CCDL Whitepaper

Summary: This document describes the test definition language CCDL and its application for a simple example..	
Reference/Related Documents:	
Notes:	
Keywords	CCDL, TRM, ITE

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

Revision History

	Name	Date	Issue
Author	Michael Wittner	10. May 2010	01
	Michael Wittner	23. January 2014	02
Execution flow visualization added			

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

Table of Contents

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

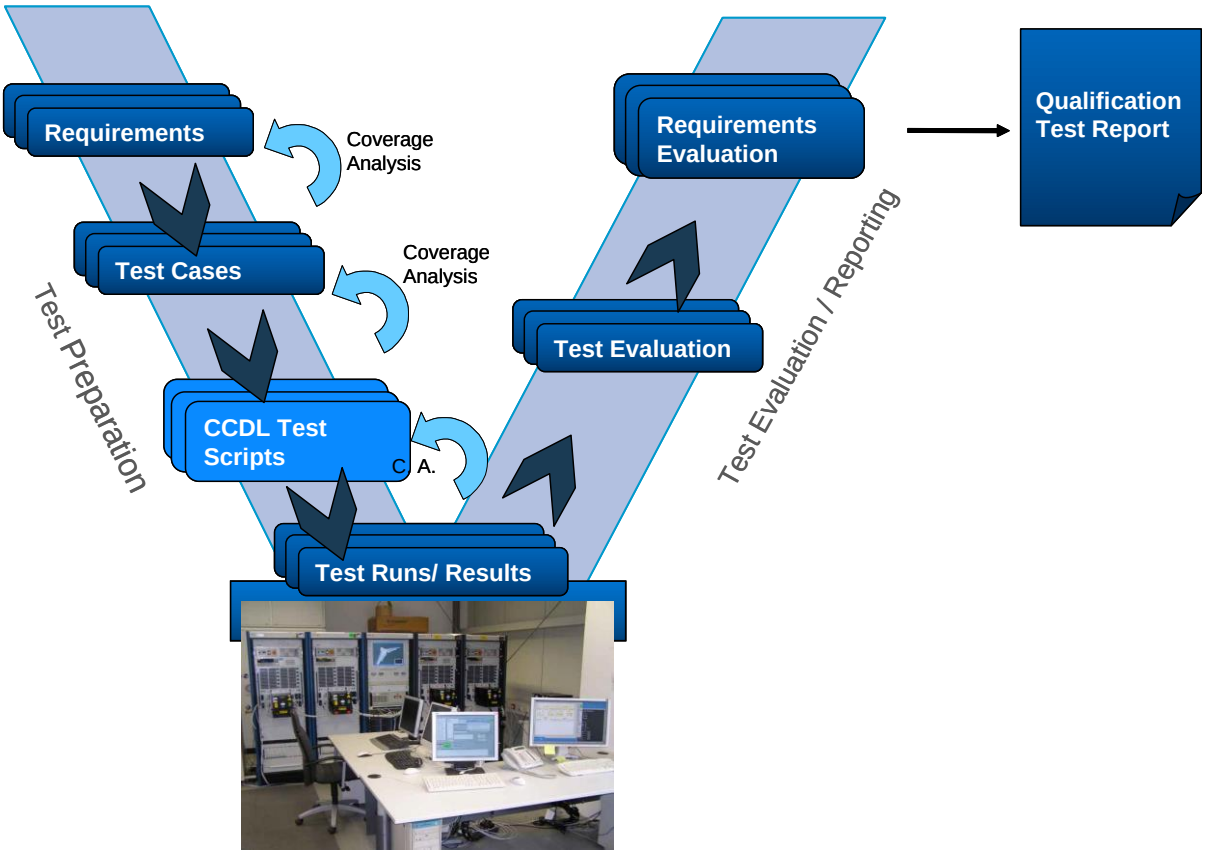
1 Introduction

Verification of safety critical systems requires full coverage of system under tests requirements. This results in many and complex test scenarios, to be executed and evaluated. Manual execution of such tests is error prone and not efficiently, though automated testing of the system under test (SUT) is required.

