

# Concept Manual vTESTstudio

Version 5.0  
English

## **Imprint**

Vector Informatik GmbH  
Ingersheimer Straße 24  
D-70499 Stuttgart

The information and data given in this user manual can be changed without prior notice. No part of this manual may be reproduced in any form or by any means without the written permission of the publisher, regardless of which method or which instruments, electronic or mechanical, are used. All technical information, drafts, etc. are liable to law of copyright protection.

© Copyright 2017, Vector Informatik GmbH. Printed in Germany.  
All rights reserved.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	About this User Manual	4
1.1.1	Access Helps and Conventions	4
1.1.2	Certification	5
1.1.3	Warranty	5
1.1.4	Support	5
1.1.5	Registered Trademarks	6
<b>2</b>	<b>Overview</b>	<b>7</b>
2.1	General	8
2.2	vTESTstudio and CANoe	10
2.3	Function Overview	10
2.4	Structure of a Project	11
2.4.1	Project Tree	11
2.4.2	File-Based Storage	11
2.4.3	Re-Use of Files	12
2.4.4	Home Directory and Libraries	12
2.4.5	Storage Location of Files on the File System	13
2.4.6	Structuring Using (Shareable) Folders	13
2.5	Concepts for High Test Coverage	13
2.6	Traceability by the Connection of REQM/TDM Tools	17
2.7	Test Design Documentation	20
<b>3</b>	<b>Test Design Editors</b>	<b>23</b>
3.1	CAPL Editor	24
3.2	C# Editor	25
3.3	Test Table Editor	26
3.4	Test Diagram Editor	27
3.5	State Diagram Editor	28
<b>4</b>	<b>Interaction with CANoe</b>	<b>29</b>
4.1	CANoe System Environment	30
4.2	Execution in CANoe	32
4.3	Reporting	34
<b>5</b>	<b>Language Interaction</b>	<b>35</b>
5.1	Interface Functions	36
<b>6</b>	<b>Parameters, Curves and Variants</b>	<b>39</b>
6.1	Parameters	40
6.1.1	Concept	40
6.1.2	Find Test Case Data by the Classification Tree Method	42
6.2	Curves	43
6.3	Variants	44

<b>7</b>	<b>Use Cases</b>	<b>47</b>
7.1	Generating Two Similar Test Units for Different OEMs	48

# 1 Introduction

In this chapter you find the following information:

---

1.1	About this User Manual	page 4
	Access Helps and Conventions	
	Certification	
	Warranty	
	Support	
	Registered Trademarks	

---

## 1.1 About this User Manual

### 1.1.1 Access Helps and Conventions

To find information quickly

The user manual provides you the following access helps:

- > at the beginning of each chapter you will find a summary of its contents,
- > in the header you see the current chapter and section,
- > in the footer you see to which program version the user manual replies,
- > at the end of the user manual you will find an index.







**Reference:** Please refer to the online help for detailed information on all topics.

Conventions

In the two following charts you will find the conventions used in the user manual regarding utilized spellings and symbols.

Style	Utilization
<b>bold</b>	Blocks, surface elements, window- and dialog names of the software. Accentuation of warnings and advices. <b>[OK]</b> Push buttons in brackets <b>File Save</b> Notation for menus and menu entries
CANoe	Legally protected proper names and side notes.
Source code	File name and source code.
Hyperlink	Hyperlinks and references.
<CTRL>+<S>	Notation for shortcuts.

Symbol	Utilization
	Here you can obtain supplemental information.
	This symbol calls your attention to warnings.
	Here you can find additional information.
	Here is an example that has been prepared for you.
	Step-by-step instructions provide assistance at these points.
	Instructions on editing files are found at these points.
	This symbol warns you not to edit the specified file.
	This symbol indicates multimedia files like e.g. video clips.

Symbol	Utilization
	This symbol indicates an introduction into a specific topic.
	This symbol indicates text areas containing basic knowledge.
	This symbol indicates text areas containing expert knowledge.
	This symbol indicates that something has changed.

### 1.1.2 Certification

#### Certified Quality Management System

Vector Informatik GmbH has ISO 9001:2008 certification.  
The ISO standard is a globally recognized quality standard.

### 1.1.3 Warranty

#### Restriction of warranty

We reserve the right to change the contents of the documentation and the software without notice. Vector Informatik GmbH assumes no liability for correct contents or damages which are resulted from the usage of the user manual. We are grateful for references to mistakes or for suggestions for improvement to be able to offer you even more efficient products in the future.

### 1.1.4 Support

#### You need support?

You can get through to our hotline at the phone number  
+49 (711) 80670-200  
or you send a problem report to the **CANoe Support**.

### 1.1.5 Registered Trademarks

#### Registered trademarks

All trademarks mentioned in this user manual and if necessary third party registered are absolutely subject to the conditions of each valid label right and the rights of particular registered proprietor. All trademarks, trade names or company names are or can be trademarks or registered trademarks of their particular proprietors. All rights which are not expressly allowed are reserved. If an explicit label of trademarks, which are used in this user manual, fails, this should not mean that a name is free of third party rights.

- > **Windows**, **Windows Vista**, **Windows 7** and **Windows 8** are trademarks of the Microsoft Corporation.
- > **vTESTstudio** is a trademark of Vector Informatik GmbH.



## 2 Overview

This chapter contains the following information:

---

2.1	General	page 8
2.2	vTESTstudio and CANoe	page 10
2.3	Function Overview	page 10
2.4	Structure of a Project	page 11
	Project Tree	
	File-Based Storage	
	Re-Use of Files	
	Home Directory and Libraries	
	Storage Location of Files on the File System	
	Structuring Using (Shareable) Folders	
2.5	Concepts for High Test Coverage	page 13
2.6	Traceability by the Connection of REQM/TDM Tools	page 17
2.7	Test Design Documentation	page 20

---

## 2.1 General

Overview	<p>vTESTstudio is a multifaceted and integrated work environment for developing tests for embedded systems.</p> <p>You can use different test design languages to write tests in vTESTstudio. The following editors are supported:</p>
Test design editors	<ul style="list-style-type: none"><li>&gt; <b>CAPL Editor:</b> CAPL is an event-oriented programming language of CANoe that can also be used for programming test sequences, test cases, and functions.</li><li>&gt; <b>C# Editor:</b> C# is a .NET programming language that CANoe expands with libraries especially designed for testing and for accessing control units.</li><li>&gt; <b>Test Table Editor:</b> This editor enables writing of tests in tabular form and can be used without the need for programming expertise.</li><li>&gt; <b>Test Diagram Editor:</b> Editor for modeling of test sequences in a graphic notation. This editor is contained in the <b>Graphical Test Design</b> product option.</li><li>&gt; <b>State Diagram Editor:</b> Editor for modeling the expected behavior of the SUT as state model for automatic test case generation. This editor is contained in the <b>Graphical Test Design</b> product option.</li></ul>
Re-use concept	Through the use of libraries and a clear and easy to understand re-use concept, whole files as well as individual test cases can easily be re-used in various tests.
Structured test creation	For a structured test creation tests are supported that can contain several files.
Parameterization	Due to an integrated parameter concept, ECU configuration parameters and test vectors can be administrated comfortable in separated files. From these files they can be re-used in different tests.
Stimulation curves	By the use of the Waveform Editor you can define curves to be used in the test as stimulation curves for the system under test.
Variants	A continuous variant support for test logic, test implementation and parameter values allows an implementation of variant-dependent tests.

## User interface

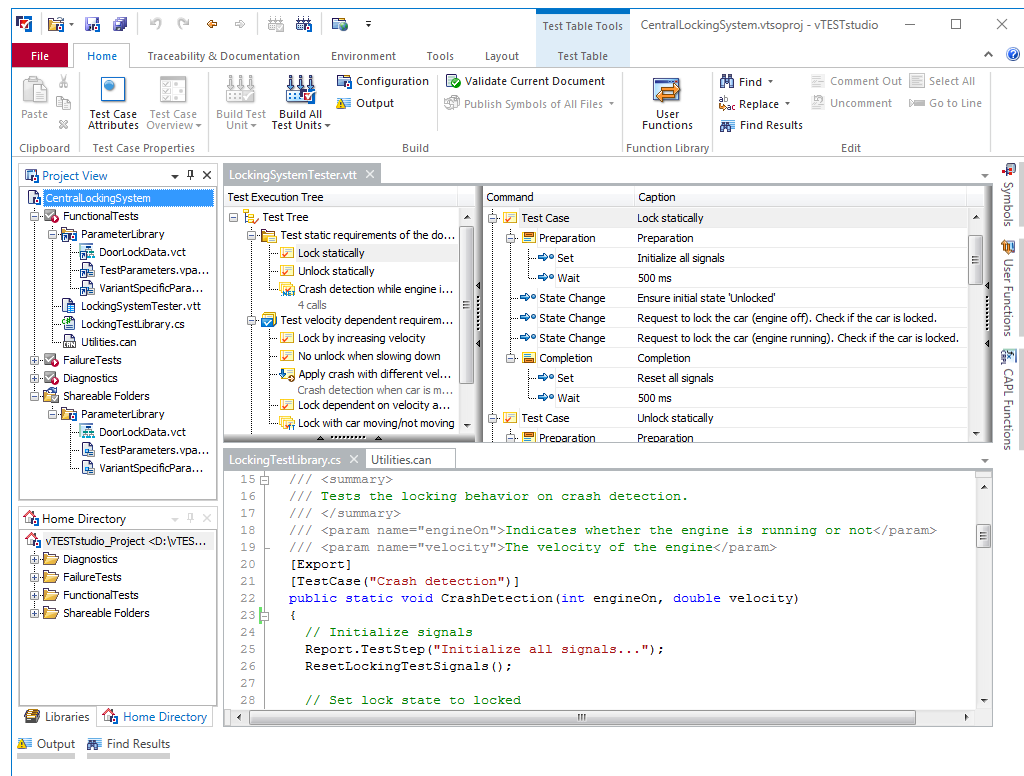


Figure 1: vTESTstudio user interface

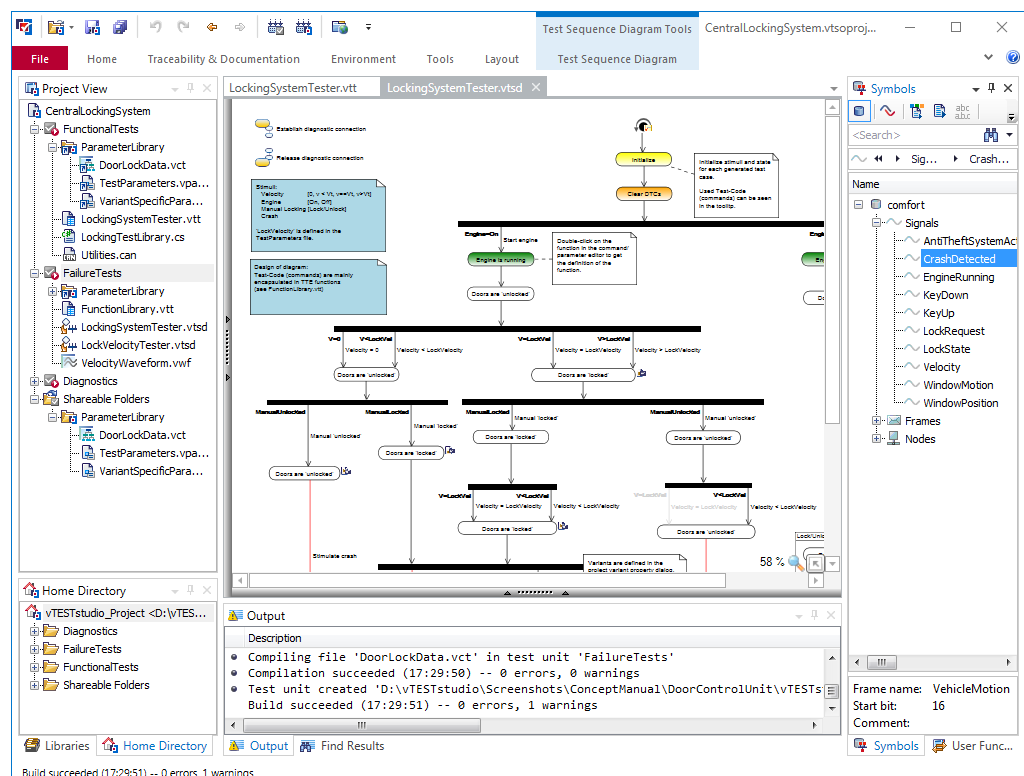


Figure 2: vTESTstudio user interface – Test Diagram Editor

## 2.2 vTESTstudio and CANoe

### Hand in hand

The interaction with **CANoe** enables easy access to the system under test in all languages. Database symbols, system variables, and diagnostic descriptions can be easily accessed in all languages. The bus systems and protocols CAN, LIN, FlexRay, Ethernet, WLAN and J1939 are supported.

