



Product:	ACTIR
Product version:	v0.4
Document ID:	UM-ACTIR
Doc revision:	PA4
Written/Aprr.:	KD /
Date:	5. Mar. 2010

Industrial Control Design AS



ACTIR4CDP v0.4

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The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.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	6. EXAMPLES.....	9
1.1. The ACTIR4CDP testing framework.....	3	6.1. Overview.....	9
1.1.1. ACTIR – Component Testing.....	3	6.2. Generator Component Test.....	9
1.1.2. WATIR – Web Interface testing.....	3	6.2.1. Getting started.....	9
		6.2.2. Run tests.....	9
		6.2.3. Inspecting the log.....	9
2. ACTIR FEATURES.....	4		
2.1. Create Application configuration.....	4	7. APPENDIX.....	10
2.2. Run CDP executable.....	4	7.1. Example Component Test for Generator.....	10
2.3. Access CDP through Web Interface.....	4	7.2. References.....	11
2.4. Logging.....	4		
2.5. Failures and Errors.....	4		
3. REQUIREMENTS.....	5		
3.1. Basic ACTIR Installation on Windows.....	5		
3.1.1. Ruby.....	5		
3.1.2. GEM (rubygems) package installer.....	5		
3.1.3. Various Windows API gems for Ruby.....	5		
3.2. Installing WATIR on Windows.....	6		
3.2.1. WATIR (for Windows/IE) gem.....	6		
3.2.2. FireWatir (for Firefox) gem.....	6		
3.3. Installing ACTIR on Linux.....	6		
4. RUNNING.....	7		
4.1. Running ACTIR tests.....	7		
4.1.1. Command-line options.....	7		
4.2. Running WATIR tests.....	7		
4.2.1. Command-line options.....	7		
5. APPLICATION CONFIGURATION.....	8		
5.1. How to start a CDP Application from ACTIR	8		
5.1.1. Application directory configuration.....	8		
5.1.2. Starting CDP executable.....	8		
5.1.3. Shutdown.....	8		
5.1.4. Cleanup.....	8		

1. Introduction

1.1. The ACTIR4CDP testing framework

The ACTIR4CDP framework (Automated Component Testing in Ruby for CDP) consists of Ruby libraries that allows you to interact with a CDP application through the Web Interface. Any manual tests that can be done through the Web Interface of one or more CDP applications lend themselves to automation with this framework.

The primary target for the framework is high-level system and component testing, as well as Web Interface testing. For testing approaches at other levels, please refer to the *CDPUnit* add-on which provides Unit Testing and Component Testing more closely integrated with CDP components.

There are two primary approaches of using the testing framework; ACTIR and WATIR.

1.1.1. ACTIR – Component Testing

The primary function of the ACTIR framework to facilitate running of system and component tests on CDP applications, control the configuration, setup and execution of those tests and reporting the results back to a simple report.

The ACTIR component testing framework communicates directly with a running CDP application through HTTP and XML. No Web Browser dependency, which makes for faster, easier tests, useful for testing internal CDP component logic.

1.1.2. WATIR – Web Interface testing

The WATIR framework (Web Application Testing in Ruby) is a third-party library used for automatic web-testing that involves a Web Browser, such as Internet Explorer or Firefox. The benefits of this, is that client-side web-functionality can be tested, ensuring that HTML, JavaScript, XML and XSLT are served and operating correctly.

The ACTIR framework provides extensions and helper-functionality to ease the use of WATIR when testing CDP applications and their complete Web Interface.

2. ACTIR Features

This chapter describes the various ACTIR features and testing types.

2.1. Create Application configuration

The framework helps you create Application configuration on the fly, by using existing templates as basis, copying in test-specific configuration and allowing the test-case itself make modifications to the XML configuration before a test is run.

2.2. Run CDP executable

The framework hands execution and handling of CDP binary on the local computer, so that there is no need for manual starting/stopping of applications before the testing can be started.

2.3. Access CDP through Web Interface

After starting a CDP binary, access to the CDP application and components is provided through a Ruby-wrapper that accesses the Web Interface of the running CDP application through HTTP and XML.

2.4. Logging

The CDP Message logs are captured and available during the running of tests, which allows keeping a closer eye on the CDP application status without needing to fetch the logs through the Web Server.

2.5. Failures and Errors

Failures are test-cases that were run, but an unexpected result occurred in an assertion.
Errors are unexpected exceptions or errors that occurred during execution of a test-case.

When either of these occur, a test case will be counted as a failure, and no more code in that test-case is run. In the ACTIR the failures are reported through the Test::Unit framework in Ruby.

To investigate a failure, the test-code is the best place to start. Why did the assertion fail, what input differed etc. Re-running only the test that failed with verbose output can also be useful.

To investigate an error, you might also want to rerun the test with more debugging information, or even ask the test-runner to keep the resulting Application configuration directory for your manual inspection.

3. Requirements

ACTIR4CDP currently works on Windows and Linux.