To improve the test coverage while using less human resources, there is a need for a tool, which allows to define test scenarios including the expected system reactions in a simple and unambiguous way, automatically run the test scenarios, automatically evaluate and report the behavior of the system under test after each test run.

The **c**heck **c**ase **d**efinition **l**anguage (CCDL) is an approach to automate system level testing by providing a high level script language that allows defining test stimulations and expected results in a human readable form. The CCDL bridges the gap between a purely textual description of a test and the compilation into a test stimulation program required by any automated test execution tool. A well defined interface to the underlying test execution engine allows execution of CCDL written tests on any test tool that provides the required functionality.

Moreover, CCDL is embedded into a complete testing process starting from the definition of tests, linking tests to system requirements, executing tests and review as well as reporting of test results as shown in the figure below (the V model development process).



Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

The CCDL testing process provides open interfaces to test management solutions and it is already integrated into the **I**ntegrated **T**est **E**nvironment (ITE) from Razorcat Development GmbH which supports the whole testing life cycle according to the V model mentioned above.

The CCDL language provides means to link individual expected reactions of the system under test to the respective system requirements. Such traceability of test results to system requirements and vice versa is one of the most important issues arising while testing safety critical systems according to aerospace, automotive or medical standards.

2 Application Area of CCDL

The CCDL language is applicable for system testing where the SUT is seen as a black box with defined input and output interfaces. These interfaces are the only points of stimulation and check for expected system reactions. The internal behavior of the system is unknown and only described by more or less detailed system requirements and maybe other kind of documentation. These requirements are the base for all testing and though every test stimulation and checks for expected reactions within the CCDL test script may be linked directly to the system requirements.

When stimulating a system under test using CCDL, it may be possible to apply internal coverage measurements (e.g. branch or decision coverage of the software that controls the system) but this is not the scope of CCDL. Such measures may be applied additionally.

3 State of the Art System Testing

System testing requires a test bench hardware which is connected to the SUT and which provides means for measurement and control of the SUT (normally called Hardware-in-the-Loop, HIL) as well as software that controls the test execution. The hardware part is out of scope of the CCDL, because the CCDL may be executed on any suitable hardware platform. Moreover the software necessary to control the test execution is the essential target of the CCDL testing process.

Typical HIL systems provide a programming language for control of the testing process (e.g. C or Python). Such programming languages are designed for programming but they are not appropriate for testing. The control flow of a real-time program is completely different to the control flow required for conducting real-time tests. Writing tests in programming languages requires high level programming skills, which is not the primary scope of a tester.

Programming languages used for testing have the following disadvantages:

- programmer required, not a tester
- a lot of programming overhead even for simple tests
- poor documentation (program code is hard to understand)
- hard to review
- programmer needs a comprehensive understanding of both SUT and test bench

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

- no automated test evaluation
- no automated test reporting

Some HIL systems provide a graphical flow chart based programming environment. Such flow charts come a little closer to what the CCDL provides, but there is still programming required to implement the control flow for the flow chart elements.

Flow charts have the following disadvantages:

- Hiding information relevant for testing within flow chart elements (properties)
- Hierarchical structuring on different levels is complicated to understand
- Reporting and documentation is complicated

