

LABCAR-AUTOMATION 4.2.2

How to ...? Frequently asked questions – Tips & Tricks



Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract.

Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

© Copyright 2012 - 2015 ETAS GmbH, Stuttgart

The names and designations used in this document are trademarks or brands belonging to the respective owners.

Contents

1	Introduction.....	5
1.1	Conventions.....	5
1.2	Installation Paths	6
2	For Test Case Developers	7
2.1	How do I write test cases?.....	7
2.2	Offline Test Case Design.....	7
2.3	Test Case Design and Debugging within Visual Studio	11
2.4	Test Case Design without Test Management tools (Embeddable Package)	12
2.5	Automated Label Mapping	13
2.6	Intermediate close of INCA Experiment	13
2.7	Building Test Case Definition (.tcd) File	13
3	For Test Bench Configuration Responsibles	14
3.1	Configuration Wizard	14
3.1.1	Typical Use Case.....	14
3.1.2	Features of the Configuration Wizard	14
3.2	Test Bench Configurations can be fully determined by "Unit under Test"	15
3.3	The SUT Mapping Editor	16
3.4	Access to INCA with or without LABCAR-OPERATOR.....	16
3.4.1	Using INCA with LABCAR-OPERATOR	17
3.4.2	INCA Standalone.....	18
3.4.3	Necessary Installations for the use of INCA standalone	18
3.5	Flashing with INCA and PROF	20
3.5.1	Pre-requisitions.....	20
3.5.2	Preparing the test bench configuration	20
3.5.3	Lines in code	24
3.6	Parameter recording for Sync DL.....	25
3.7	Failure Simulation Load Cut off	26
3.8	Connection to dSpace Test Bench	27
4	For Testers	29
4.1	Changing the layout of the report.....	29
4.1.1	Changing report layout after creation per report	29
4.1.2	Changing report layout for all reports	29
5	Complex features	30
5.1	Offline Project generation	30
5.1.1	Typical Use Case.....	30
5.1.2	Features of the Project Generator	30
5.1.3	Feature Handling	30
5.2	Report Structure 'Abstract Section'	31
5.2.1	Feature Description	31
5.2.2	Test case development for use of the Abstract Section	31
5.2.3	Report View with Abstract Section.....	33
5.3	Improved Data logging functionality	33
5.3.1	Typical use case.....	33

5.3.2	Feature Description	34
5.3.3	Feature Handling	34
5.3.4	Test bench configuration for LCO project dataloggers	35
5.3.5	Hints	35
5.4	Maps and Curves.....	35
5.4.1	Typical use case.....	35
5.4.2	Feature Description	35
5.4.3	Feature Handling	35
5.4.4	Mapping rules	36
5.4.5	Example	36
5.5	Real Time Tests support	38
5.5.1	Typical use cases	38
5.5.2	Feature Description	38
5.5.3	Feature handling.....	39
5.5.4	Test case development for Real Time testing	40
5.5.5	Test bench configuration for Real Time testing	40
5.5.6	Hints	40
5.6	Soft-Stop Function for Test cases	41
5.6.1	Typical use cases	41
5.6.2	Feature Description	41
5.6.3	Feature Handling	41
5.6.4	Test case development for Soft Stop	41
5.6.5	Test bench configuration for Soft Stop	42
5.6.6	Hints	42
5.6.7	Difference between status shown in Test Handler and Verdict reported by test case	42
5.7	Error Manager.....	43
5.7.1	Typical Use Cases	43
5.7.2	Feature Description	43
5.7.3	Feature Handling	43
5.7.4	Test Case development with Error Manager	43
5.7.5	Test Bench Configuration for Error Manager	45
5.7.6	Hints	45
5.8	Working with a Signal Generator	45
5.8.1	Typical use cases	45
5.8.2	Feature Description	45
5.8.3	Feature Handling	46
5.8.4	Test case development for the Signal Generator	47
5.8.5	Test bench configuration for the Signal Generator	48
5.8.6	Mappings.....	50
5.8.7	Working with dSpace.....	50
5.8.8	Example with LABCAR-OPERATOR 3.2.5	51
6	General issues.....	52
6.1	Test Handler Tool Options – across different installations	52
6.2	Silent Installation	52
6.3	License Management.....	54
7	ETAS Contact Addresses	55

1 Introduction

This document helps you to

- setup your use case,
- getting answers for frequently asked questions,
- find useful hints when running in trouble,
- description of complex features.

Whenever you need some information about usage of LABCAR-AUTOMATION, this document should be your first entry point to find a solution for your issue.

