section %d, variable has value: [%s]\n"
                   , section_number
                   , string);
            /* string is malloc'ed, so clean it up */
            free(string);
        }

        ...
    }
}
```

MIN Logger

- **Easy-to-use logging facility for the test modules**
- **Versatile configurable features**
 - **Different logging formats supported: txt, html**
 - **Different output targets: file and syslog**
 - **Information importance can be specified (e.g. important information is written in red when generating html)**
 - **Configurable via min.conf file**
- **Used also for test framework logging**



Example: MIN Logging

```
#include <min_logger.h>

/*
** Simple logger example.
** Logs one line to testlog.txt in /tmp
** */
int logger_example0() {
    int retval = 0;
    MinLogger *logger;

    /*
    ** Create a logger instance
    ** */
    logger = mnl_create( "/tmp"      /* output dir */
                        , "testlog" /* output file */
                        , ESTxt      /* unsigned int loggertype */
                        , ESFile     /* unsigned int output */
                        , 1          /* TSBool overwrite */
                        , 1          /* TSBool withtimestamp */
                        , 1          /* TSBool withlinebreak */
                        , 0          /* TSBool witheventranking */
                        , 0          /* TSBool pididtologfile */
                        , 0          /* TSBool createlogdir */
                        , 0          /* unsigned int staticbuffersize */
                        , 0          /* TSBool unicode */ );

    /*
    ** Log something
    ** */
    retval = mnl_log( logger
                    , ESBold
                    , "Example log number %d"
                    , 1 );

    /*
    ** Free the logger
    ** */
    mnl_destroy( &logger );

    return retval;
}
```

MIN Logger output example

- `cat /tmp/testlog.txt`
 - 15.Apr.2008 05:16:23.220 Example log number 1

MIN Event System

- Provides easy-to-use interface for synchronization between test cases
- State events
 - Indicate that some state is active/inactive and they can be set and unset
 - E.g. phone call active/inactive
- Indication events
 - Indicate that some event happened and they can only be set
 - E.g. received SMS
- Identified with character string
- Event commands:
 - Set event
 - Unset event
 - Request event
 - Wait event
 - Release event

Example: State Events

[Test]

title wait for call

request call state

wait call

release call

[Endtest]

Example: Indication Events

```
#ifdef TEST_VAR_DECLARATIONS
    minEventIF *event;
#endif /* TEST_VAR_DECLARATIONS */
/**
 * MIN_SETUP defines activities needed before every test case.
 */
MIN_SETUP
{
    event = min_event_create("wait4sms", EIndication);
}
/**
 * MIN_TESTDEFINE defines a test case
 */
MIN_TESTDEFINE(test_wait4sms)
{
    tm_printf (1, "", "Registering for the wait4sms event ");

    event->SetType (event, ERegEvent);
    Event (event);
    MIN_ASSERT_EQUALS (event->event_status_, EventStatOK);

    tm_printf (1, "", "Waiting for the event ");

    event->SetType (event, EWaitEvent);
    Event (event);
    MIN_ASSERT_EQUALS (event->event_status_, EventStatOK);

    tm_printf (1, "", "Releasing the event");
    event->SetType (event, ERelEvent);
    Event (event);