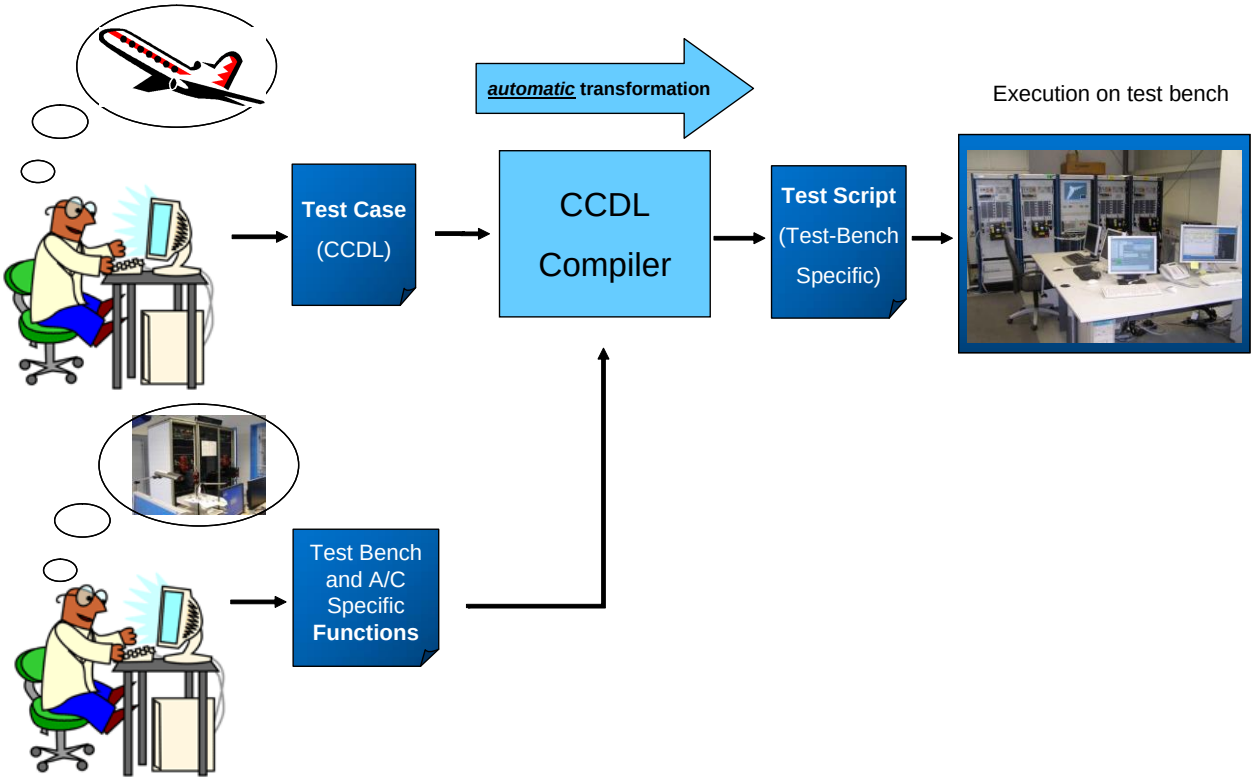
4 The CCDL Testing Process

4.1 Advantages of CCDL

When using CCDL, the test engineer does not need a comprehensive understanding of the test bench. He can focus on the SUT and define the test scenario with a dedicated, unambiguous test language, which is independent of the test bench. The test language is “high level”, easy to learn and intuitive to read, so that the CCDL written test scenarios can be used as test documentation. Test execution, evaluation and reporting can be done fully automatic. Open interfaces allow an integration of the CCDL into multiple test benches and test management process tools of different types.

Another challenge of system testing is the handling of both the complexity of the test bench as well as the complexity of the SUT itself. The test engineer should have knowledge about the SUT whereas the test bench is just a verification tool for him. But in most cases, the tester also needs to have deep insight into the test bench functionality and SUT specific internal behavior. The CCDL introduces a concept of splitting the tester’s work into writing of test cases (in CCDL) and on the other side programming of test bench or SUT specific CCDL user functions like shown within the figure below.

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com



This concept allows concurrent engineering of test cases and test bench specific functions. Also the skills required for either writing of tests or programming of CCDL user functions are different. A few highly skilled engineers are required to define and program the user functions while the testers need only testing skills. They may concentrate on the test of the SUT while specific testing functionality is encapsulated by CCDL user functions.