Independent of their notation the tests are configured and loaded in **CANoe**, executed in real time, and documented in a detailed test report.

### Schema

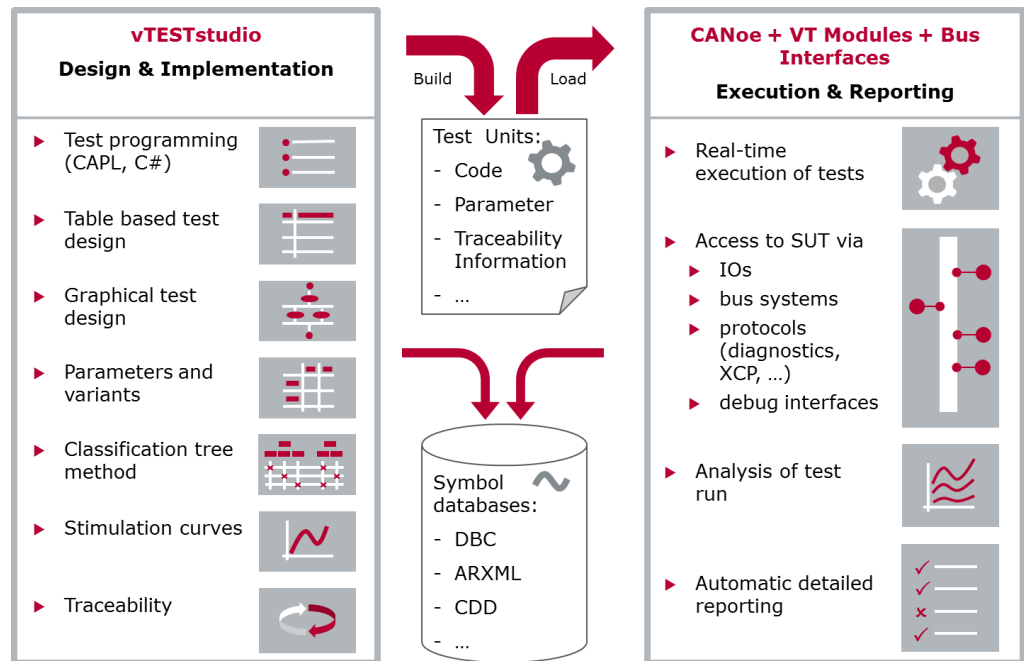


Figure 3: Schematic Overview – vTESTstudio and CANoe

## 2.3 Function Overview

### Important features

Among the things that **vTESTstudio** provides for a successful test environment are:

- > Integration of graphical test design editors, programming editors, parameter editors and curve editors in one tool
- > Comfortable capabilities for reviews by clear modeling of the test logic in the Test Diagram Editor or the expected behavior of the SUT in the State Diagram Editor
- > Easy re-use through possible separation of test logic, implementation, parameter values, and test vectors
- > Possible pre-processing of events in CAPL and C# for access and evaluation in test sequences
- > Integrated variant support for test structure, implementation, and parameters
- > Concepts for high test coverage
- > Universal traceability of externally defined requirements and test descriptions in the test implementation and in the test report
- > Automatically generated test design documentation for documentation purposes as well as internal and external reviews

## 2.4 Structure of a Project

### 2.4.1 Project Tree

#### Terms **project** and **test unit**

A **vTESTstudio** project consists of any number of **test units**. A test unit represents the configurable and executable unit in **CANoe**. It consists of a set of files that contain the test implementation (CAPL files, C# files, test tables, test diagrams and parameter files).

While a **vTESTstudio** project, for example, covers an ECU to be tested, a test unit could have implemented the tests for a particular functionality of the system under test.

#### Examples

##### Example 1

- > Project for tests of the overall functionality of an ECU
- > Test units for core features, error detection, diagnostics, ...

##### Example 2

- > Project for tests of the entire product line of an ECU
- > Test units for tests of a variant of an ECU

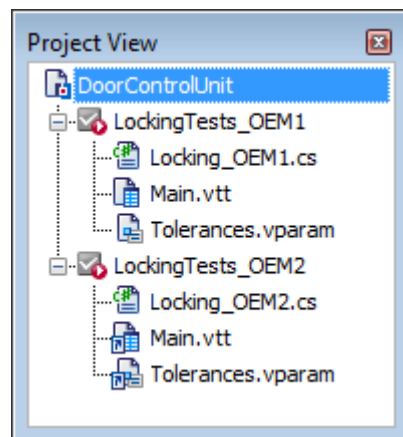


Figure 4: Project view

### 2.4.2 File-Based Storage

#### Storage of project components

All **vTESTstudio** project components are stored using **file-based** storing. A **vTESTstudio** project consists of a project file and any number of source code and parameter files that are referenced in the project file.

Files relevant for the test (CAPL files, C# files, test table files, parameter files, etc.) are created in the context of a test unit. Within a test unit, for example, test cases and test functions can be accessed across file- and language boundaries. Similarly, the parameters of a parameter file are known (only) in the context of the corresponding test unit.

### 2.4.3 Re-Use of Files

#### Linking of files

Files can easily be re-used by linking them to other test units. This is done by moving them from the source to the destination using a drag & drop operation. Thus, for example, the same test sequence can be re-used in multiple test units, e.g. with only using different parameter files for parameterizing the test sequence.

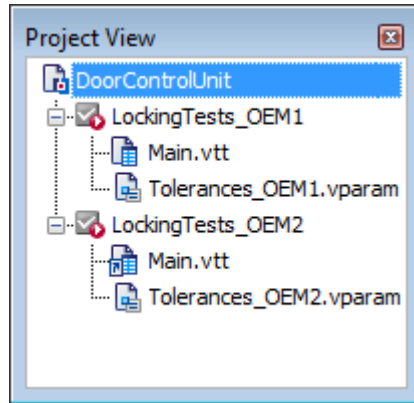


Figure 5: Project view with re-used file Main.vtt

### 2.4.4 Home Directory and Libraries

#### Home directory

The directory where the project file is located is called the **Home Directory**. Files from the root directory and files from specially defined **Libraries** can be used in the project.

#### Libraries

The Libraries view provides a filtered view of the file system. A library represents only a folder on the file system that is made known in the project. Files from libraries can also be linked to test units using a drag & drop operation.

If a library folder is moved on the file system, in **vTESTstudio** only the link to the library has to be updated. The paths of the linked files in the test units are updated automatically.

## 2.4.5 Storage Location of Files on the File System

### Storage of files

If a file is created in the context of a test unit, it is automatically created on the file system under the path `<Home Directory>\<Test Unit Name>`.

If the files are to be organized differently on the file system, this can be done using the Home Directory view. This view shows the file system located underneath the project directory. If a file is moved in this view, possible links to this file in the test units will be automatically updated.

## 2.4.6 Structuring Using (Shareable) Folders

### Re-use of folders

In order to structure the contents of test units more clearly, folders can be created within a test unit (as many folders, including nested folders, as required). This has no effect on the content of the test and is only used to organize files.

In addition to re-use individual files, the re-use of whole folders is also possible. For example, if a folder contains a test sequence description and an associated parameter file, these two files can be re-used together by linking the whole folder to another test unit.

If a folder with files is created underneath a test unit and then linked to another test unit, the folder with the original files is located on the file system underneath the first test unit. If such a "master test unit" is not desired, the folder can be created as a **shareable folder** instead. This is done by the project's shortcut menu command **New Shareable Folder**. Folders and files created by this are created on the file system under the path `<Home directory>\<Shareable Folders>`. From there they can also be linked to the test units using a drag & drop operation.

## 2.5 Concepts for High Test Coverage

### Parameterized test case lists

For high test coverage without significant programming effort, so-called parameterized test case and test sequence lists are supported.

To define a test case list, a test case with input parameters must be defined, for example, in C#:

```
[Export][TestCase]
public static void CheckMotor(int temperature, double voltage)
{
    // ...
}
```

The test case list for this can be defined in the **Test Table Editor**, for example. The command selection contains a **CheckMotor [list]** entry for this purpose. If this entry is chosen, an editor appears that can be used to assign multiple values for each test case parameter:

Combinatorics:	Sequential	
Name:	<b>temperature</b>	<b>voltage</b>
Type of Values:	Single Values	Value Range
Value Source:	-	[min=1, max=11, inc=0.5]
Values:	-20	1
	-15	1.5
	0	2
	10	2.5
	40	3
	Add value...	3.5
		4
		4.5
		...
		11

Figure 6: Parameterized test case list in the Test Table Editor

In addition to the input of individual values or a range, the use of list parameters and struct list parameters from parameter files is supported as well (also see section [Parameters](#)):


Combinatorics:	Sequential	
Name:	<b>temperature</b>	<b>voltage</b>
Type of Values:	List from Parameter File	Value Range
Value Source:	 TemperatureList	[min=1, max=11, inc=0.5]
Values:	-20	1
	-15	1.5
	0	2
	10	2.5
	40	3
		3.5
		4
		4.5
		...
		11

Figure 7: Using list parameters in test case lists

By a sequential combination, a number of test case calls are now generated automatically so that each parameter values is used at least once:



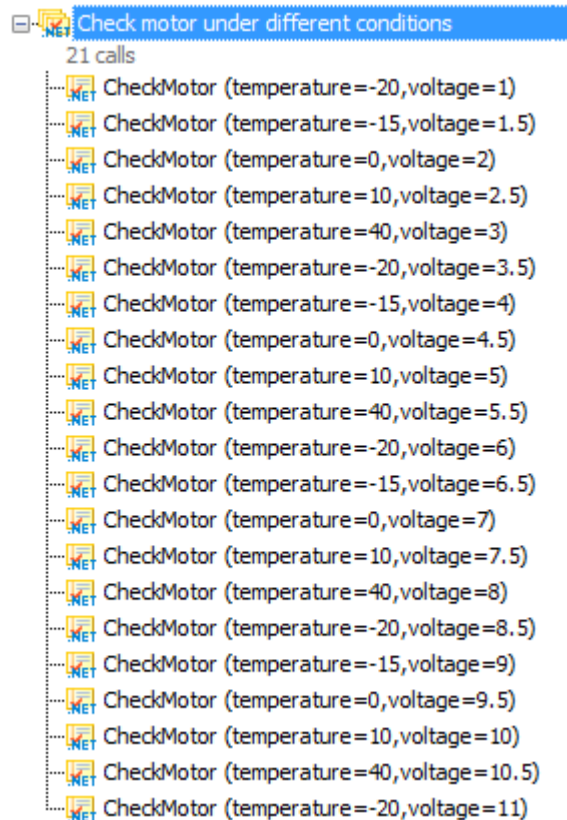


Figure 8: Generated test case list using **Sequential**

If a higher test coverage is to be achieved by testing pairwise or every combination of parameter values, the **Combinatorics** property must be changed from **Sequential** to **Pairwise** or **Combinatorial**:

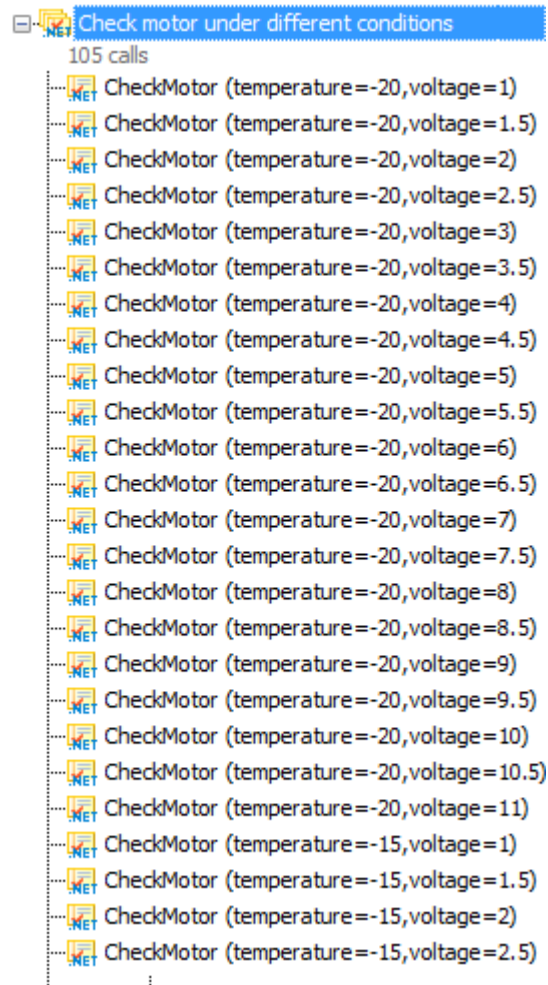


Figure 9: Generated test case list using **Combinatorial**

For the definition of a test case list in the Test Table Editor or in C#, all language combinations or crossovers listed in [Language Interaction](#) are supported.

## 2.6 Traceability by the Connection of REQM/TDM Tools

### Concept

You can use **vTESTstudio** to trace externally defined requirements and test descriptions during test implementation and in the test report (traceability). This is done using exchange files in an open XML format. As a result, any REQM/TDM system can be used coupled with **vTESTstudio**.

### Schema

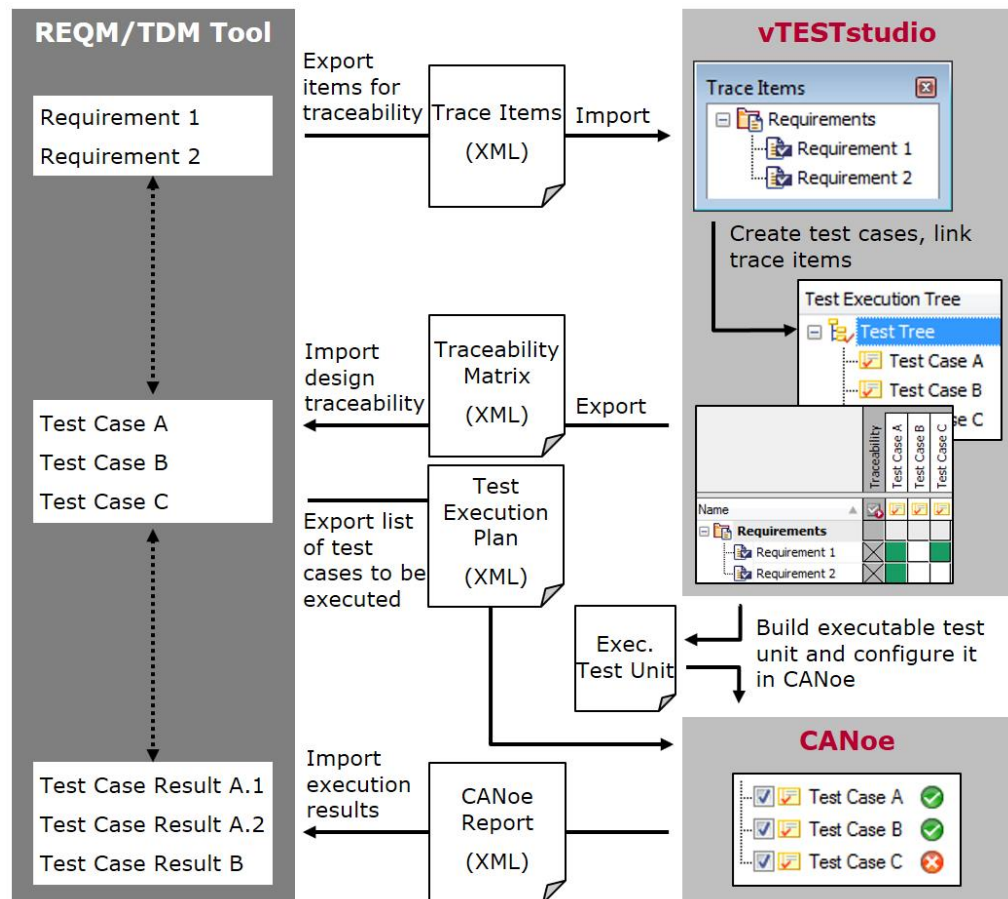


Figure 10: Traceability workflow

### Import of requirements or test descriptions in vTESTstudio

The information exported from the REQM/TDM system is imported into **vTESTstudio** and can be linked there to test cases. The linked elements can be requirements or test descriptions. Because these elements are used to achieve traceability, they are designated as **trace items**.

### Trace Item Explorer

All trace items imported into a **vTESTstudio** project are available via the Trace Item Explorer. From there, they can be easily linked to test cases using drag & drop.

In the REQM/TDM system, requirements and test descriptions might be structured in hierarchical folders or comparable elements. These folders are displayed in the Trace Item Explorer in the same hierarchy. However, folders cannot be directly linked to test cases.

### Traceability Matrix

The **Traceability Matrix** in **vTESTstudio** gives an overview over all trace items of the project and their test case links. By this overview of test design coverage you can easily see, for example which trace items are not covered yet by any test case implementation. Furthermore, from within the traceability matrix comfortable navigation from a trace item to linked test cases is possible. For documentation purposes the traceability matrix can be exported to **Excel**.

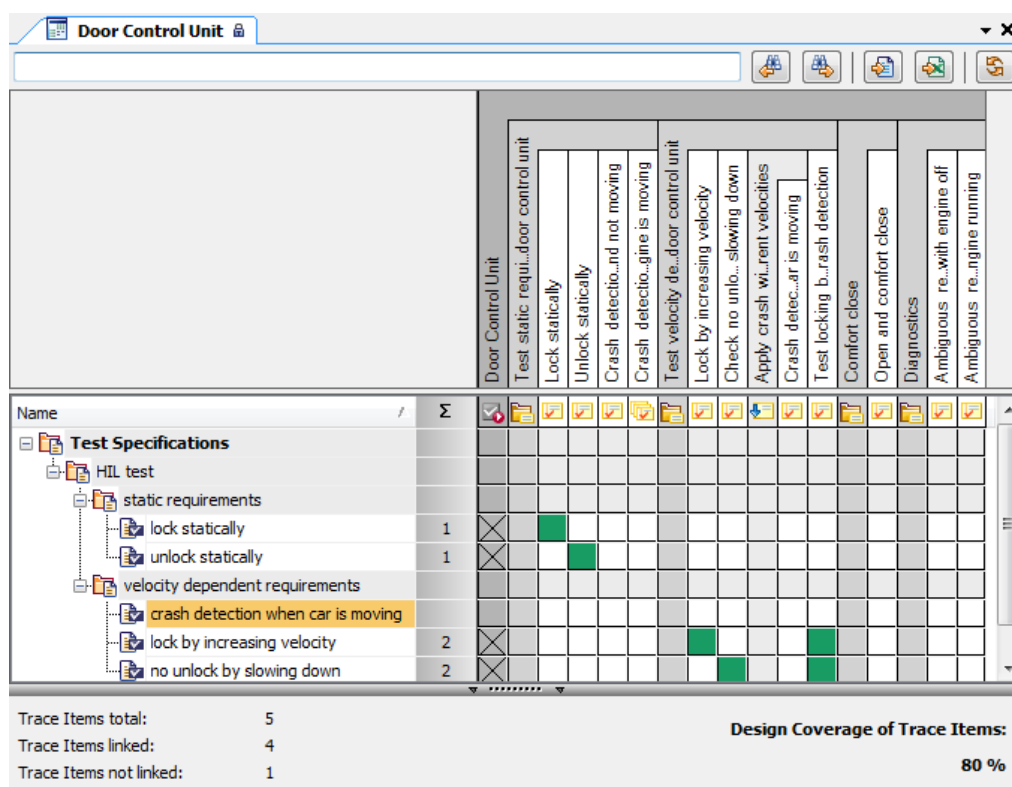


Figure 11: Traceability

### Import of design traceability in REQM/TDM system

For synchronization with the REQM/TDM system an export of the traceability matrix to XML is supported. This enables importing the information about implemented test cases and linked requirements or test descriptions to the REQM/TDM system.

### Planning of test execution

Based on this information test case execution can be planned, i.e. which test cases shall be executed. With the help of a so-called "test execution plan" the selection and execution of test cases in CANoe can be automated.

### Test report

For each trace item linked to a test case, the test report contains a corresponding reference at the test case. This enables traceability from the test case result back to the requirement or test description.

### Import of test results

The test report can be played back to the REQM/TDM system to be able to analyze

in REQM/TDM  
system

test results within this system.

Supported  
REQM/TDM systems

The connection of the following systems is supported out-of-the-box:

- > IBM DOORS (classic)
- > IBM DOOR NG und Rational Quality Manager
- > Siemens Polarion ALM
- > PTC Integrity Lifecycle Manager
- > Jama Connect
- > Intland codeBeamer ALM

So-called "Connection Utilities" are available for free. By a command line interface, they can be used for continuous integration and continuous testing as well (e.g. with Jenkins).

Further REQM/TDM systems can be connected by the use of the open interfaces.

## 2.7 Test Design Documentation

### Concept

For a test unit in vTESTstudio a test design documentation in PDF format can be generated automatically. This test design documentation provides an overview of all implemented test cases and test steps. Beyond the documentation it can be used for internal and external reviews of the test design.