This document is grouped into different sections regarding the functional roles of the automated testing process and the general issues concerning e.g. installation and terms of use.

1.1 Conventions

Formatting of dialog elements

If names of entry fields used by a dialog are used in the documentation they are written in *italic* letters.

E.g: *Folder, Device, Database*

Formatting of entries

Entries to be made in dialog fields are written in **bold** letters.

1.2 Installation Paths

All installation paths in this document refer to a **Windows 7** (64bit) system environment. The standard directories in this environment are:

Program Files:

C:\Program Files (x86)\ETAS\LABCAR-AUTOMATION 4.x

Configuration Files:

C:\ProgramData\ETAS\LABCAR-AUTOMATION 4.x

Examples and Default Data:

C:\Users\Public\Documents\ETAS\LABCAR-AUTOMATION 4.x

When working in a **Windows XP** environment, these directories are located at:

Program Files:

C:\Program Files\ETAS\LABCAR-AUTOMATION 4.x

Configuration Files:

C:\Documents and Settings\All Users\Application Data\
ETAS\LABCAR-AUTOMATION 4.x

Examples and Default Data:

C:\Documents and Settings\All Users\Documents\
ETAS\LABCAR-AUTOMATION 4.x

For a more detailed explanation please have a look into the LABCAR-AUTOMATION User's guide, chapter: 2.4.1 the "\Users\Public\Documents\ETAS" Folder.

2 For Test Case Developers

2.1 How do I write test cases?

First of all, before tests can be performed, they have to be created. There are two major possibilities to do this.

- Using the Automation Sequence Builder
 - ➔ Please have a look into the User's guide at chapter 4.1 Automation Sequence Builder
- Using a .Net® compatible Software Development language like C# or Python
 - ➔ Please read the document 'ATCL getting started.pdf'. It provides a quite comprehensive and detailed description how this works.

2.2 Offline Test Case Design

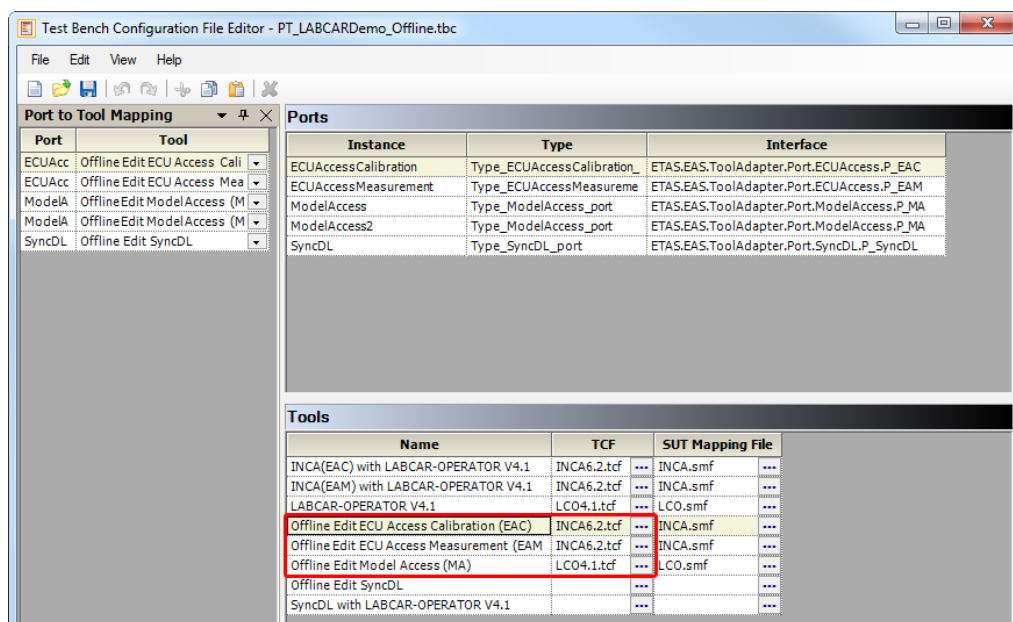
To reduce the expensive usage of test bench, there is a possibility to develop the test case code with a virtual test bench, called Offline Test Bench.

Offline Test Benches fake the behavior of a real test bench. Test cases or useful functions making use of ATCL signatures can be created, set to operation and validated without a full-fledged test system.

The behavior of the complete test system corresponds to the real one. The whole tool adapter chain, like SUT mappings, data conversion and unit conversions, is passed. Only the final API calls to the test tools are faked by the use of text files containing the expected test bench behavior or input dialogues returning the expected values.

