



Product:	CDPUnit
Product version:	v0.2
Document ID:	UM-CDPUnit
Doc revision:	PA2
Written/Apr.:	KD /
Date:	11. Mar. 2010

Industrial Control Design AS



CDPUnit v0.2

User Manual

The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.

Industrial Control Design AS, www.icd.no, support@icd.no, forum.icd.no

Contents

1. INTRODUCTION.....	3	4.4. Signals.....	8
1.1. About.....	3	4.5. States.....	8
1.2. Overview.....	3	4.6. Alarms.....	8
1.2.1. The CDPUnit add-on.....	3	4.7. Parameters.....	9
1.2.2. The ComponentTestRunner Component.....	3	4.8. Messages.....	9
1.2.3. The ComponentTestRunner State Machine.....	3		
2. COMPONENTTESTRUNNER FEATURES.....	4	5. DEMO APPLICATIONS.....	10
2.1. Registering Unit Tests.....	4	5.1. Overview.....	10
2.1.1. Description.....	4	5.1.1. Application overview.....	10
2.2. Registering Component Tests.....	4	5.1.2. Demo directory structure.....	10
2.2.1. Description.....	4	5.2. Using	10
2.3. Commands.....	4	5.2.1. Getting started.....	10
2.3.1. RegisterTests.....	4	5.2.2. Register tests.....	10
2.3.2. RunTests.....	4	5.2.3. Run tests.....	10
2.4. Logging.....	5	5.2.4. Inspecting the log.....	11
2.5. Failures and Errors.....	5		
3. APPLICATION CONFIGURATION.....	6	6. TEST AUTOMATION.....	12
3.1. How to add the ComponentTestRunnercomponent to a CDP Application.....	6	6.1. Command-line running.....	12
		6.2. Web interface control	12
		6.2.1. Sending messages through web interface.....	12
		6.3. Scripting language bindings.....	12
		6.3.1. ACTIR4CDP.....	12
4. CONFIGURATION.....	7	7. APPENDIX.....	13
4.1. Activate.....	7	7.1. Example Component File.....	13
4.1.1. Description.....	7	7.2. References.....	13
4.1.2. Example.....	7		
4.2. fs.....	7		
4.2.1. Description.....	7		
4.2.2. Example.....	7		
4.3. TestSetup.....	7		
4.3.1. Description.....	7		
4.3.2. Example.....	7		

1. Introduction

1.1. About

This document describes the ComponentTestRunner component of the CDPUnit library. Details will be provided to make the reader understand how the component works, and how to configure and use it with the CDP system.

For information regarding the CDPUnit API, see the Programmers Manual.

1.2. Overview

1.2.1. The CDPUnit add-on

The CDPUnit add-on consists of CDP components and library extensions that allows you to integrate with C++ unit tests written using the open-source CppUnit testing framework with your CDP application.

1.2.2. The ComponentTestRunner Component

The primary function of the ComponentTestRunner (or CTR) is to collect unit tests and component tests, control the execution of those tests and reporting the results back through ordinary CDP signals.

1.2.3. The ComponentTestRunner State Machine

The state machine of the ComponentTestRunner component includes 4 states. The logger is started in 'Off' state. In this state no tests are registered and no tests can be run.

Next tests are registered either explicitly by message 'RegisterTests' text command or implicitly by message 'RunTests' text command. Once the tests are registered it goes to the 'Ready' state.

Running the tests is started by the 'RegisterTests' text command and puts the ComponentTestRunner into the 'RunningTests' state. Once the test run is done, it goes to the 'Complete' state.

2. ComponentTestRunner Features

This chapter describes the various ComponentTestRunner features and testing types.

2.1. Registering Unit Tests

2.1.1. Description

Each unit test that has registered itself with the CppUnit::TestFactoryRegistry and has been linked in with your CDP application will be picked up by the ComponentTestRunner during run-time registration of tests, unless it has been explicitly disabled.

Registering with the TestFactoryRegistry is done at compile-time using CppUnit macros.