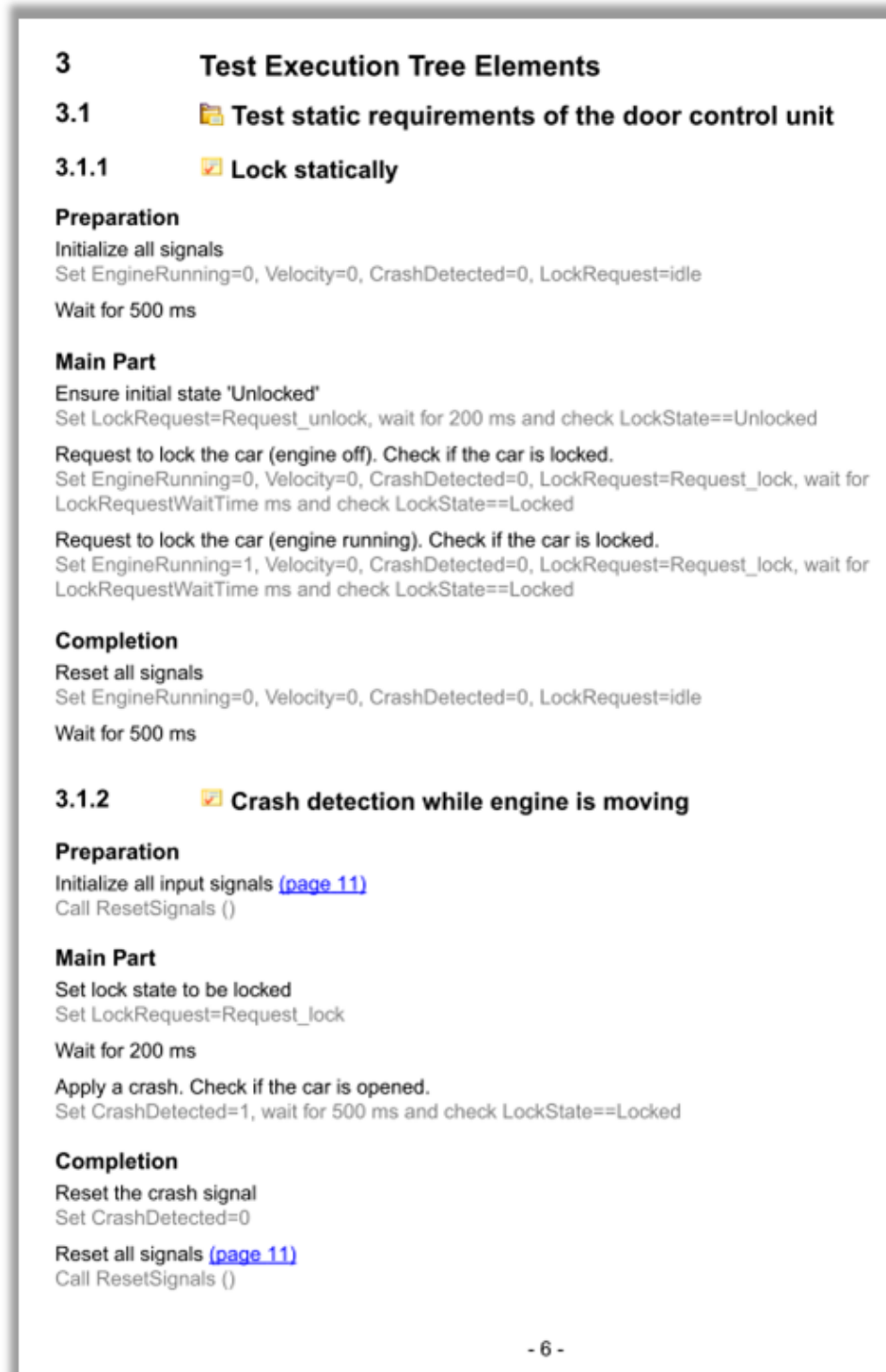


Figure 12: Automatically generated test design documentation as PDF







## 3 Test Design Editors

In this chapter you find the following information:

---

3.1	CAPL Editor	page 24
3.2	C# Editor	page 25
3.3	Test Table Editor	page 26
3.4	Test Diagram Editor	page 27
3.5	State Diagram Editor	page 28

---

## 3.1 CAPL Editor

### Features

The CAPL Editor integrated in **vTESTstudio** provides the functions of a modern development environment such as

- > Code completion and syntax checking while writing
- > Configurable syntax highlighting
- > Syntax-sensitive indentation
- > Expandable function blocks and function reference in a tree structure for faster navigation
- > Hierarchical function list with search function for direct transfer to the source text
- > Direct transfer of network symbols, environment data, diagnostic symbols, and parameters from the Symbol Explorer

### User interface

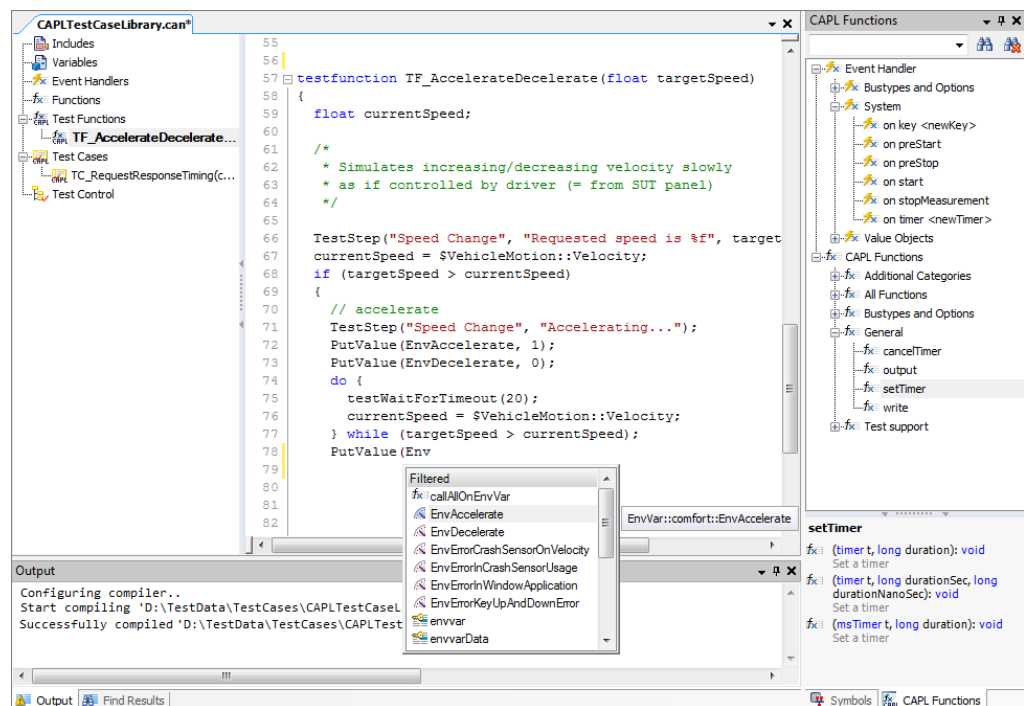


Figure 13: CAPL Editor

### Access to symbols

Direct access to signals, frames and environment data is possible by the CAPL API. This enables e.g. read and write access to signal values in the **CANoe** runtime environment. Example:

```
$Velocity = 100;
```

### Event procedures

In addition to the programming of sequential test sequences, test cases, and functions, the programming of event procedures is possible. In order to react to events (e.g., a signal change) in **CANoe**, event handlers can be defined. The methods are then called as soon as the event occurs during the test.

## 3.2 C# Editor

### Features

Similar to the CAPL Editor, the C# Editor integrated in vTESTstudio provides the functions of a modern development environment such as

- > Code completion and syntax checking while writing
- > Configurable syntax highlighting
- > Syntax-sensitive indentation
- > Expandable function blocks
- > Hierarchical function list with search function for direct transfer to the source text
- > Direct transfer of network symbols, environment data, diagnostic symbols, and parameters from the Symbol Explorer

### User interface

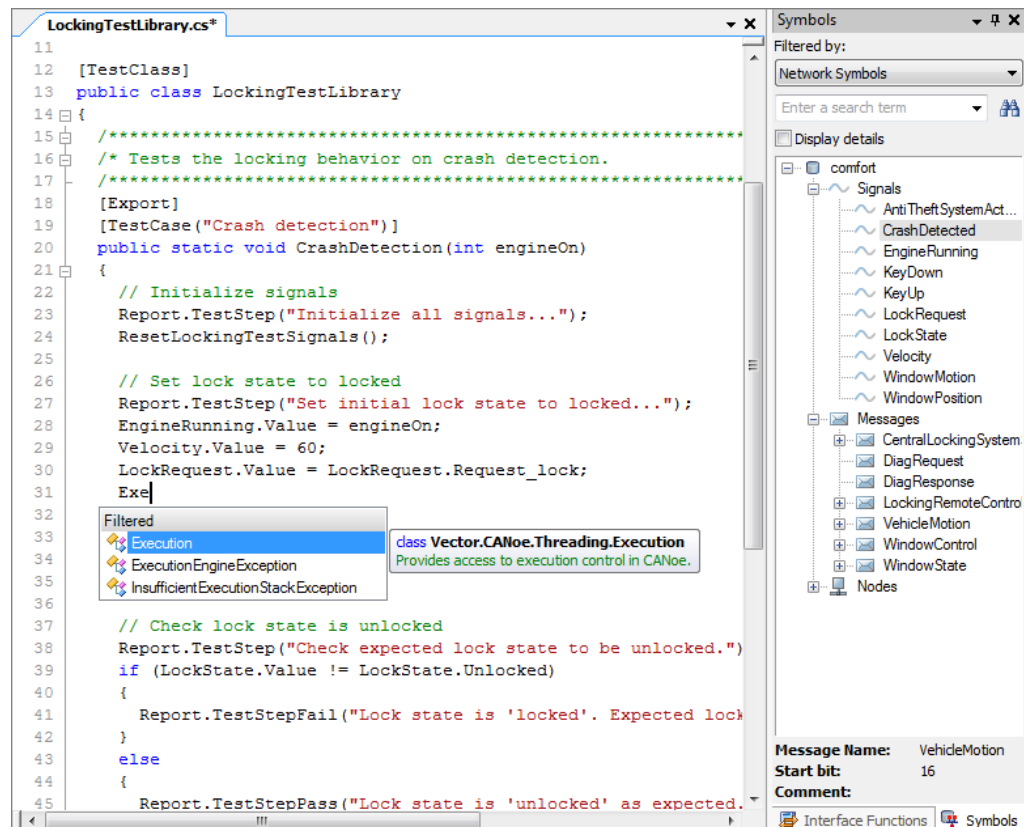


Figure 14: C# Editor

### Access to symbols

Automatically generated .NET classes are available for accessing signals and environment data as well as CAN frames. These enable access to the values in the CANoe runtime environment. Example:

```
Velocity.Value = 100;
```

### Event procedures

In addition to the programming of sequential test sequences, test cases, and functions, the programming of event procedures is also possible with the C# Editor. In order to react to events (e.g., a signal change) in CANoe, event handlers can be defined by the use of a special C# attribute. The methods are then called as soon as the event occurs during the test (exactly the same as in CAPL).

### Connection to MS Visual Studio

By an integrated connection it's also possible to use Microsoft Visual Studio for the development of tests in C# as an alternative to the C# editor in vTESTstudio.

### 3.3 Test Table Editor

#### Overview

The Test Table Editor allows users to easily define test sequences in tabular form without the need for programming expertise. Special commands are available for stimulating and testing the system under test.

#### Features

To make work easy, the following features are available:

- > Command and symbol completion while writing
- > Direct transfer of network symbols, environment data, diagnostic symbols, and parameters from the Symbol Explorer
- > Symbol and value type checks while editing
- > Access to user-defined functions and test cases in other languages
- > Ease of operation using both the keyboard and the mouse

#### User interface

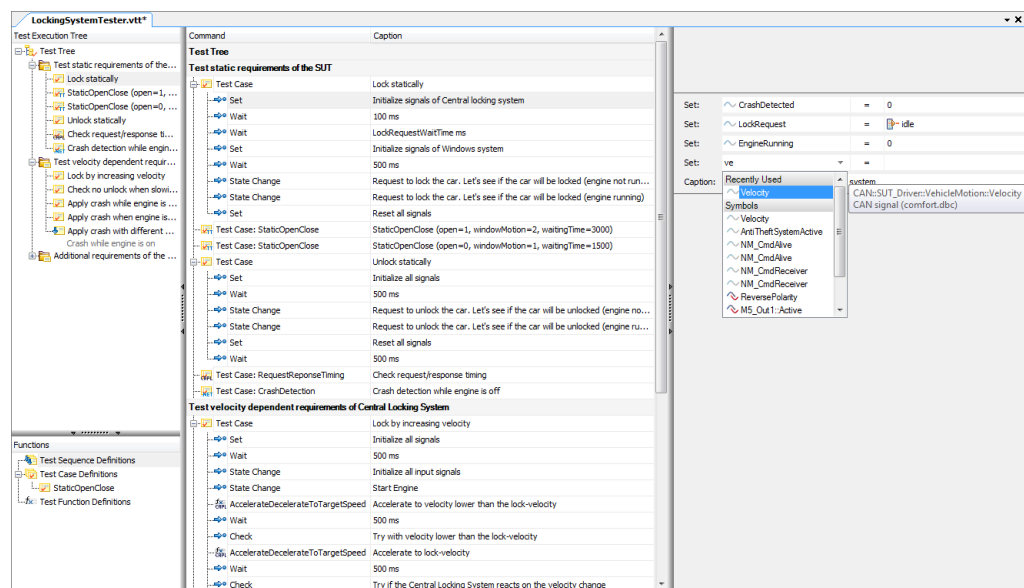


Figure 15: Test Table Editor

## 3.4 Test Diagram Editor

### Overview

The Test Diagram Editor can be used to define tests in a graphical way. The graphical notation is particularly suitable for reviews. Test code in tabular notation is located behind each graphical element. By default one test case is generated for each path through the diagram; a more fine-grained test case definition is possible as well.