    MIN_ASSERT_EQUALS (event->event_status_, EventStatOK);
}
/**
 * MIN_TEARDOWN defines activities needed after every test case
 */
MIN_TEARDOWN
{
    min_event_destroy (event);
}
```

Task: Using indication events

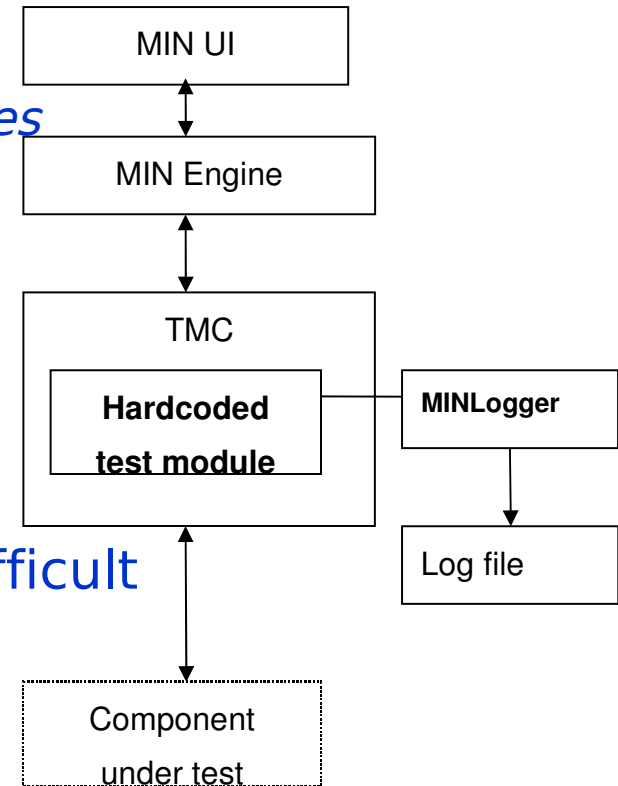
- Create one hardcoded test module e.g. Event
- Create two test cases for it using events (See Section MIN Event System from MIN_Test_Framework_Users_Guide.doc for more information)
 - Test case WaitEvent that requests, waits and then releases indication event “CoffeeReady”
 - Test case SetEvent that sets indication event “CoffeeReady”
- Add include “min_test_event_if.h”
- Run them with Console UI
 - First run WaitEvent
 - See that WaitEvent is on Ongoing cases list
 - Run SetEvent
 - See that both cases are executed

MIN Test Module Wizard

- For generating Test Module templates
- Procedure:
 - Run script ***createtestmodule***
 - Enter test module type, name and creation path
 - Creates new test module project, ready to be compiled
 - Go to created test module directory
 - Run script **build.sh**
- Possible test module template types of MIN currently are:
 - **hardcoded** (for hardcoded test module)
 - **Normal** (for normal test module)
 - **minunit** (for MINUnit test module)
 - **testclass** (for scripter test module)

Hardcoded test module

- All test cases implemented inside a test module
 - See one example in directory *min/src/test_libraries*
 - One of those can be used to try out MIN
- Used if
 - No parameterization in test cases
 - No initialization in test cases
- Recommended way to use
 - Basic way for test case implementation