4.2 CCDL Compiler

The CCDL provides means to create test procedures with powerful language features while remaining readable and understandable by non-testers. It is possible to create sophisticated tests with only a few lines of CCDL statements.

The compiler creates C code programs running on any test execution environment (adaptable by a small abstraction layer). Standard features of test benches like relay matrixes, resistor banks and additional measurement devices are comfortably embedded into the CCDL language and may be applied directly using dedicated CCDL statements.

Benefits of the CCDL language:

- For test engineers: Easy to learn with minimum training effort
- For audit purposes: Easily understandable even without training
- Chronological as well as event based test stimulation
- Monitoring of expected system reactions asynchronously as well as synchronously to test stimulation
- Automatic test evaluation and failure reporting

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

Technical Features:

- Virtual (state) machine controls test execution in real time
- Abstraction layer allows execution on different test benches
- Specific functionality of the test bench is available via high-level CCDL functions

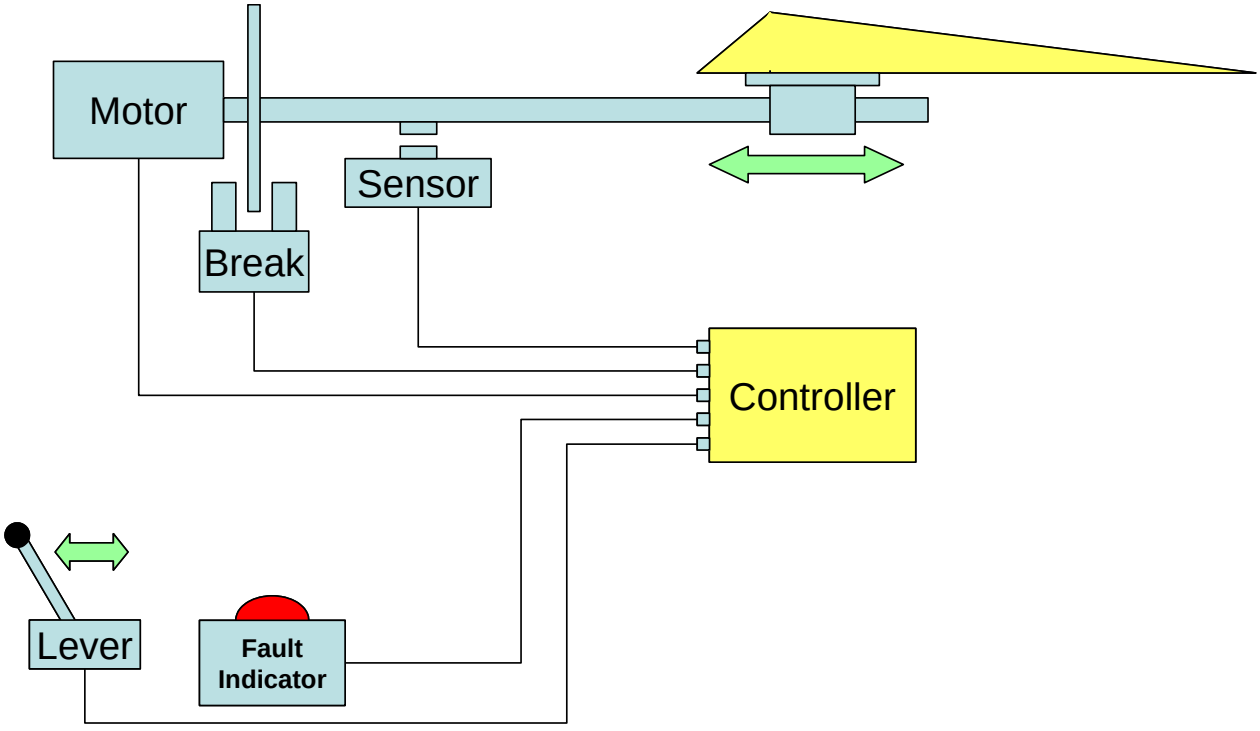
4.3 CCDL Editor/Debugger

This additional package provides a syntax controlled editor for CCDL procedures. It is seamlessly integrated into the test management (ITE) client and allows linking of requirements to individual CCDL statements (e.g. expected reactions of the system under test).