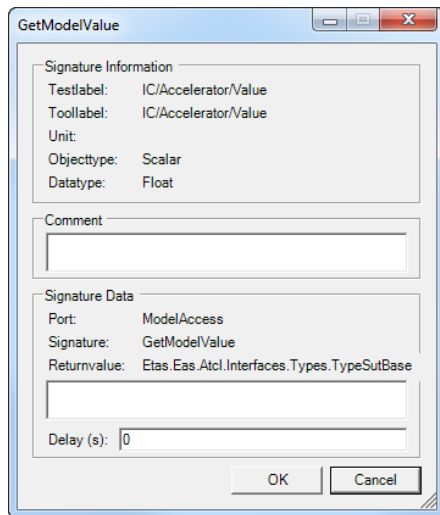
- ➔ This mechanism can be used to execute test sequences offline from the Automation Sequence Builder as well as test cases from the Test Handler

An "Offline Test Bench" is specified in the TBC File Editor as any other Test Bench by selecting the "Offline Tools". These tools replace the real-world tools.



There are two ways how expected values can be provided.

- For single values an input at the testers console via dialogue is the most appropriate method.
- More frequently expected return values are preferably described in a text file called 'offline file'. In this file each line represents one record for one return value.



```

1 "P_MA" GetModelValue "Accelerator" 0.0 0:
1 "P_MA" GetDataLoggerState "-" PortConfigured 0:
1 "P_MA" GetLoggedSignals "-" D:\LCA-Tests\2_TBC\TC_AllSignaturesOffline\Testbench Configuration\log_filename.dat 0:
1 "P_MA" GetSignalGeneratorState "-" PortConfigured 0:
1 "P_MA" GetSelectedElements "label" [0:1.0] 0:
1 "P_MA" GetArrayLength "-" 23 0:
1 "P_EAC" ChecksumIsValid "-" true 0:
1 //1 "P_EAC" GetValue "MyLabel" 5 0:
1 "P_EAM" GetValue "Accelerator" 0.0 0:
1 "P_EAM" GetDataLoggerState "-" PortConfigured 0:
1 "P_EAM" GetSelectedElements "label" [0:1.0] 0:
1 "P_EAM" GetArrayLength "-" 23 0:
1 "P_EAM" GetSelectedSignals "-" [A:B:C] 0:
1 "P_SyncDL" GetLoggedSignals "-" D:\LCA-Tests\2_TBC\TC_AllSignaturesOffline\Testbench Configuration\datalogger.dat 0:
1 "P_EAM" GetLoggedSignals "-" D:\LCA-Tests\2_TBC\TC_AllSignaturesOffline\Testbench Configuration\log_filename.dat 0:
    
```

Example with calls for all offline signatures

Offline file design

- Columns per row (separated by tabulators) are:
 - Repetition: Number of times the line is used
 - Port: Port on which the signature is called
 - Signature: The signature which needs a return value
 - Label: The label for which a value shall be fetched
 - Return Value: The value/values, data to be returned
 - Return Delay: The time it shall take at least to return
- Commenting is allowed („//“),
- Inclusion of other offline files is allowed („#include“)
- With { } it is possible to group rows. The number before the curly bracket is the repetition of this group

Possible values in offline file and the dialogue

- Possible values for all signatures retrieving states:

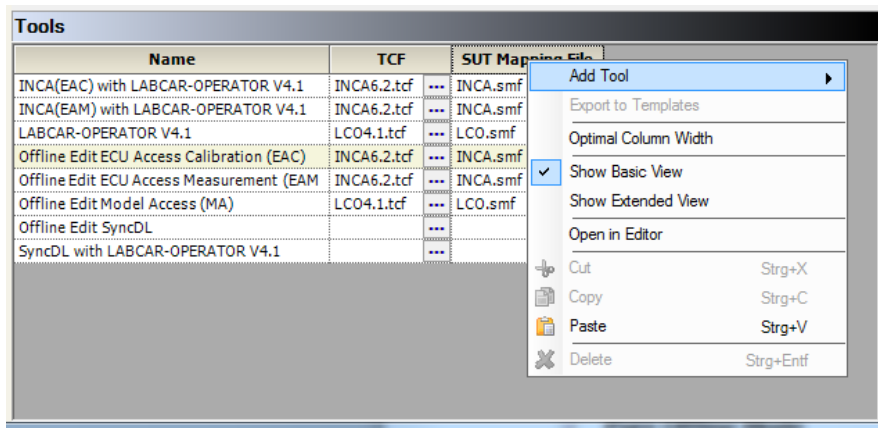
PortAny
PortLoading
PortError
PortUndefined
PortStarting
PortCompleted
PortCreated
PortPaused
PortToolConfigured
PortConfigured
PortRecording
PortLLConfigured
PortRunning
PortStopped
PortHLConfigured
PortClosed
PortReset
PortListening
PortStateUnknown

- The values for signatures retrieving values as array have to be given in the following format:
[0;1.0]
[A;B;C]
- The GetLoggedSignals signature expects a path to the datalogger ".dat" file containing the values.