 - Parameterization is not possible
 - Reusing the code of other test case more difficult

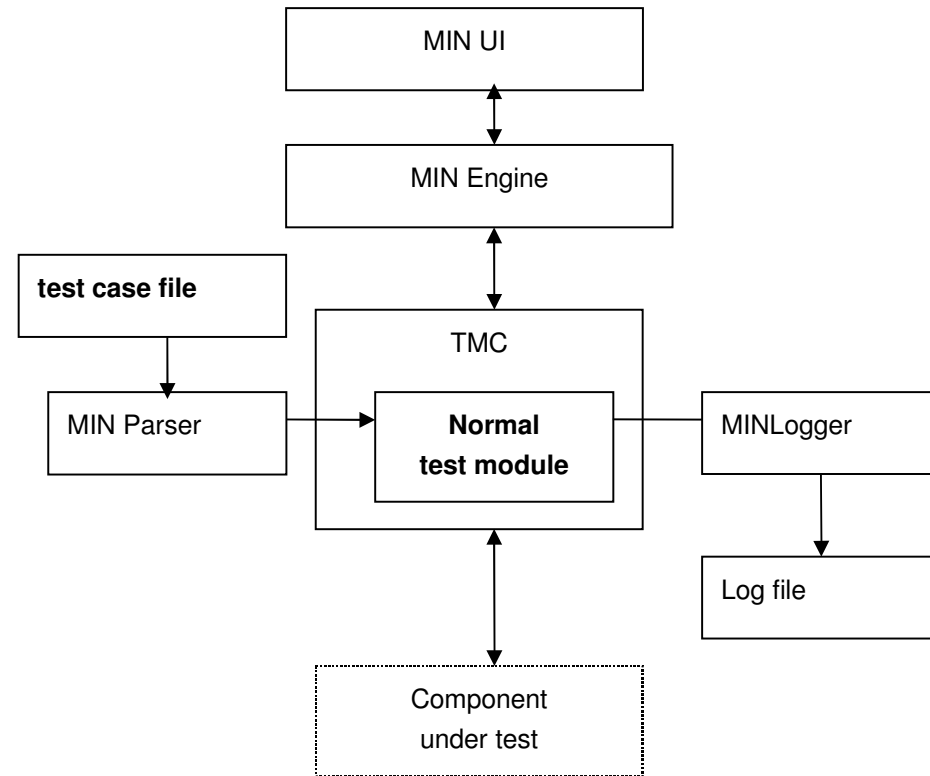


Task: Implementing test cases for Hardcoded test module

- Create your own hardcoded test module (e.g. TMHardcoded) using **createtestmodule**

Normal Test Module

- Normal test module template used for general purposes
- Used if
 - Parameterization is needed
 - Test scripter can't be used
 - Unique type of test cases are made
 - Does not limit implementation
- Recommended way to use
 - Test cases use parameters from test case file



Task: Implementing test cases for normal test module and using MIN Parser

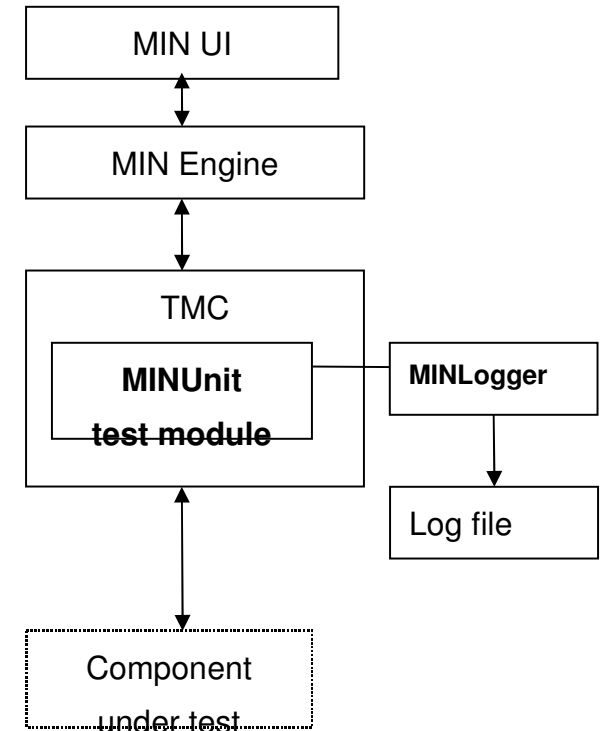
- Create your own normal test module (e.g. TMNormal) using createtestmodule
- Make your own test case file (e.g. TMNormal.cfg)
- Use MIN Parser to read data from your .cfg file
 - How to use parser? More information from MIN Test Framework User's Guide
- In GetTestCases -function read names(titles) of your test cases
- In RunTestCase -method read all data of your test cases and use Print -function to print data to MIN Console UI

Task: Implementing test cases for normal test module and using MIN Logger

- Use MIN Logger to create logs from your test module

MINUnit Test Module

- Compliant with popular xUnit test pattern.
- The fastest way to implement test cases
- MINUnit has clean and neat structure:
 - MINUnit groups code common for all test cases which makes test cases shorter
 - Use of ASSERT macros eliminates the need to use conditional expressions to make decision about result of a test. Macros take as a parameters expected and actual values