2.2. Registering Component Tests

2.2.1. Description

The Component Tests are all registered at run-time, since the test classes often also are CDP components. The ComponentTestRunner will iterate over its sub-components to see if any of them implement the DynamicTestInterface and thus can provide testcases.

Using a special top-level CDP Application component you can get all components in your CDP application checked for testcases and have them all registered. (TODO!)

2.3. Commands

2.3.1. RegisterTests

The 'RegisterTests' text command is triggered by a CDP message, and can be sent through the web interface, CDP browser or from other CDP components/applications.

The 'RegisterTests' command will only register the tests, preparing a test suite collection. Actual execution of the tests is not done until the 'RunTests' command is sent.

2.3.2. RunTests

The 'RunTests' text command is triggered by a CDP message, and can be sent through the web interface, CDP browser or from other CDP components/applications.

It will implicitly call the 'RegisterTests' command if no registered test suite collection is found. Then it starts execution of the test cases in a separate thread, updating signal values to reflect progress.

If any failures or errors occur during the test run, the appropriate alarms are set.

2.4. Logging

All detailed logging of the testing progress is outputted on the CDP console using CDPMessage function.

Especially for test case failure details, you will need to inspect the console output. Look for the “[CDPUnit]” tag in the output logs.

See also 'LogFile' XML element.

2.5. Failures and Errors

Failures are testcases that were run, but an unexpected result occurred in an assertion. Errors are unexpected exceptions or errors that occurred during execution of a testcase.

When either of these occur, a test case will be counted as a failure, and no more code in that testcase is run. In the ComponentTestRunner component the failure is reported in signals counting the number of failures and errors, and through alarms.

To investigate the failure, please look at the CDP console output. It should provide more details on the individual test case failures. Each test start and completion is logged to help identify other relevant CDP log messages that might be pertinent to the failure at hand.

The final test report will also iterate each failure and any details captured, such as the test method, filename and linenumber, the assertion that failed etc.

Example final test report for MyComponentTestApp with a failure:

```

11:35:05 [CDPUnit] ===== Test Report: =====
11:35:05 [CDPUnit]
11:35:05 [CDPUnit] !!!FAILURES!!!
11:35:05 [CDPUnit] Test Results:
11:35:05 [CDPUnit] Run: 3   Failures: 1   Errors: 0
11:35:05 [CDPUnit]
11:35:05 [CDPUnit]
11:35:05 [CDPUnit] 1) test: ComponentTestRunner.TutorFilter_Test::testFilterCoefficient (F)
line: 89 TestTutor/IIRFilterTutor_TestTutor.cpp
11:35:05 [CDPUnit] double equality assertion failed
11:35:05 [CDPUnit] - Expected: 40
11:35:05 [CDPUnit] - Actual   : 36
11:35:05 [CDPUnit] - Delta    : 0.001
11:35:05 [CDPUnit]

```

3. Application Configuration

This chapter describes how to instantiate the ComponentTestRunner component within a CDP application.

3.1. How to add the ComponentTestRunner component to a CDP Application

Add the following to your project's Application.xml:

Inside the <Components> element, add an instance of a ComponentTestRunner component, for instance:

```
<Component Name="ComponentTestRunner" Type="ComponentTestRunner"
  src="Components/ComponentTestRunner.xml"></Component>
```

Or, inside the <Subcomponents> element, add:

```
<Subcomponent Name="ComponentTestRunner" Type="ComponentTestRunner"
  src="Components/ComponentTestRunner.xml"> </Subcomponent>
```

This will tell CDP to initialize a component named "ComponentTestRunner" from a component file located at "Components/ComponentTestRunner.xml". Make sure that your Models folder contains a ComponentTestRunner.xml model file, or the component will not be initialized correctly.

Configuration is done by modifying the component xml file. It should not be necessary to modify the model XML file. An example of ComponentTestRunner.xml component XML file is found in 7.1.

4. Configuration

This chapter describes the various XML configuration parameters of the ComponentTestRunner component.