The package also includes a debugger for step by step execution and playback of recorded test runs. The user may review and play back recorded test runs based on the input and output data logged during test execution.

5 CCDL Sample

The following very simple actuator system of an airplane wing part shall illustrate the functionality of the CCDL. The system consists of a controller that controls the movements of a wing part depending on the lever setting (i.e. the lever is the input from the operator). The motor drives the wing part and the sensor measures speed and position of the system. The controller will be the SUT in the following example.



Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

The system shall be verified against the requirements given within the specification of the system. The default position of the lever is 0 and it may be moved to positions 1 and 2. This drives the motor until the wing part comes into the respective position.

5.1 Requirements of the Sample System

As an excerpt from the system specification, the following requirements for the controller were selected and they shall be verified by means of system testing:

- RQMT:0815-1 The motor shall operate the system at a speed of 1000 rpm
- RQMT:4711-1 If any overspeed (more than 1100 rpm) is detected, the system shall stop the motor and activate the break within 100 ms. A fault warning shall be indicated.

5.2 Definition of Tests

The next step in testing is the definition of test scenarios for the SUT. We will consider the following test definition for the overspeed tests:

- Reset the system to initial state and positions
- Set the lever position to position 1
- Wait until the motor has reached the normal speed (refer to requirement RQMT:0815-1)
- Simulate a sensor failure: Set the sensor to an offset of 110 rpm above the originally measured value (refer to requirement RQMT:4711-1)
- Check that the system gets stopped after 100 ms (refer to requirement RQMT:4711-1)

This test describes the steps to be taken in order to prepare the SUT for the test as well as the stimulation, error injection and the expected reaction of the SUT. The CCDL script will implement this test and provide means to automatically check the expected system reactions.

5.3 Initial Conditions of the Test

One of the prerequisites for the test are the initial conditions and settings of the SUT as well as the test bench. The CCDL provides the **Initial Condition** block to specify this initial setup for the test:

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

Demo_Sample_Ccdl_Whitepaper__213.ccd

```
CCD test.ccd

Initial Conditions:
{
    // Set default values for test environment
    set TES.ComputedAirSpeed to 120 [kts]
    set TES.Altitude to 8000 [ft]

    // Set default values for controller
    set CTRL.MotorSpeed to 0
    set CTRL.LeverPosition to 0
}
```

The controller is specified as **CTRL** whereas the test bench environment model is specified as **TES**. Parameters of both systems are initialized within the initial conditions block.

5.4 Test Steps

The stimulation of the test and the check for expected system reactions is carried out within test steps. The Test definition above may be tested with the CCDL implementation shown below

Demo_Sample_Ccdl_Whitepaper__213.ccd

```
Test Step 1, Timeout 99 [s]:
{
    // Action: Set lever position to 2
    set CTRL.LeverPosition to 2

    // Check for motor speed of system
    // - Set a trigger variable if the event occured
    set trigger T1 when CTRL.MotorSpeed >= 1000 [rpm]          (RQMT:0815-1)

    // When system is in state "ready for this test":
    // Set failure condition: Manipulate sensor
    when T1:
        // Manipulate sensor value
        set CTRL.MotorSpeed to offset 110 [rpm]                (RQMT:4711-1)

    // Check if system detects the failure condition
    within T1 .. T1 & 100 [ms]: {
        expect CTRL.BreakState      => ENGAGED                 (RQMT:4711-1)
        expect CTRL.FailureWarning  => 1                       (RQMT:4711-1)
    }
}
```

This test step stimulates the system, waits for the system to operate properly, then injects the failure condition and finally checks for the expected reactions of the SUT.