Test Bench configuration for Offline tests

The offline file is specified additionally to the offline tool in the TBC Editor for each tool. The file name has to be entered in the „Adapter Config File“ - Column by selecting the Extended View via right mouse click.

One file may store the return behavior of many tools, thus the same file can be referenced for different tools.



Creation and usage of the offline files realized in different modes:

Pure Offline Mode

Pure offline tool adapters will replay the offline file for each signature. In case a mismatch of file and test case occurs, it creates an error like a real test bench error.

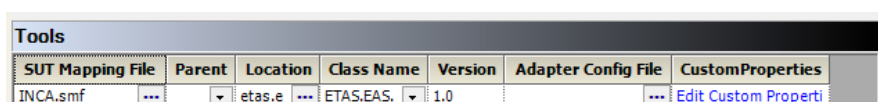
Offline/Edit Mode

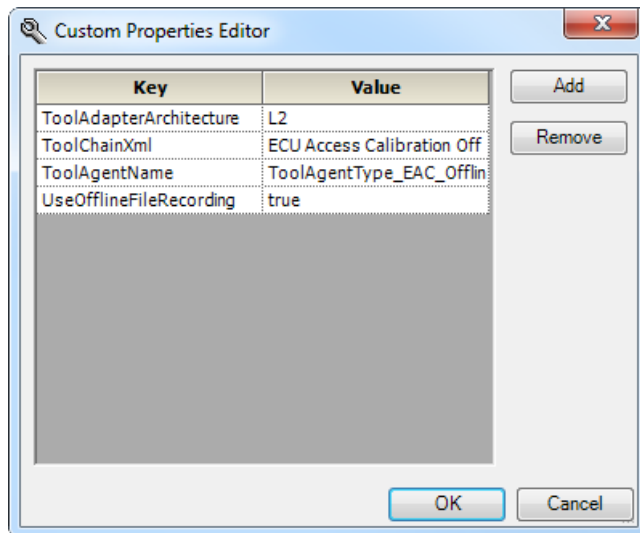
This mode is used to create/modify new or existing offline files using "Offline/Edit" tool adapter.

If a mismatch between called signature and file occurs, the user specifies the desired behavior in a dialog and the file is extended.

Offline Recording

The third mode is for the creation of an offline file. With this mode the necessary data are recorded via an online test run. The recording has to be switched 'on' for the online tool in the custom properties in the TBC file editor.





Replay of such a created offline file can be triggered via pure offline mode.

“Offline”, “Offline/Edit” and “Offline Recording” – Tool adapter are available with following ports:

Model Access

GetLoggedSignals, GetValue, GetSelectedElements, GetSelectedElementsLength, GetSignalGeneratorState, GetDataLoggerState

ECU Access Measurement

The offline feature of this port is realized for LCO 4.1 and higher only
GetLoggedSignals, GetValue, GetSelectedElements, GetSelectedElementsLength, GetDataLoggerState

“ECU Access Calibration”

The offline feature of this port is realized for LCO 4.1 and higher only
ChecksumIsValid, GetValue

“Offline” and “Offline/Edit” – Tool adapter are available with following port as well:

“Synchronized Logging”

P_SyncDLGetLoggedSignals, getState

All “Set*” – signatures are assumed to be successful on an Offline Test Bench

2.3 Test Case Design and Debugging within Visual Studio

Starting and debugging Test Cases from Visual Studio („F5”) is fully supported.

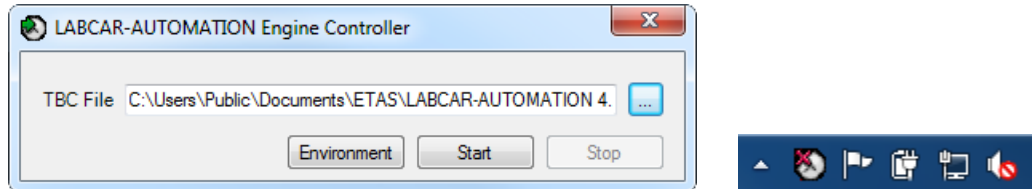
It includes specification of Test Bench Configurations and Report File location.