4.1. Activate

4.1.1. Description

Specifies the time to delay start-up of the component, after configure has been run. If the *AutoStartTest* parameter is set, the component will then automatically start running the tests.

4.1.2. Example

```
<Activate>1</Activate>
```

4.2. fs

4.2.1. Description

This tag sets the rate in Hertz at which the process function of the component is executed, which in turn affects the rate of execution of test-cases in an approximate fashion. Only one test-case is started per fs, but a single test-case might take multiple fs to complete.

4.2.2. Example

```
<fs>10</fs>
```

4.3. TestSetup

4.3.1. Description

Container for various test-runner specific options, such as a designated output log file for the test logs and specific test selection.

4.3.2. Example

```
<TestSetup>
  <LogFile>testlog.txt</LogFile>
  <Select>TestExampleUnitTest</Select>
</TestSetup>
```

4.4. Signals

The following signals are in the ComponentTestRunner component:

Signal Name	Description
RegisteredTests	Number of registered tests.
RunnedTests	How many tests have been run.
TestsPassed	Number of passing tests.
TestFailures	Number of failed tests.
TestErrors	Number of errors occuring during testing.
WasTestRun	Has the test suite been run? (Set when all tests have been run.)
WasTestSuccess	Did the test run complete successfully or not? (I.e. no failures or errors.)

4.5. States

The following states are in the ComponentTestRunner component:

State	Description
Off	The test runner is off.
RegisteringTests	Registering tests.
Ready	Ready to run tests.
RunningTests	In the process of running tests.
Complete	Tests have been run.

4.6. Alarms

The following alarms are in the ComponentTestRunner component:

Alarm Name	Description
TestErrorOccured	An error occurred during the running of the tests.
TestFailureOccured	A test failed during the running of the tests.

4.7. Parameters

The following parameters are in the ComponentTestRunner component:

Parameter Name	Description
UseTestFactoryRegistry	Register unit tests found in the TestFactoryRegistry of CppUnit.
RegisterSubcomponentTests	Register component tests found in subcomponents of the CTR.
RegisterAllComponentTests	Register component tests found in any subcomponent below the current application. Note: Only checks subcomponents of Application, not other top-level components.
AutoStartTest	Auto-start registering and running on component activation. Note: Be sure to set Activate to a sensible non-zero value.
AutoRegisterTests	Automatically start registering testcases on application activation. (Does not need to be set if auto-starting.)
AutoShutdown	Automatically shut down CDP application after tests have run.

4.8. Messages

The following messages can be sent the ComponentTestRunner component:

Message Name	Description
RegisterTests	Find and register test cases as specified in XML configuration and through the parmas.
RunTests	Start execution of the tests. (Optional text argument to specify test selection.)
Off	Turn off the test runner.

5. Demo Applications

This chapter describes how to get started with the provided demo applications.

5.1. Overview

5.1.1. Application overview

The demo consists of two separate applications:

- MyUnitTestApp – shows how to run regular CppUnit tests from within a CDP application.
- MyComponentTestApp – shows how to extend a CDP component with test functionality.

5.1.2. Demo directory structure

There are three directories within the MyUnitTestApp demo directory:

- Application – the CDP application configuration
- CDP_Application – the CDP executable that links in the tests and the test frameworks.
- MyUnitTestLib – an example library consisting only of regular CppUnit tests.

The MyComponentTestApp demo directory has the same Application and CDP_Application in addition to:

- TutorLib – an example CDP component (taken from CDPTutor)
- TestTutorLib – a CDP test component that extends the above component.

5.2. Using

5.2.1. Getting started

Locate the CDPUnit demo directory. Go to the Application directory of a specific demo and start the application. Now, open a CDPBrowser (or a web browser).

5.2.2. Register tests

As systems grow larger, the accumulated registering of tests can become a larger task. Especially if you have configurable selection of the tests, which is useful for debugging and development. For those reasons, registering of the tests is done separately, so that the number of registered tests can be checked before the actual running is executed.