```

/**
 * GLOBAL VARIABLES SECTION
 */
#ifdef TEST_VAR_DECLARATIONS
    /**
     * Example of variable common for some test cases
     */
    char *ex;
#endif /* TEST_VAR_DECLARATIONS */
/**
 * END OF GLOBAL VARIABLES SECTION
 */

/**
 * TEST CASES SECTION
 */
#ifdef TEST_CASES
    /**
     * MIN_SETUP defines activities needed before every test case
     */
    MIN_SETUP
    {
        min_info("UNIT ex SETUP\n");
    }
    /**
     * MIN_TEARDOWN defines activities needed after every test case
     */
    MIN_TEARDOWN
    {
        min_info("UNIT ex TEARDOWN\n");
    }
    /**
     * MIN_TESTDEFINE defines a test case
     */
    MIN_TESTDEFINE(ex_1)
    {
        MIN_ASSERT_NOT_EQUALS(0,1);
    }

```

Test Scripter Test Module

- provides one way to implement scripted test cases
- it executes methods listed in the test case file from test classes
- Idea:
 - Implement small building blocks, i.e. functions, with unlimited number of parameters to test class
 - E.g. wrapper to API function calls
 - Create test cases to script file (test case file)
 - E.g. call functions in different sequences with different parameters
- Ready made script runner
 - Script file parsing
 - Execution of functions from test class

TestScripter test module (cont')

- Used if
 - Want to make test cases using small building blocks
 - Events used
 - Events controlling is handy from test script file
 - A lot of same type of test cases
 - Want to change testing steps execution order between test cases
 - Want to make more test cases without coding
 - Test class implemented once and more test cases made by adding new lines to script (text file)
- Recommended way to use
 - Test cases defined in test script file (text file)
 - Easy to understand what the test case is doing by reading test script file

Test Script File

- TestScripter is controlled with test script file
- Contains test case definition's
 - title: test case title which is showed to tester in UI
 - createx: create new object from specified test class
 - delete: delete specified object
 - Other lines begin with the name of the created object followed with command executed from that object with its arguments

```
[Test]
title Capture
createx CameraTest cam1
cam1 Init primary highres
cam1 capture "/tmp/pic.jpg"
cam1 Close
delete cam1
[Endtest]
```

Task: Test class & Test Scripter

1. Create test class(e.g. TMTestClass) and build it
2. See Example.cfg from Group folder of your test class
3. Edit min.conf file
 - Add Scripter to ModuleName
 - Add Example.cfg to TestCaseFile
4. Run Example test case from MIN Console UI and check parameters of test case via screen of Console UI
5. Add new method Example2 and one test case for it to Example.cfg file
 - Use parser to get integer data from test case
6. Add new method Example3 and one test case for it to Example.cfg file
 - Use parser to get char data from test case

Test Scripter test Combining feature

- Allows to create new test cases from existing ones
 - Extend and combine existing cases
 - Efficient way of implementing negative test cases
- Combining functionality can be used to
 - TMC's with multiple clients
 - Test operations when resources are in use
 - Simulate client crashes
 - Simulate client out-of-resource situations, i.e. buffer overflow
- Simple scripting language for test generating
 - Start test case
 - Abort, pause and resume test
 - Wait until test is completed
 - Use event system easily

Example: Test Combining features in System Testing

- Assume that there is networking test module
 - CSD and GPRS data transfer test cases
- Assume that there is Camera test module
 - Test case that takes a picture
- Combine camera test case and networking test case to
 - Take picture when CSD call is on
 - Take picture when GPRS data transfer is ongoing

```
[Test]
title Take a picture while CSD is active
request CSDCallActive
run netmodule nettests.txt 3
wait CSDCallActive
run cameramodule cameratests.txt 1
release CSDCallActive
[Endtest]
```


Example: Test Combining features in Module Testing

- Start CSD data transfer and when phone call is active kill test thread
 - Simulates a client crash when making CSD call
 - Similar cases can be implemented with any server
- Test process can also be paused
 - Causes buffer overflows
 - Simulate system overload situation