For this purpose the LABCAR-AUTOMATION Engine Controller is running as Windows service to which the test case developer connects.

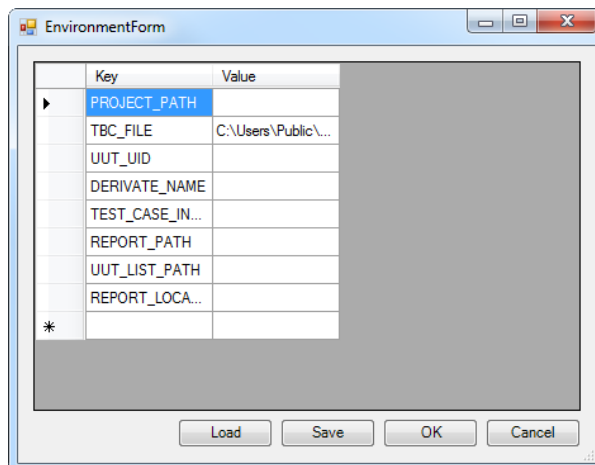
To start the LABCAR-AUTOMATION Engine Controller select via Start Menu:

Start → Programs → ETAS → LABCAR-AUTOMATION 4.x → Test Design (ATCL) → LABCAR-AUTOMATION Engine Controller

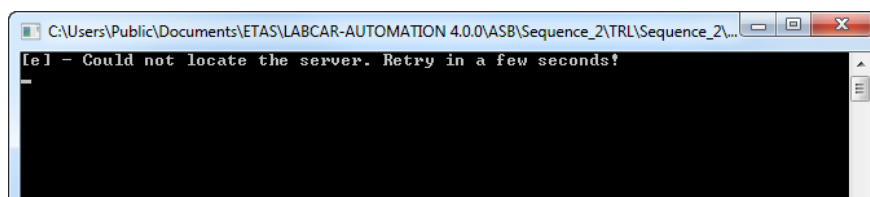
The LABCAR-AUTOMATION Engine Controller places an icon in the Windows® task bar. Via this icon the user has to specify by double click the necessary Test Bench Configuration (TBC) which LABCAR-AUTOMATION shall use for the test. Usually this is an offline test bench configuration as the developer might not have all the appropriate tools installed.



To search out, which values are set in the selected TBC click the button 'Environment'. As soon as a TBC file is selected, start the LABCAR-AUTOMATION Engine Controller with the appropriate button. The TBC can be only switched if the LABCAR-AUTOMATION Engine Controller is stopped.



If the LABCAR-AUTOMATION Engine Controller was not started you'll get a Windows® generated error like shown below as soon as you start your test case:



2.4 Test Case Design without Test Management tools (Embeddable Package)

- The Test Release Procedure is simplified. The TCD Generator is capable to run in batch mode for automated creation of Test Case Library files.
- Microsoft® .NET Framework Version 2.0 is fully integrated.
- With the improvements in the ATCL "ParameterManager" Interface the fully parameterized Default Parameter Sets can be created directly in test case.

2.5 Automated Label Mapping

When during execution a test label is detected which was not mapped in one of the SuT mapping files, relevant for the used tool, it will be mapped automatically to the same label of the given a2l – File. Label name, type and ranges are taken over from a2l.

If a test label was mapped actively in the mapping file, this mapping is the master. With this the original functionality of your existing test cases is guaranteed.

2.6 Intermediate close of INCA Experiment

In case of long-lasting test cases it might be expedient to close the INCA experiment interim. This ensures stability of such longer tests. Reopen of the Experiment is possible with the EAM port signatures StopCommunication and StartCommunication. The optional parameter then contains the former closed experiment.

These methods change the state of the port. After calling StopCommunication the state is "PortToolConfigured" and after StartCommunication the state is "PortConfigured".

2.7 Building Test Case Definition (.tcd) File

The test case definition file is necessary for the usage of test cases (e.g. .exe or .py) inside of LABCAR-AUTOMATION projects. It contains the link to all structuring files as there are:

- Test Parameter File (.tpa)
- Test hierarchy definition (.thd)
- Test architecture definition (.tad)

As well additional used dynamic link libraries can be included into the .tcd file.