### Features

To make work easy, the following features are available:

- > Easy configuration through the insertion of graphic elements into the graphic interface using a drag & drop operation
- > Command and symbol completion while writing
- > Direct transfer of network symbols, environment data, diagnostic symbols, and parameters from the Symbol Explorer
- > Symbol and value type checks while editing
- > Access to user-defined functions and test cases in other languages
- > Usage and re-use of sub-diagrams possible
- > Display and preview of generated test cases and their content in table form

### User interface

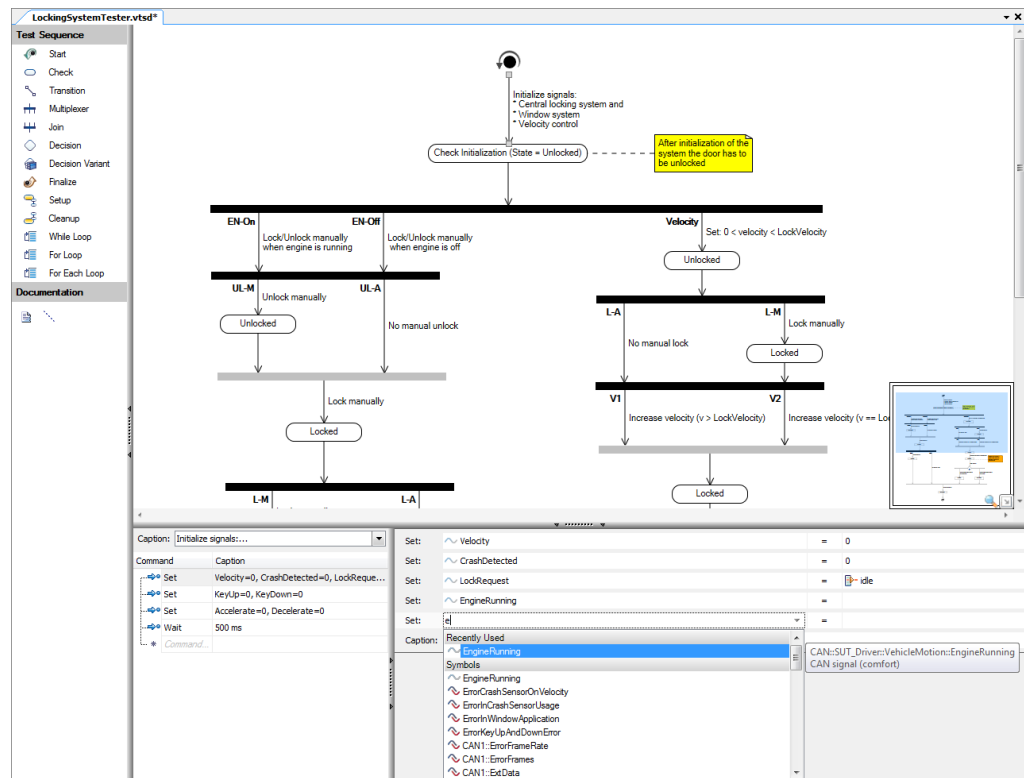


Figure 16: Test Diagram Editor



**Note:** The Test Diagram Editor is only available with the option **Graphical Test Design**.

## 3.5 State Diagram Editor

### Overview

The State Diagram Editor can be used to model the expected behavior of the system under test as state model. Test code in tabular notation is located behind each graphical element. Based on transition coverage test cases are automatically generated out of the model. Different generation algorithms are supported, e.g. Chinese Postman algorithm and a breadth search based algorithm.

### Features

To make work easy, the following features are available:

- > Easy configuration through the insertion of graphic elements into the graphic interface using a drag & drop operation
- > Command and symbol completion while writing
- > Direct transfer of network symbols, environment data, diagnostic symbols, and parameters from the Symbol Explorer
- > Symbol and value type checks while editing
- > Access to user-defined functions and test cases in other languages
- > Display and preview of generated test cases and their content in table form

### User interface

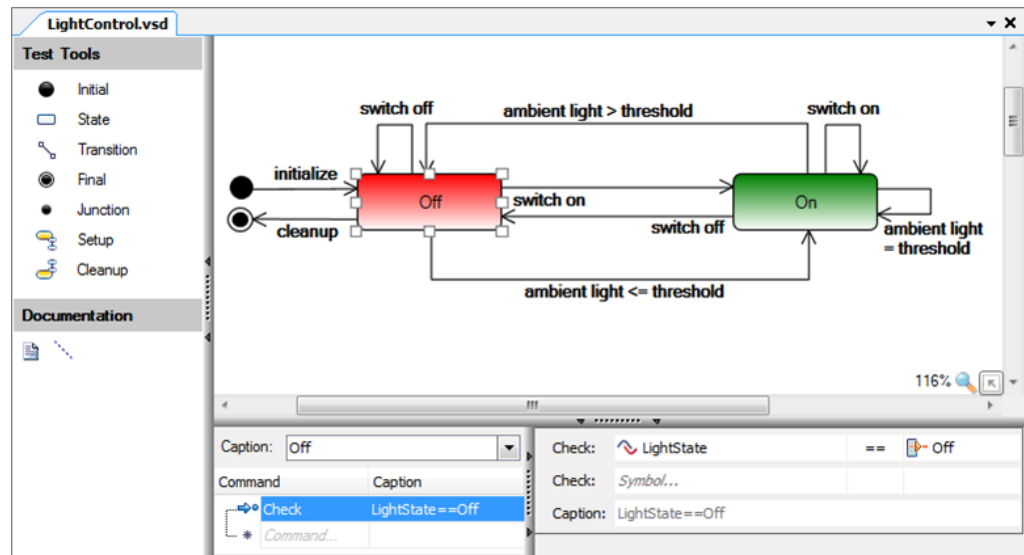


Figure 17: State Diagram Editor



**Note:** The State Diagram Editor is only available with the option **Graphical Test Design**.

## 4 Interaction with CANoe

In this chapter you find the following information:

---

4.1	CANoe System Environment	page 30
4.2	Execution in CANoe	page 32
4.3	Reporting	page 34

---

## 4.1 CANoe System Environment

### Access

The integration of description files for the system environment (databases, diagnostic descriptions...) enables easy direct access to symbols of the **CANoe** system environment. The database symbols, system variables, etc., contained in the system environment are available for use in the test coding. The symbols can be inserted into the test sequences in two different ways: via text completion in the individual editors or using a drag & drop operation from the Symbol Explorer.

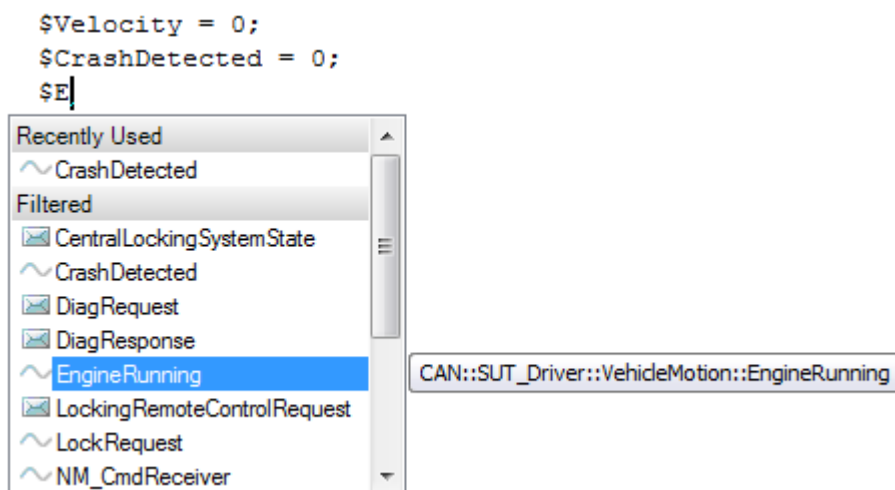


Figure 18: Symbol completion in the programming editors

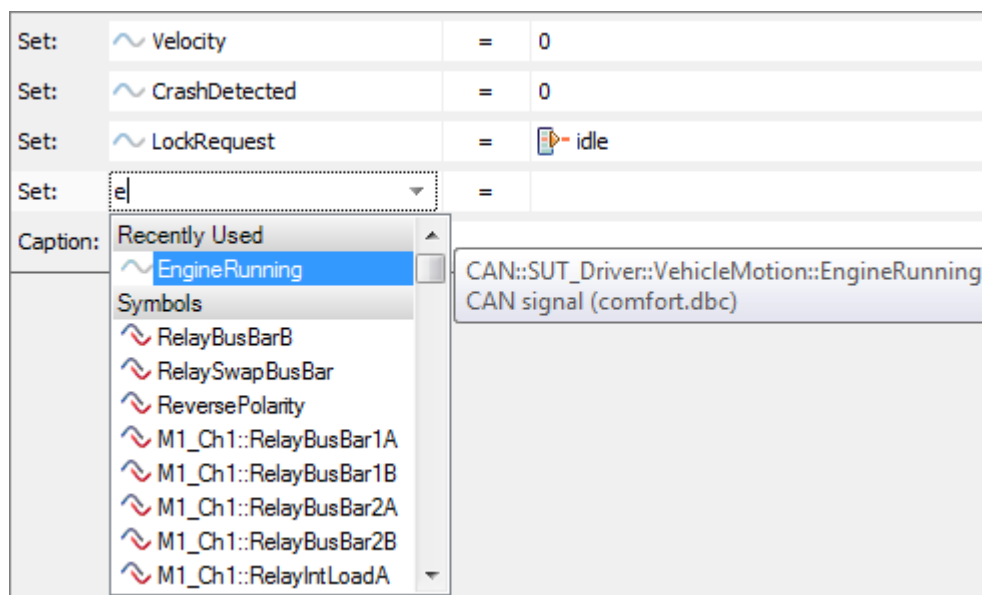


Figure 19: Symbol completion in the Test Table Editor



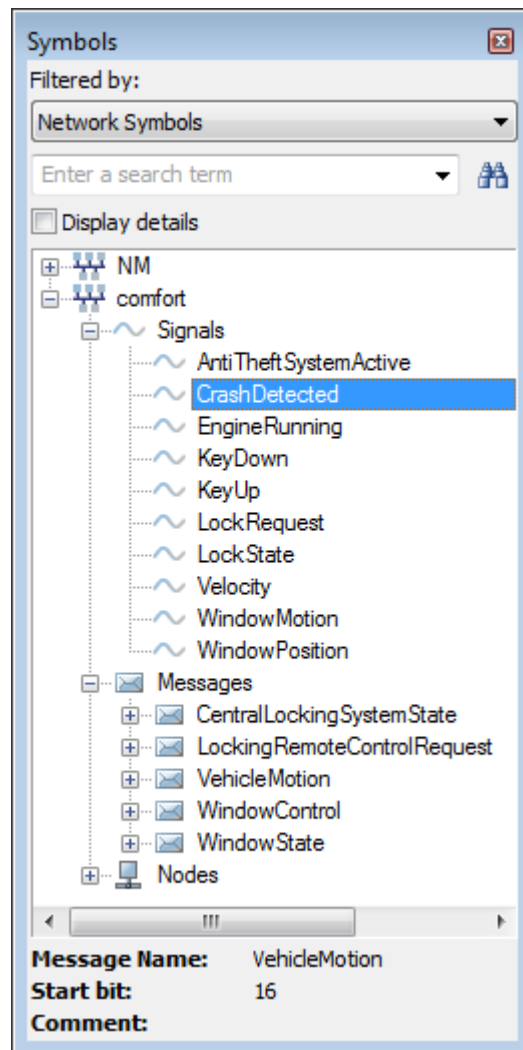


Figure 20: Symbol Explorer for inserting symbols using drag &amp; drop

## 4.2 Execution in CANoe

### Creating a test unit

In **vTESTstudio** an executable test unit can be created. An executable test unit has the extension \*. VTUEXE and contains all necessary data for the test (source files, parameter files...).