```
[Test]
title Phone application crash
request CSDCallActive
run netmodule nettests.txt 3
testid=nettest
wait CSDCallActive
pause nettest
cancel nettest
release CSDCallActive
[Endtest]
```

Task:Test Scripter combiner functionality

- 1.Create new test case file e.g. Event.cfg
- 2.Add scripter and your test case file to min.conf file
- 3.Implement Scripter test case
 - Request “CoffeeReady” event
 - Run SetEvent test case from your Event test module
 - Wait “CoffeeReady” event
 - Release “CoffeeReady” event
- 4.Run your Scripter test case with MIN Console UI

And after that some extra tasks ☺

- 1.Try to run other test cases by using Test Scripter combiner functionality
 - 1.TMHardcoded, TMNormal, TMTestClass etc...

Stress/interference testing

- MIN supports stress/interference testing, using sp-stress package tools. It allows for testing the code under conditions of:
 - Heavy cpu load
 - High memory usage
 - Heavy IO operations
- Feature requires sp-stress package to be installed
- Can be used in scripted and “coded” test cases.

Stress/Interference Examples

- Coded cases:

```
int test_1 (TestCaseResult * tcr)
{
    int i = 0;
    testInterference* disturbance =
        ti_start_interference(ECpuLoad,76);
    long double x1 = 0;
    long double x2 = 11.34223;
    for (i = 0; i < 100000000; i++) {
        x1 = sinl(x2);
    }
    ti_stop_interference(disturbance);
    tm_printf(1, "xxx", "res = %f", x2);
    tm_printf(1, "xxx", "iterations = %d", i);
    RESULT (tcr, TP_PASSED, "PASSED");
}
```

- Scripted cases:

```
[Test]
title interference - scr
createx heavycTestModule math
testinterference disturbance start cpuload 80 0 1111
math Heavy_math foo
testinterference disturbance stop
delete math
[Endtest]
```

Task: Using MIN in real HW

- Create MIN Debian packages in Scratchbox for ARM target
- Create Debian package(s) from your test module
- Install Debian packages in target device
- Run tests

Python support:

It is possible to run test cases written in Python. Python script interpreter module is similar in use to „scripter” module – user defined test cases are implemented in „test case file” – in this case : Python script. Examples of use would be:

- Testing C code in python extension modules, with test logic defined in Python
- Testing code written from the beginning in Python.

Python test case definition:

```
import min_ext
import sut_pextp

def case_sut():
    """Create file"""
    min_ext.Print_to_cui("file creation")
    retval = sut_ext.File_creation("/tmp/python_created_file")
    return retval
```

- To use min specific functionality , min_ext extension library needs to be imported (reference for available functions in user manual)
- If user wants to use his own C code, he needs to create and import his own C extension
- Min treats Python function with name strating with case_ as a test case
- Title can (but does not have to) be defined in docstring
- Cases defined this way are executed the same way as other test cases

Lua support:

It is possible to run test cases written in Lua.

Lua Scripter module is similar in use to MIN Scripter module.

User defined test cases are implemented in test case file that, in this case, is written in Lua.

Examples of use would be:

- Testing C code in as it has been done by using MIN Scripter.

Lua test case definition:

```
function case_function() -- Test Case title
    ret = min.run(„TestModule“,1);
    return TP_PASSED;
end

Function case_function2() -- Loading test module
    hnd = min.load(„LuaTestClass“);
    ret = hnd.Example();
    return ret;
end
```

- No dependencies to external libraries are present, Lua comes with MIN out of the box
- If user wants to use his own C code, he needs to create Lua Test Module
- MIN sees test cases from test case file only when they are prepended with „case_“
- Title can (but does not have to) be defined in the same line on which function is declared, in a comment (comment in Lua starts with --).
- Cases defined this way are executed the same way as other test cases

MIN Contact Info

- Documentation Included in release (user guide)
- Contacts:
 - MIN Contact: antti.heimola@nokia.com
 - Report bugs to: DG.MIN-Support@nokia.com

GNU Free Documentation License

- See <http://www.gnu.org/licenses/fdl-1.2.txt>