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

This simple example already outlines the powerful language features of CCDL: The **trigger** expression denotes a certain point in time where the respective condition is fulfilled. Based on this trigger, the stimulation (the **when** statement) and expected reaction checks (the **within** statement) will be carried out at point in time where the SUT is in the desired state for testing. Time intervals (**T1 .. T1 & 100 [ms]**) using trigger expressions and offsets allow precise expected reaction checks in real time. The expected reaction operator **=>** is applicable for boolean expressions. It checks whether the value changes exactly once from the negated boolean value to the boolean value specified in the expression (within the given time interval).

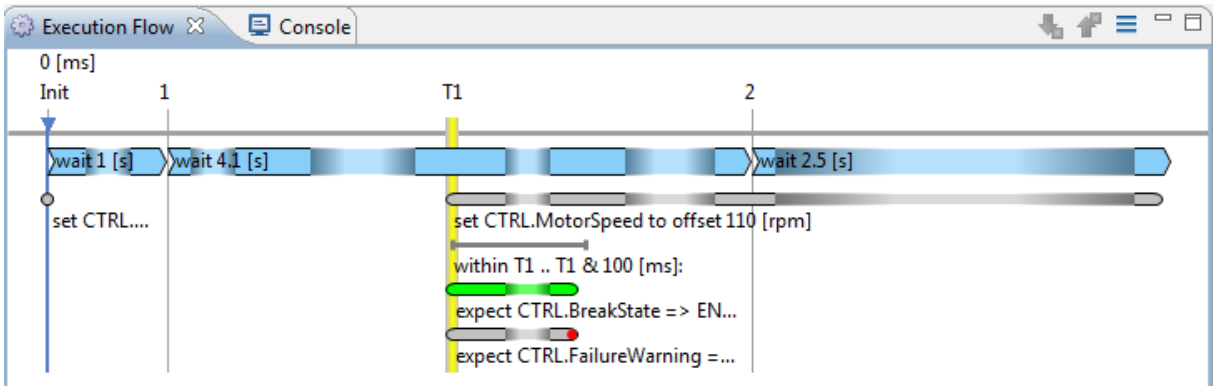
5.5 Test Preparation

Before executing the test, the CCDL script has to be compiled into an executable application that shall run on the test bench. The CCDL compiler produces C-Code that is executable on the test bench (through the CCDL runtime module and based on the adaptable interface library). It may be integrated into the normal compilation process of the test bench.

5.6 Test Execution Result

During execution of the test, the initial condition settings will be applied and all specified test steps will be executed one after another. Test steps have an optional timeout period which will abort the test if the execution time exceeds the specified time.

The CCDL development environment provides a visualization of the execution flow after the test is finished. Below is an example showing the temporal behavior of the test execution and the results of the expected reaction evaluations. It also shows the point in time where the trigger **T1** event condition was reached which in turn caused the subsequent manipulation and checking statements to be processed.



On successful test completion, the CCDL real time code generates an automatic evaluation result log file. This log file contains the procedure text and the passed/failed results of all expected reactions specified within the CCDL procedure.

Below is an excerpt of the result log file for the sample CCDL.