### Configuring and executing a test unit in CANoe

The executable test unit can be configured and executed in **CANoe**.

For the execution in **CANoe** in the **CANoe** configuration any number of test configurations can be created. A test configuration can again contain any number of executable test units that will be executed in sequential order.

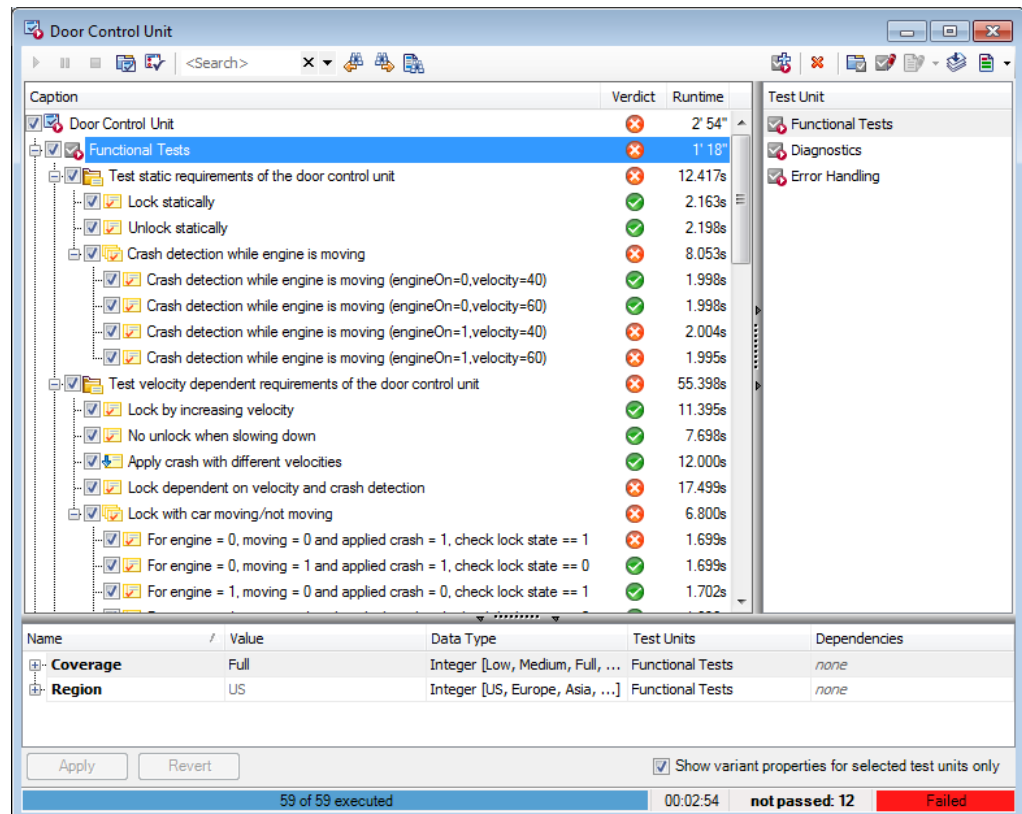


Figure 21: Test execution in CANoe

**Test Trace Window**

Details of the test run are visualized in the Test Trace Window already during the test execution.

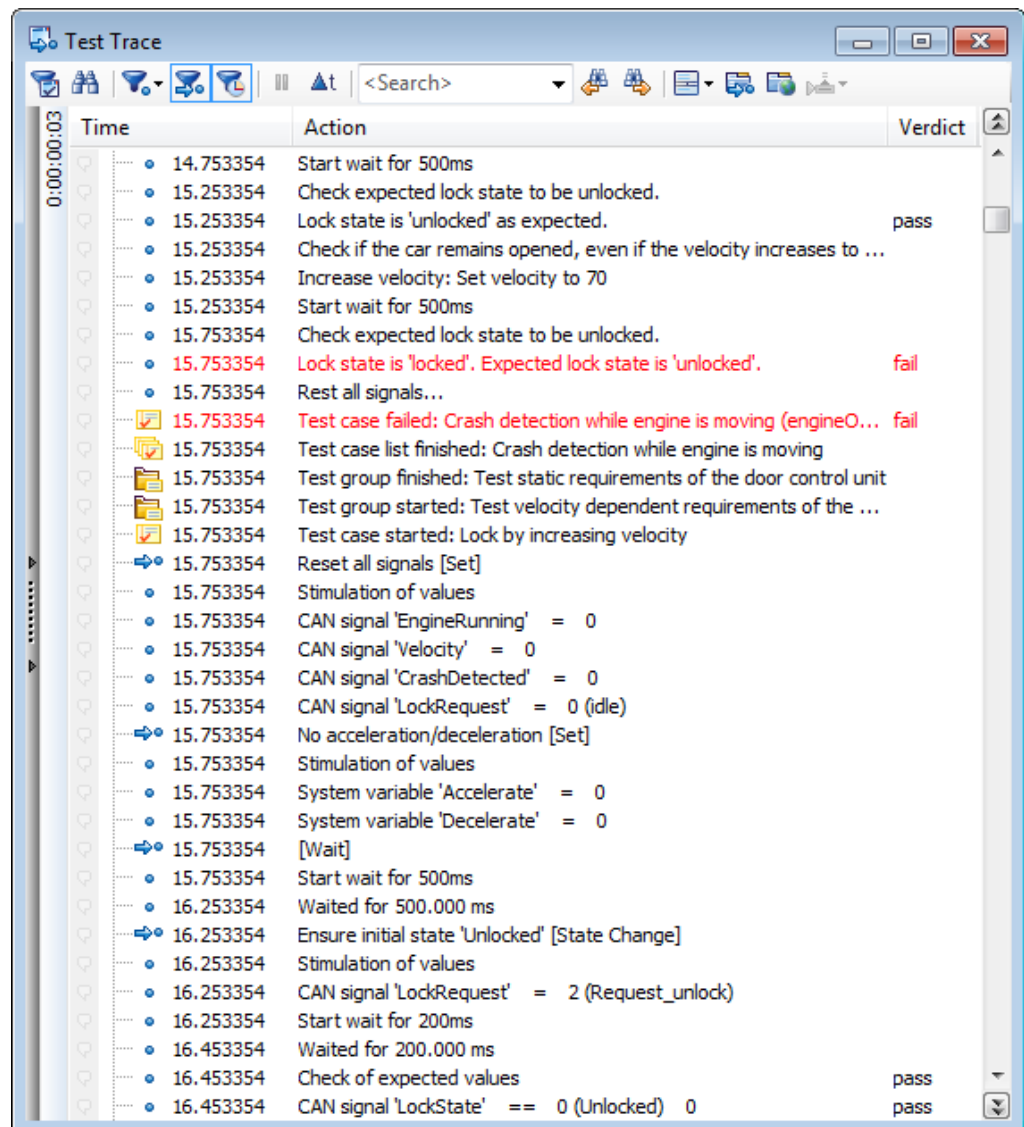


Figure 22: Test Trace Window in CANoe

**Debugging**

CANoe also supports debugging of CAPL and C# code that is part of a test unit.

## 4.3 Reporting

### CANoe Test Report Viewer

During the execution of a test unit in **CANoe** a detailed test report is created automatically. The **CANoe Test Report Viewer** supports a lot of filter and grouping functionality as well as user defined queries for a comfortable and comprehensive analysis of the test report.

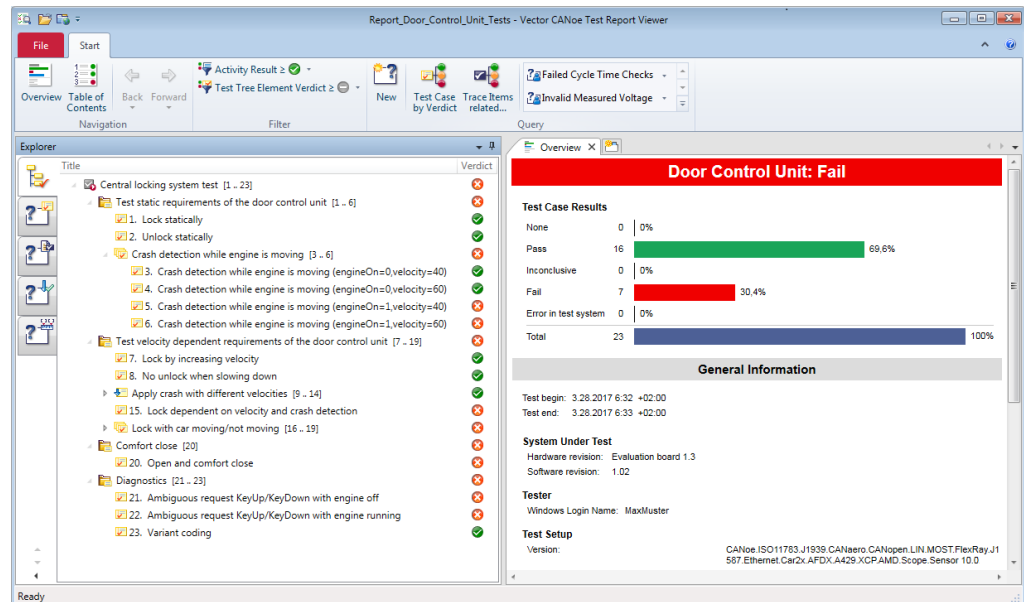


Figure 23: CANoe Test Report Viewer

## 5 Language Interaction

In this chapter you find the following information:

---

5.1 Interface Functions

page 36

---

## 5.1 Interface Functions

### Overview

Interface functions allow for functions of one language to be used in another one. An interface function can be a test case, a test sequence, a test function, or a simple function. Other than the test function the simple function is not represented as a block in the report.

### Definitions

An interface function can be defined as follows in the respective languages:

- > **C#:**  
Attribute **[Export]** on test cases, test sequences, test functions or simple functions.
- > **CAPL:**  
Keyword **export** ahead of test cases, test sequences, test functions or functions.
- > **Test Table Editor:**  
**Export** setting on test cases, test sequences or functions which are defined in the functions view.

All interface functions which are available within a test unit are displayed in an explorer.