Running ACTIR tests requires the following dependencies:

- Ruby language (1.8 or newer)
- GEM (rubygems) package installer.
- Windows API gems (win32-api, win32-open3 etc)

For WATIR tests you will also need:

- WATIR (for Windows/IE).
- FireWatir (for Firefox in Windows and Linux)
- Win32OLE bindings.
- JSSh (binary) Firefox extension

3.1. Basic ACTIR Installation on Windows

Most of the dependencies for ACTIR are in the Ruby installation package by default.

3.1.1. Ruby

Download One-Click Installer from:

<http://rubyinstaller.org/>

Note: ACTIR has been tested with Ruby **1.8.6-p27 (RC2)**.

Alternative download location: <http://www.ruby-lang.org/en/downloads/>

3.1.2. GEM (rubygems) package installer

Should be part of the One-Click Installer for Ruby above.

If not, see: <http://www.rubygems.org/>

3.1.3. Various Windows API gems for Ruby

Exact list of dependencies will vary with the tests and functionality used.

The primary dependency is on *win32-open3* for executing the CDP binaries.

To install it, you will either need a suitable compiler setup, or install a specific ready-made binary version

```
gem install --platform x86-mswin32 --version 0.3.1 win32-open3
```

3.2. Installing WATIR on Windows

WATIR has some additional dependencies, needed to control a Web Browser.

3.2.1. WATIR (for Windows/IE) gem

Details on installing WATIR: <http://wtr.rubyforge.org/install.html>

To install using gem, run this command:

```
gem install watir
```

3.2.2. FireWatir (for Firefox) gem

FireWatir allows control of Firefox browser in Windows and Linux.

To install using gem, run this command:

```
gem install firewatir
```

You will also need the binary JSSh Firefox extension, which is not available as a GEM.

Details on installing FireWatir: <http://wiki.openqa.org/display/WTR/FireWatir+Installation>

(Tested with JSSh version 0.9 in Firefox 3.0.1.)

Note: Using Web Browser automation can be frustrating if the browser pops up modal dialogs, hangs for user input or works inconsistently from one time to the next. Please check the URLs mentioned above for tips on configuring your browser to be “automation friendly”.

3.3. Installing ACTIR on Linux

Ruby and its libraries fit nicely with Linux, and most of the dependencies should be easily available in the package manager for your distribution. Assuming Debian or Ubuntu in the following examples

```
apt-get install ruby ruby-devel rubygems  
gem install firewatir
```

4. Running

4.1. Running ACTIR tests

Quick start all ACTIR tests:

```
ruby run_webinterface_tests.rb
```

Run a specific testcase, with verbose output from Test::Unit

```
ruby run_webinterface_tests.rb -- --testcase TestGenerator --verbose
```

Run a specific test, with additional debug output from ACTIR framework:

```
ruby run_webinterface_tests.rb --debug -- --name test03_sinus_output
```

Note: The double dash (--) separates options to ACTIR from options to Test::Unit.

4.1.1. Command-line options

Options to ACTIR test-runner script.

- --cdpbase – give alternative CDPBase directory for fetching templates.
- --cdpdevel – development directory, used for internal testing
- --debug – detailed debug output
- --verbose – verbose output (info)
- --keep – keep the generated application directory

4.2. Running WATIR tests

Quick start all WATIR tests on a specific host:

```
ruby run_watir_tests.rb --host http://localhost:89/
```

4.2.1. Command-line options

Options to WATIR test-runner script.

- --host – URL to Web Interface of CDP application to test on.
- --browser – What Web Browser to use. (IE, FF etc)
- --component – Which component test to load.
- --application – CDP application to start up before testing

5. Application Configuration

This chapter describes how to instantiate and configure a CDP application from within the ACTIR framework. This makes testing a lot quicker, as you do not need to manually start the CDP application(s) that you wish to test.

5.1. How to start a CDP Application from ACTIR

Running an ACTIR test-case presumes that you have the following:

- A ready-built CDP executable (CDP.exe)
- A (partial) CDP application setup (Application directory)

5.1.1. Application directory configuration

When the framework starts a CDP application, it creates a separate Application directory for this test run, to avoid polluting and mangling an existing Application setup. The normal procedure goes like this:

- Copy Application folder from CDP Developer project template as basis.
- Copy test-specific Application configuration on top.
- Copy in CDP executable.

The Application is then ready to be started. The procedure can be augmented to fetch from multiple places, fetching add-on specific XML templates etc. This means that the test-specific Application configuration only needs to concern itself with the changes for the test.

You may also modify specific XML configuration files from within the test-case. This is useful for doing minor modifications on the configuration (like changing Parameters), without needing a whole new test-specific configuration setup.

5.1.2. Starting CDP executable

The framework starts the CDP binary executable, hooking itself into the CDP Messages log to monitor progress. Once the message log have provided information about what port the WebServer is running, an object is created that provides access to the CDP application and components via the Web Interface. This is the main portable means of manipulating the CDP process while it is running.

5.1.3. Shutdown

Once a test is complete, the CDP application is shut-down, normally by sending a CM_REBOOT to the application. However, it may fallback to other means, such as sending a key-combo on the console or doing a brutal process kill.