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

```
1  ------------------------------------------------
2    RESULT    |   EVAL COUNTER    |   PROCEDURE TEXT
3  ------------------------------------------------
4              |                   |   CCD test.ccd
5              |                   |
6  ...
7              |                   |
8              |                   |   Test Step 1, Timeout 99 [s]:
9              |                   |   {
10             |                   |    // Action: Set lever position to 2
11             |                   |    set CTRL.LeverPosition to 2
12             |                   |
13             |                   |    // Check for motor speed of system
14             |                   |    // - Set a trigger variable if the event
15             |                   |    set trigger T1 when CTRL.MotorSpeed >= 1
16             |                   |
17             |                   |    // When system is in state "ready for th
18             |                   |    // Set failure condition: Manipulate sen
19             |                   |    when T1:
20             |                   |       // Manipulate sensor value
21             |                   |       set CTRL.MotorSpeed to offset 110 [rp
22             |                   |
23             |                   |    // Check if system detects the failure c
24             |                   |    within T1 .. T1 & 100 [ms]: {
25   OK        |   (120/200)       |       expect CTRL.BreakState     => ENGAGED
26   FAILED    |   (0/200)         |       expect CTRL.FailureWarning => 1
27             |                   |    }
28             |                   |   }
29 ================================================
30  Expected Reactions:
31  Total :        2
32  Passed:        1
33  FAILED:        1
34
35  Automatic test result:     *** FAILED ***
36 ================================================
```

5.7 Requirements Coverage Result

Another result file contains the list of requirements attributed to the expected reaction checks of the CCDL procedure. A cumulated result value will be calculated for each requirement depending on the assigned expected reaction results. These automatically calculated requirement results will be propagated into the test management system (ITE) for further analysis. They may also be used for the later requirement based evaluation of the logged data of the test run.

6 CCDL Features

6.1 Chronological Test Execution

Each statement of the CCDL is executed one after another. Statements lasting longer than one time frame remain active until they are finished. This ensures

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

chronological execution of the CCDL statements. Only statements activated by trigger expressions are executed in parallel to the normal chronological control flow.

6.2 Trigger Functions for event-based Test Control

Trigger expressions allow event based stimulation or checks within the CCDL test control flow. The trigger is defined by a logical expression that is evaluated each time frame starting from the point in time where the CCDL control flow reached the trigger statement.

The CCDL statements within a trigger expression are executed when the trigger is activated (in parallel to the normal CCDL control flow). This may never happen if the trigger condition always fails.

6.3 Automatic Unit Conversion

Each parameter has its defined unit (given through the compiler configuration), but the tester may assign values using different units. The assigned values will then be converted automatically to the unit required for the parameter.

6.4 Multi-Parameter Access

A special naming convention allows accessing several parameters within a single line of code. Using the identifier “abc[1;2;3]def” denotes the following list of identifiers: abc1def, abc2def, abc3def. This shortens the CCDL script when testing safety critical SUTs where always a number of redundant parameters have to be stimulated or checked.

6.5 Monitoring of Parameters

The monitoring statement allows checking of parameters for the whole test execution or within a test step.

6.6 Checks for Parameter Change

Checking that a parameter changes its value from one value to another within a given time frame requires only a single line of CCDL code: The CCDL statement “expect param_x => 0” checks that the parameter changes its state exactly once from 1 to 0.

6.7 Parameter Checks within defined Time Frames

Parameter checks may be carried out for a certain period of time. CCDL may check that a parameter condition is valid:

- for the whole time period (“during” statement)
- at least once within the given time frame (“within” statement)

The time period for both statements may be defined using trigger expressions (combined with time offsets) which allows precise event based checks of parameter values.

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com

6.8 Requirement Links

The CCDL language allows attributing requirement links to each expected system reaction. These requirement links will be processed by the CCDL compiler in order to create a reference list of requirements. Further processing within a test management system provides means to trace the CCDL evaluation results back to the requirements for later test evaluation and review.

6.9 Automatic Test Evaluation

Each expected system reaction within the CCDL procedure is logged and summarized within the automatically calculated test evaluation result after the test is finished. This test result provides immediate feedback to the tester after each test run.

Razorcat Development GmbH · Witzlebenplatz 4 · 14057 Berlin · Tel. +49 (0) 30-536 357-0 · Fax +49 (0) 30-536 357–60
Geschäftsführer: Dipl.-Inform. Michael Wittner · Amtsgericht Berlin-Charlottenburg HRB 65326 · www.razorcat.com