### Explorer

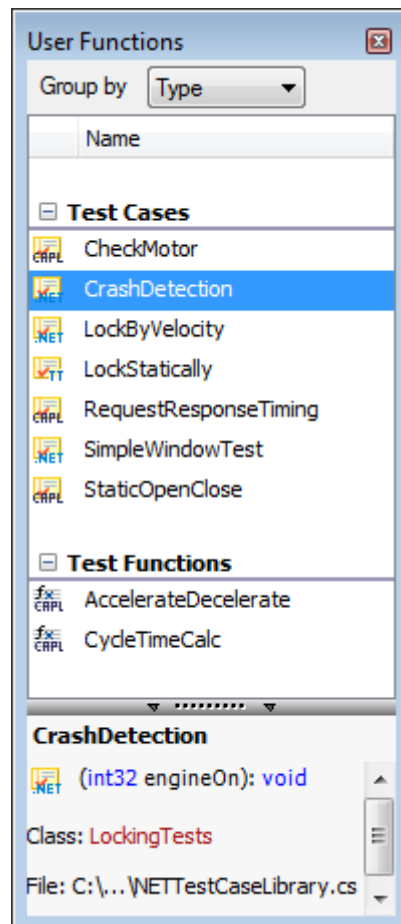


Figure 24: Explorer for interface functions

Out of the explorer, the functions can be added to the respective editors using drag & drop. Alternatively the functions can be added to the editors by the use of the text input completion feature.

**Language interaction** The following language interaction is supported:

Calling Language	Defining Language				
	CAPL	C#	Test Table	Test Sequence Diagram	State Diagram
CAPL	—	—	—	—	—
C#	•	•	—	—	—
Test Table	•	•	•	—	—
Test Sequence Diagram	•	•	•	—	—
State Diagram	•	•	•	—	—





## 6 Parameters, Curves and Variants

In this chapter you find the following information:

---

6.1	Parameters	page 40
	Concept	
	Find Test Case Data by the Classification Tree Method	
6.2	Curves	page 43
6.3	Variants	page 44

---

## 6.1 Parameters

### 6.1.1 Concept

#### Definition

The term "parameter" refers to any constant value that can be accessed within the test sequence from all implementation languages.

Examples of parameters: Configuration parameters for a control unit, test vectors, tolerances, etc.

Parameters are defined and maintained in separate files. Available parameters are displayed in the Symbol Explorer within the tab **Parameters**.

#### Kinds

There are several kinds of parameters:

##### > (Scalar) Parameter:

A scalar parameter represents exactly one constant value that can be accessed in the test sequence.

**Example:**

Name	Value
CycleTimeTolerance	50

##### > (Scalar) List Parameter:

A list parameter has 1...n values for the same variable. In the test sequence, iteration over all values of the list can be performed in order, for example, to perform a test under different temperature values.

**Example:**

Name	Value
OutsideTemperature	-40, -10, 0, 15, 30, 50

##### > Struct Parameter:

A struct parameter represents a set of associated (scalar) values. In the test sequence, the individual values of a struct can be accessed in order, for example, to apply a test vector (stimulating and expected values) to the test system.

**Example:**

Struct	Member	Value
LockingTestVector	Velocity	60
	CrashDetected	0
	Wait	500
	LockState	1

##### > Struct List Parameter:

A struct list parameter corresponds to a list of value tuples for a struct. In the test sequence, iteration over all list elements can be performed in order, for example, to apply various definitions of a test vector onto the system.

**Example:**

Struct	Member / Value			
LockingTestVectorList	Velocity	CrashDetected	Wait	LockState
	60	0	500	1
	40	0	500	0
	60	1	250	0

**Hierarchical structure** Parameter definitions can be structured hierarchically using namespaces.

## 6.1.2 Find Test Case Data by the Classification Tree Method

Classification tree method

By an integrated editor for the classification tree method test case data - in terms of test vectors - can be defined. The graphical user interface supports finding the relevant input data for a test. Automatic or manual combination of all crucial input values allows to efficiently defining the minimum number of required test vectors.

Boundary value analysis

A dedicated support of boundary values enables the targeted testing in critical value ranges of the input data.

User interface

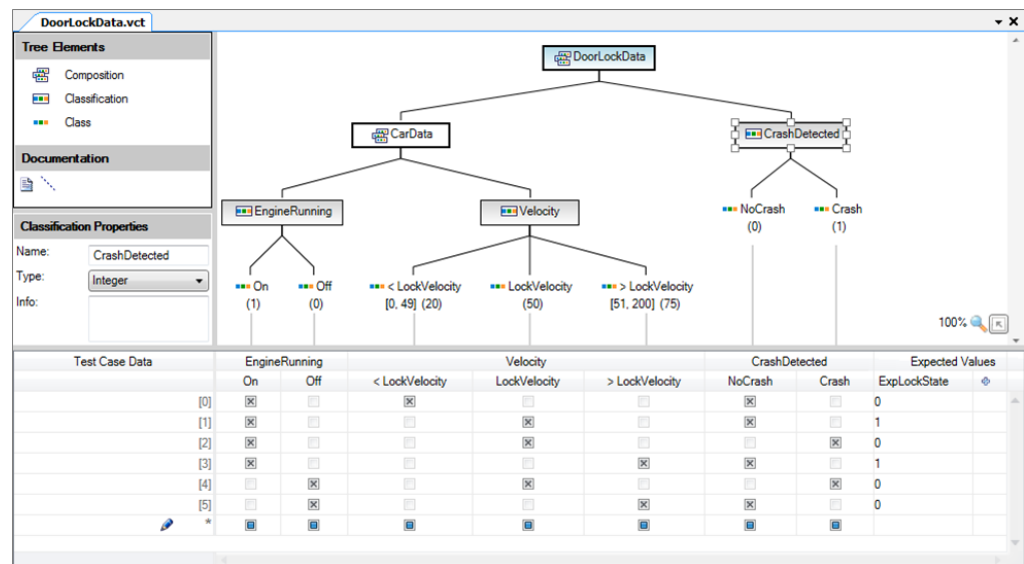


Figure 25: Definition of test vectors by the editor for the classification tree method

Parameterization of test case lists with test vectors

The test vectors can be used in implemented test cases, e.g. for the parameterization of test case lists in the Test Table Editor (see section Concepts for High Test Coverage).

Signature:	CheckAutomaticLock(int64 engineRunning, double velocity, int64 crash, int64 lockState) : void			
Parameter Values				
Struct List:	DoorLockData			
Combinatorics:	sequential			
Name:	engineRunning	velocity	crash	lockState
Type of Values:	Member of Struct List	Member of Struct List	Member of Struct List	Member of Struct List
Value Source:	DoorLockData.EngineRunning	DoorLockData.Velocity	DoorLockData.CrashDetected	DoorLockData.LockState
Values:	1	20	0	0
	1	50	0	1
	1	50	1	0
	1	75	0	1
	0	50	1	0
	0	75	0	0
Test Case Caption:	CheckAutomaticLock ( engineRunning = {engineRunning}, velocity = {velocity}, crash = {crash}, lockState = {lockState} )			
Use Property...	Trace Items, Variant Dependencies, External References			
Caption:	Check locking behavior			

Figure 26: Usage of test vectors for the parameterization of a test case list in the Test Table Editor

## 6.2 Curves

### Stimulation curves

By the use of the so-called Waveform Editor curves for the stimulation of the system under test can be defined. Predefined segment types (sinus, pulse ...) enable easy definition of e.g. voltage curves defined by ECU test standards like LV124.

Multiple curves can be synchronized easily within the same editor.

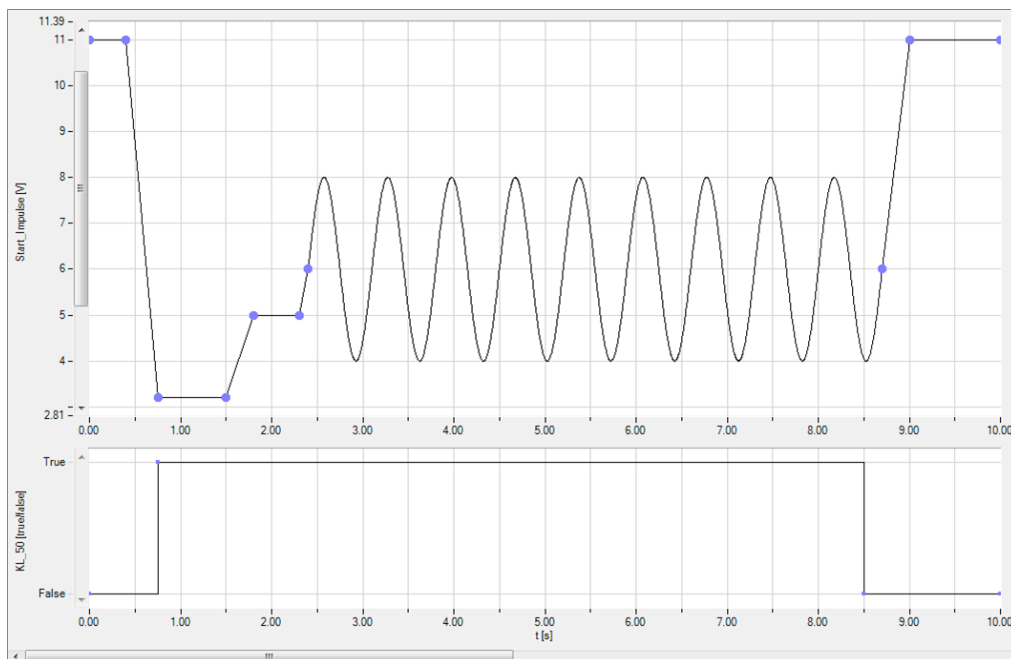
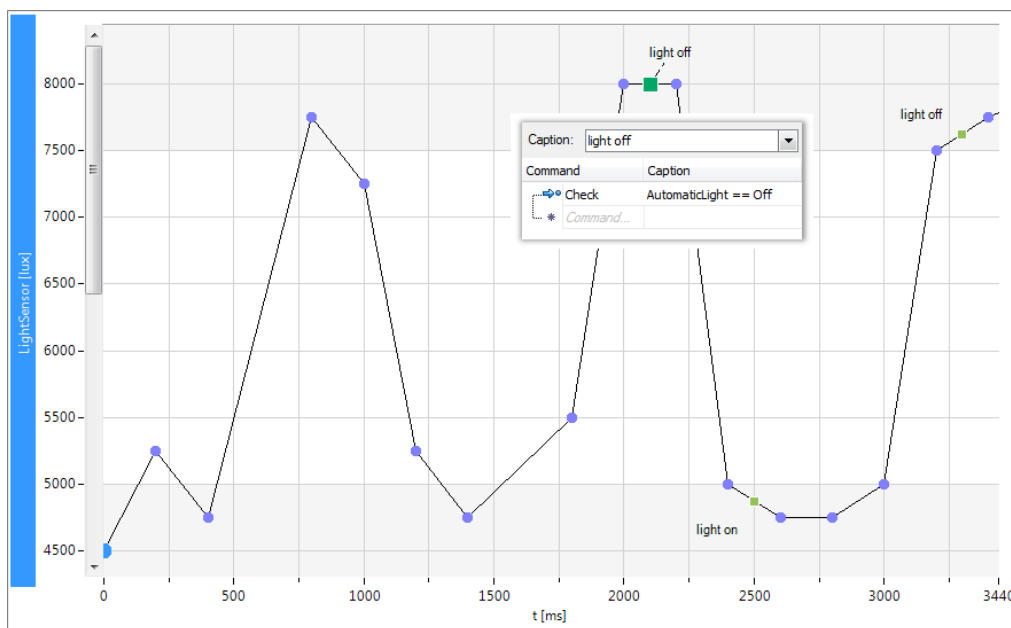


Figure 27: Definition of stimulation curves with the Waveform Editor

### Synchronized check points

By the definition of check points for a stimulation curve the reaction of the system under test triggered by the stimulation can be verified.



## 6.3 Variants

### Variant properties

ECU variants and test variants can be realized using so-called variant properties.

#### Examples:

Variant Property	Possible Values
Region	US, Europe, Asia
Coverage	full, low, medium
Model	OEM1, OEM2

### Access

Variant properties can be used for conditional test coding as well as for access to variant-dependent parameter values and for defining variant-dependent test cases or test groups.