5.1.4. Cleanup

The generated Application directory is cleaned up after the test, to avoid filling the hard drive with all the various tests. This is important since each test within a component test will run in its own Application directory, to avoid dependencies and configuration spill-over contaminating the test results. If you wish to inspect the Application directory after a test, you can pass the *--keep* option to the test-runner.

6. Examples

This chapter describes how to get started writing ACTIR tests based on example tests.

6.1. Overview

The example tests provide a starting point to writing your own. They presume that CDP Developer is installed with the various demo projects. See the *Generator_ComponentTest.rb* file in *WebUITests* for details.

6.2. Generator Component Test

6.2.1. Getting started

Locate the ACTIR directory (*WebInterfaceTest*) in a console or terminal.

6.2.2. Run tests

Run the test starter-script from the console with options specifying which test to run:

```
ruby run_webinterface_tests.rb -- --testcase TestGenerator --verbose
```

The double dash (--) separates options to ACTIR from options to `Test::Unit`.

On Linux, you'll need to run it as root, since it starts a CDP application, or through *sudo*:

```
sudo -E ruby run_webinterface_tests.rb -- --testcase TestGenerator
```

6.2.3. Inspecting the log

Example verbose output from *run_webinterface_tests.rb*:

```
Adding WebInterface tests for component WebUITests/Generator_ComponentTest.rb...
Starting TestCDP WebInterface tests...
=====
Loaded suite run_webinterface_tests
Started
test01_component_list(TestGenerator): .
test02_default_values(TestGenerator): .
test03_sinus_output(TestGenerator): .
test04_noise_output(TestGenerator): .

Finished in 16.082206 seconds.

4 tests, 18 assertions, 0 failures, 0 errors
=====
Finished running TestCDP WebInterface tests!
```

Notice that any file in the *WebUITests* folder ending in *_ComponentTest.rb* is loaded and its tests registered. Since we are explicitly stating which test-case we want to run, not all of the registered tests are run.

7. Appendix

7.1. Example Component Test for Generator

```

require 'TestCDP/WebInterface'
require 'test/unit'

class TestGenerator < Test::Unit::TestCase
  include TestCDP::Application::Configuration

  def setup
    cdpbasedir = TestCDP::Application::Configuration.cdpbase_dir
    demodir = cdpbasedir + '/Demo Projects/CDPUI/Generator/Application'
    @src_app = demodir + '/CDP'
    @compname = "Generator/Generator"
    generate_app_dir_from_templates( [demodir] )
    start_application
  end

  def teardown
    stop_application
    cleanup_generated_app_dir
  end

  def test01_component_list
    complist = @service.get_component_list
    assert_not_nil(complist.elements["Component//Subcomponent[@Name='Generator.Generator']"])
    assert_not_nil(complist.elements["Component//Subcomponent[@Name='Generator.Messenger']"])
    assert_not_nil(complist.elements["Component//Subcomponent[@Name='Generator.WebServer']"])
  end

  def test02_default_values
    signals = @service.get_signal_values
    assert_equal( 0.0, Float(signals['SinusGain']) )
    assert_equal( 0.0, Float(signals['NoiseGain']) )
    assert_equal( 0.0, Float(signals['SquareGain']) )
    assert_equal( 0.0, Float(signals['Output']) )
    assert_equal( 1.0, Float(signals['SinusAmplitude']) )
    assert_equal( 1.0, Float(signals['NoiseAmplitude']) )
    assert_equal( 1.0, Float(signals['SquareAmplitude']) )
  end

  def test03_sinus_output
    signals = @service.get_signal_values
    assert_equal( 0.0, Float(signals['SinusGain']) )
    assert_equal( 0.0, Float(signals['Output']) )
    @service.set_signal('SinusGain', 1.0)
    signals = @service.get_signal_values
    assert_equal( 1.0, Float(signals['SinusGain']) )
    assert_in_delta( Float(signals['Sinus']), Float(signals['Output']), 0.001 )
  end

  def test04_noise_output
    signals = @service.get_signal_values
    assert_equal( 0.0, Float(signals['NoiseGain']) )
    assert_equal( 0.0, Float(signals['Output']) )
    @service.set_signal('NoiseGain', 0.25)
    signals = @service.get_signal_values
    assert_equal( 0.25, Float(signals['NoiseGain']) )
    assert_in_delta( Float(signals['Noise']) * 0.25, Float(signals['Output']), 0.001 )
  end
end

```

7.2. References

Ruby programming language:

<http://www.ruby-lang.org/en/>

<http://www.ruby-lang.org/en/downloads/>

<http://rubyinstaller.org/>

GEM (rubygems) package installer:

<http://www.rubygems.org/>

Win32-open3 gem:

<http://rubygems.org/gems/win32-open3/>

<http://win32utils.rubyforge.org/>

WATIR framework– Web Application Testing in Ruby:

<http://wiki.openqa.org/display/WTR>

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

ACTIR4CDP:

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