In the CDPBrowser or web UI go into the Messages section of the ComponentTestRunner and click to send the 'RegisterTests' message. The 'RegisteredTests' signal should go from 0 to the correct number of tests. For our example demos, however, the amount of tests is quite small, so you may skip this step.

5.2.3. Run tests

In the CDPBrowser or web UI go into the Messages section of the ComponentTestRunner and click to send the 'RunTests' message. Various signals are affected by this, including 'RunnedTests' signal should increase up from 0 towards the number of registered tests.

Once the test run is complete, boolean signals 'WasTestRun' will be set to true, and unless there were failures and/or errors, the 'WasTestSuccess' signal will also be set.

5.2.4. Inspecting the log

If 'WasTestSuccess' is false after the test has run, you're likely left with non-zero values in signals 'TestFailures' and 'TestErrors'. However, the numbers do not tell you which tests failed and why. For this you will need to inspect the CDP console log. It will be available in the console where you ran the CDP application and through the web interface. If you've used the LocalLogFile element in your Application configuration, it will also be written to file along with the rest of the CDP messages.

Example output from MyUnitTestApp:

```
11:09:43.479 CDPUnitLib build date: Wed Mar 25 12:31:52 2009 Version 0.1
11:09:43.516 ComponentTestRunner::MessageRunTests ...
11:09:43.516 [CDPUnit] Starting test run ...
11:09:43.516 ComponentTestRunner::RegisterTests: Fetching unit tests from TestFactoryRegistry...
11:09:43.521 [CDPUnit] Test 'TestExampleUnitTest::testAddition' starting...
11:09:43.521 [CDPUnit] Test 'TestExampleUnitTest::testAddition' done.
11:09:43.532 [CDPUnit] Test 'TestExampleUnitTest::testSubtraction' starting...
11:09:43.532 [CDPUnit] Test 'TestExampleUnitTest::testSubtraction' done.
11:09:43.542 [CDPUnit] Test 'TestExampleUnitTest::testMultiplication' starting...
11:09:43.542 [CDPUnit] Test 'TestExampleUnitTest::testMultiplication' done.
11:09:43.552 [CDPUnit] Test 'TestExampleUnitTest::testDivision' starting...
11:09:43.553 [CDPUnit] Test 'TestExampleUnitTest::testDivision' done.
11:09:43.558 [CDPUnit] ===== Test Report: =====
11:09:43.558 [CDPUnit]
11:09:43.558 [CDPUnit] OK (4 tests)
11:09:43.558 [CDPUnit]
11:09:43.558 [CDPUnit]
11:09:43.558 [CDPUnit]
11:09:43.558 [CDPUnit] Test run complete.
11:09:43.558 OSAPIThread 'CDPUnit_TestRunner' exited normally.
```

Notice the “[CDPUnit]” tag on the main test report output. Each test start and stop is logged to help investigation in case of failures.

6. Test Automation

There are several ways to automate more of the testing process. Here a few starter points are given.

6.1. Command-line running...

By using the `AutoStartTest` parma, you can easily start and run a set of test cases configured in a CDP application.

6.2. Web interface control

6.2.1. Sending messages through web interface

The web interface can also be used to start tests by sending messages to the component. Send off an HTTP GET request to an URL like these examples:

```
http://localhost/ComponentTestRunner?Message=RunTests  
http://localhost/ComponentTestRunner?Message=RunTests&Select=TestExampleUnitTest  
http://localhost/ComponentTestRunner?Message=RunTests&Select=TestExampleUnitTest::testMultiplication
```

This can be useful for programmatically starting tests in remote CDP applications.

6.3. Scripting language bindings

Most scripting languages allow you to run external programs and capture their output while feeding them input. For CDP, you can also use the Web Interface or even the CDP protocol directly for testing.

6.3.1. ACTIR4CDP

Automated Component Testing in Ruby for CDP is a simple, light-weight testing framework that quickly allows you to automate manual testing of CDP components. Anything that can be tested through the Web Browser and/or CDPBrowser can be automated.