### Access to variant property in CAPL

```
if (varprop::Region == varprop::Region::US)
{
    // ...
}
```

### Parameter value dependent on a variant property

Name	Model		
	OEM1	OEM2	
Tolerance	50	65	

Figure 28: Parameter value dependent on a variant property

### Variant-dependent test cases

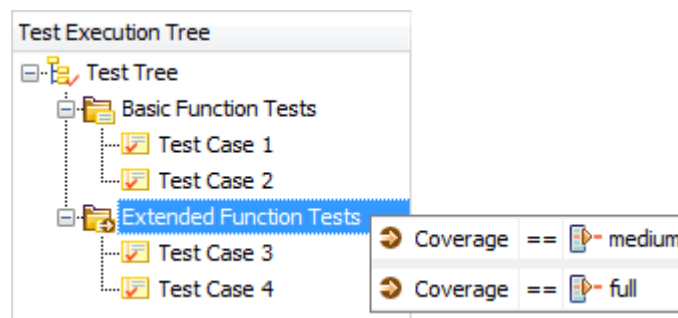


Figure 29: Variant-dependent test cases

**Transfer**

Variant properties can be easily transferred to the test code from the Symbol Explorer or using text completion.

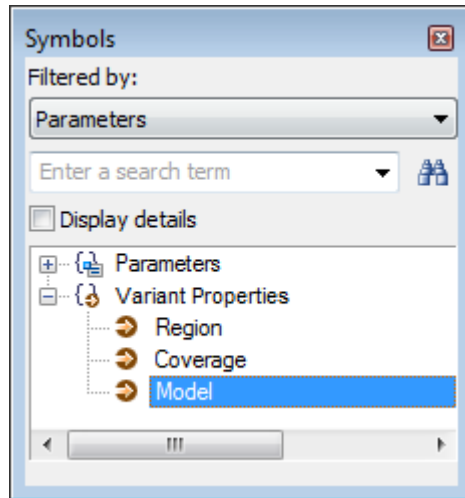
**Variant properties in the Symbol Explorer**

Figure 30: Variant properties in the Symbol Explorer

The value of a variant property is either already defined at design time in **vTESTstudio** or can be set in **CANoe** up until just before the start of the test.

Variant properties can be dependent on one other, i.e., the value of one variant property (e.g., *Region*) can determine the value of another variant property (e.g., *Model*).





## 7 Use Cases

In this chapter you find the following information:

---

7.1	Generating Two Similar Test Units for Different OEMs
-----	--

---

page 48

## 7.1 Generating Two Similar Test Units for Different OEMs

### Setup

There is a **vTESTstudio** project for the test of the door control unit. It consists of two test units – one for OEM1 and one for OEM2. The actual sequence logic is the same for both OEMs. It is implemented in a test table **Main.vtt** and is used in both test units by a file link. For information on linking files to test units, also see [Re-Use of Files](#).

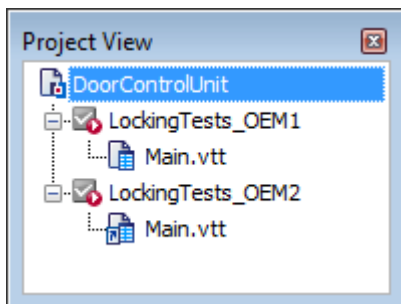


Figure 31: Using test logic in both test units

Various tolerance values are accessed during the test. These are dependent on the OEM. An **OEM** variant property is declared for this purpose with two possible values: **OEM1** and **OEM2**.

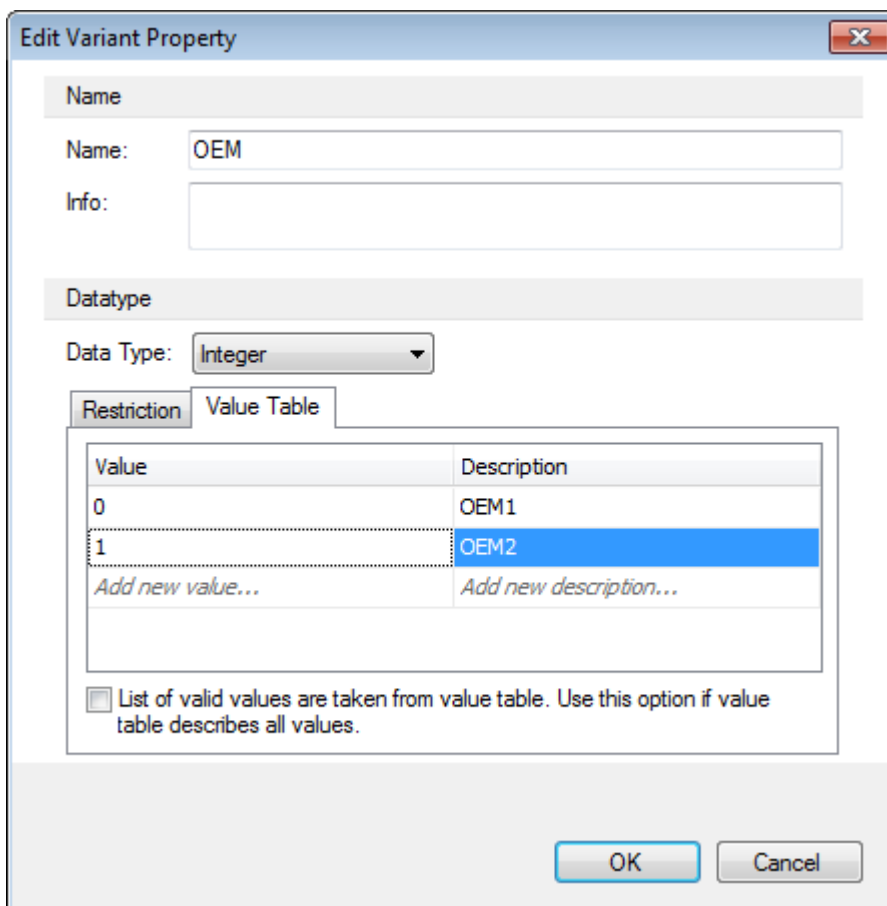


Figure 32: Definition of the variant property **OEM**

The tolerances themselves are defined in a parameter file. The specific tolerance values are defined according to the **OEM** variant property.

Name	Variant	
	OEM1	OEM2
140 MaxReactionTime	500	600
140 Tolerance1	10.2	15.1
140 Tolerance2	79.9	80.0

Figure 33: Parameters with variant-dependent values

The parameter file is also used in both test units.

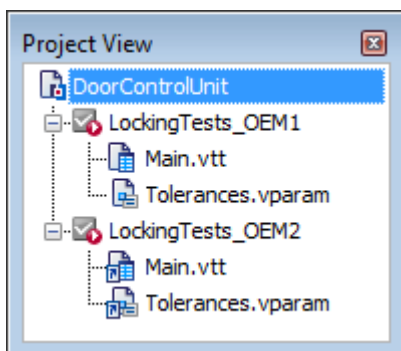


Figure 34: Using same parameter files in both test units

To ensure that the correct parameter value is used during the test execution, the user only has to set the correct value of the **OEM** variable property in each case in the configuration dialog of the test units.

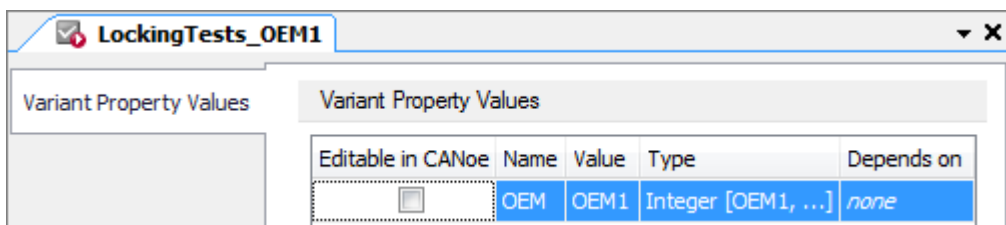


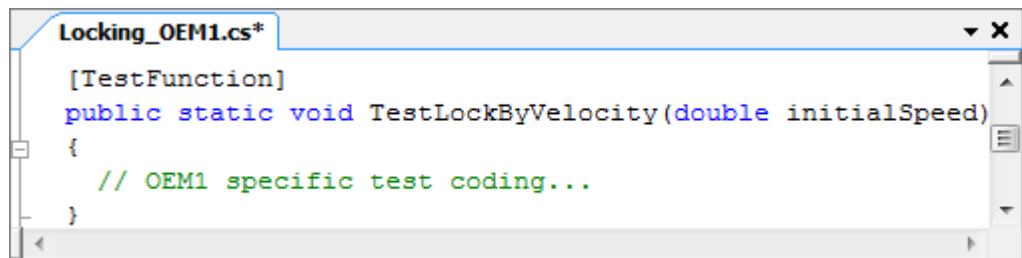
Figure 35: Defining the variant property value for the **LockingTests\_OEM1** test unit

In addition to the OEM-dependent parameter values, an OEM-dependent test function is also to be used. For this, a C# function is called in the test table:

Test Case	
	Lock by increasing velocity
Set	Stimulate input signals
Wait	MaxReactionTime ms
Check	Check expected output signals
TestLockByVelocity	Do OEM specific locking test
Wait	500 ms
Set	Reset signals

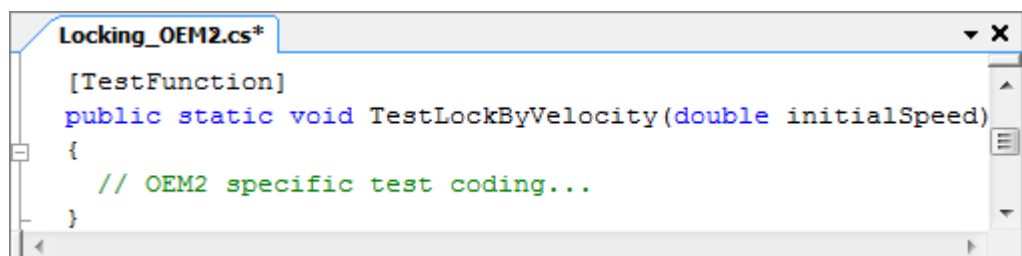
Figure 36: Calling a C# function in the test table

The implementation of the function takes place in two C# files, one of which contains the implementation for **OEM1** and the other the implementation for **OEM2**:



```
Locking_OEM1.cs*  
  
[TestFunction]  
public static void TestLockByVelocity(double initialSpeed)  
{  
    // OEM1 specific test coding...  
}
```

Figure 37: C# file with OEM1-specific implementation for **TestLockByVelocity**



```
Locking_OEM2.cs*  
  
[TestFunction]  
public static void TestLockByVelocity(double initialSpeed)  
{  
    // OEM2 specific test coding...  
}
```

Figure 38: C# file with OEM2-specific implementation for **TestLockByVelocity**

The file for OEM1 is added to the first test unit and the file for OEM 2 is added to the second test unit:

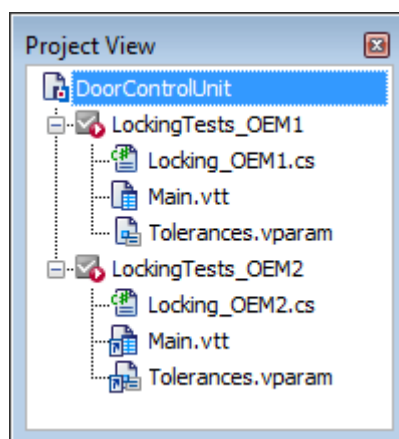


Figure 39: Using different C# files for variant-dependent implementations





## More Information

- > News
- > Products
- > Demo Software
- > Support
- > Training Classes
- > Addresses

**[www.vector.com](http://www.vector.com)**