<http://actir4cdp.rubyforge.org/>

7. Appendix

7.1. Example Component File

```

<?xml version="1.0" encoding="iso-8859-1"?>
<Component Name="ComponentTestRunner" Model="ComponentTestRunner">
  <Activate>1</Activate>
  <fs>100</fs>
  <InitialState></InitialState>

  <Description><![CDATA[ Run multiple component tests from CDP. ]]></Description>

  <TestSetup>
    <LogFile>testlog.txt</LogFile>
  </TestSetup>

  <Signals>
    <Signal Name="RegisteredTests"      Input="0" Type="int" Unit="" Value="" Routing="No routing"
    Description="Number of tests registered with the test runner."></Signal>
    <Signal Name="RunnedTests"           Input="0" Type="int" Unit="" Value="" Routing="No routing"
    Description="Number of tests that have been run."></Signal>
    <Signal Name="TestErrors"            Input="0" Type="int" Unit="" Value="" Routing="No routing"
    Description="Number of tests that failed due to errors."></Signal>
    <Signal Name="TestFailures"          Input="0" Type="int" Unit="" Value="" Routing="No routing"
    Description="Number of failed tests."></Signal>
    <Signal Name="TestsPassed"           Input="0" Type="int" Unit="" Value="" Routing="No routing"
    Description="Number of passing tests."></Signal>
    <Signal Name="WasTestRun"            Input="0" Type="bool" Unit="" Description="Have the tests been run?"></Signal>
    <Signal Name="WasTestSuccess"        Input="0" Type="bool" Unit="" Description="Was the test run a success?"></Signal>
  </Signals>

  <Alarms>
    <Alarm Name="TestFailureOccured" Group="" Level="Error" Trig="0" Enabled="1" EnabledState="" Signal="" Inverted="0"
    SignalOutSet="" Text="A test failed." Description="A test failed." CppName="m_testFailureOccured"></Alarm>
    <Alarm Name="TestErrorOccured" Group="" Level="Error" Trig="0" Enabled="1" EnabledState="" Signal="" Inverted="0"
    SignalOutSet="" Text="An error occurred during a test." Description="An error occurred during a test."
    CppName="m_testErrorOccured"></Alarm>
  </Alarms>

  <Parameters>
    <Parma Name="UseTestFactoryRegistry" Unit="" Value="1" Min="0" Max="1" DefaultValue="1" PreviousValue="0"
    TimeLastChanged="" Description="Use the CppUnit TestFactoryRegistry to find testcases."></Parma>
    <Parma Name="RegisterSubcomponentTests" Unit="" Value="1" Min="0" Max="1" DefaultValue="1" PreviousValue="0"
    TimeLastChanged="" Description="Register testcases found in our subcomponents."></Parma>
    <Parma Name="RegisterAllComponentTests" Unit="" Value="1" Min="0" Max="1" DefaultValue="1" PreviousValue="0"
    TimeLastChanged="" Description="Register testcases found in any subcomponent of Application."></Parma>
    <Parma Name="AutoStartTest" Unit="" Value="0" Min="0" Max="1" DefaultValue="0" PreviousValue="0" TimeLastChanged="Wed Mar
    10 12:57:42 2010" Description="Automatically start running the testcases on application activation."></Parma>
    <Parma Name="AutoRegisterTests" Unit="" Value="0" Min="0" Max="1" DefaultValue="0" PreviousValue="0" TimeLastChanged=""
    Description="Automatically start registering testcases on application activation." CppName="m_autoRegisterTests"></Parma>
    <Parma Name="AutoShutdown" Unit="" Value="1" Min="0" Max="1" DefaultValue="0" PreviousValue="0" TimeLastChanged="Wed Mar
    10 12:57:26 2010" Description="Automatically shutdown CDP application on test completion."></Parma>
  </Parameters>

  <Subcomponents>
    <Subcomponent Name="TutorFilter_Test" src="Components/TutorFilter_Test.xml"></Subcomponent>
  </Subcomponents>
  <RemoteComponents>
  </RemoteComponents>
</Component>

```

7.2. References

CppUnit - C++ unit testing framework
<http://cppunit.sourceforge.net/>