The .tpa, .thd and .tad files are created with a first execution of the test case .exe file. This can be done outside of any tool (e.g. the Test Handler), the executable can be started by double mouse click directly. The files are created automatically in the same folder like the executable is located.

Please be aware that the test case executable might use report functions as well. In case a Test Handler run is active at the same time, these report functions might lead to an overwriting of the report, which is actually created by the Test handler test run.

Please do not run a test case executable in parallel with test handler test runs!

With the TCD Editor you have a graphical user interface to collect the definition files for your test case and build the .tcd file or you can use the application TCDCMD.exe, which is command line tool generating the TCD file without showing a wizard.

For more detailed information, please refer to the ATCL getting started.pdf, chapter 4.4.1 'Test Case Definition File' and ff.

3 For Test Bench Configuration Responsibilities

3.1 Configuration Wizard

3.1.1 Typical Use Case

LABCAR-AUTOMATION is a well structured and quite comprehensive tool suite. Many variations are possible and supported with this collection. However, a high flexibility requires a certain effort to configure the tool suite, so that it fits to the own requirements.

Configuring LABCAR-AUTOMATION is a little bit sophisticated. Especially when a more or less static composition of the used test bench is used there has to be a fast and intuitive method to achieve a complete and correct configuration.

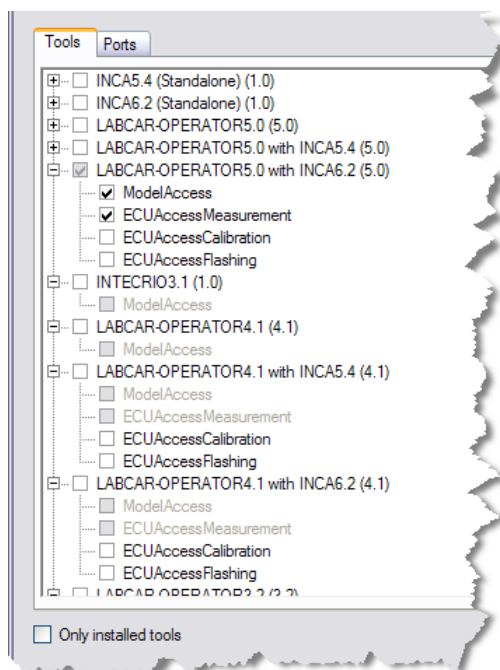
3.1.2 Features of the Configuration Wizard

A guide through all necessary actions before a LABCAR-AUTOMATION test case runs at the PC is the utmost concern of this wizard.

With the Configuration Wizard

- a complete test bench configuration is created
- a default test case is created for further editing
- a default test project is created, using the default test case and the created test bench.

Basis for the test bench configuration which is treated by the Configuration Wizard are at first the installed tools. Depending on the LABCAR-AUTOMATION package which is licensed the collection of tools to be configured is extended to all tools which are supported by LABCAR-AUTOMATION.

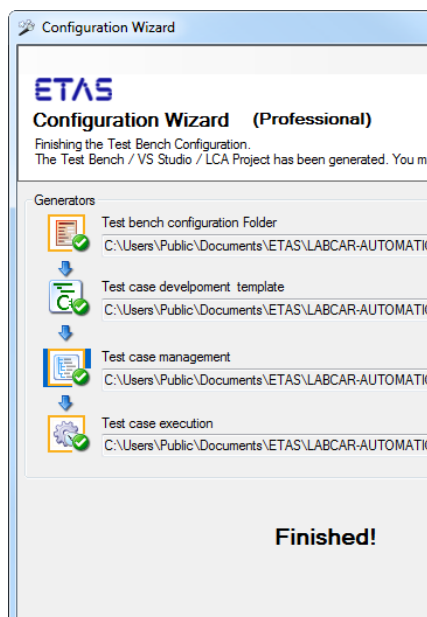


Most of the entries are preconfigured with either example files or folders or with the default entries of the appropriate tool. The Configuration Wizard moreover supports for the other entries with several search functions. All possible entries are filtered according the valid extension. In case of configuration of the INCA connections it looks for the necessary entries in the given INCA database.

After selecting all tools and providing all necessary inputs, the test bench configuration is created.

The following step provides default test case and test project using exactly this test bench. In the code of the test case all the selected tools of the test bench are available exemplarily to prepare the test case for using the tool's ports and signatures. The Test Manager, if started from Configuration Wizard by a button push, uses exactly this default test case.

Starting the Test Handler out of the Configuration Wizard it is preconfigured with the created test bench as well.

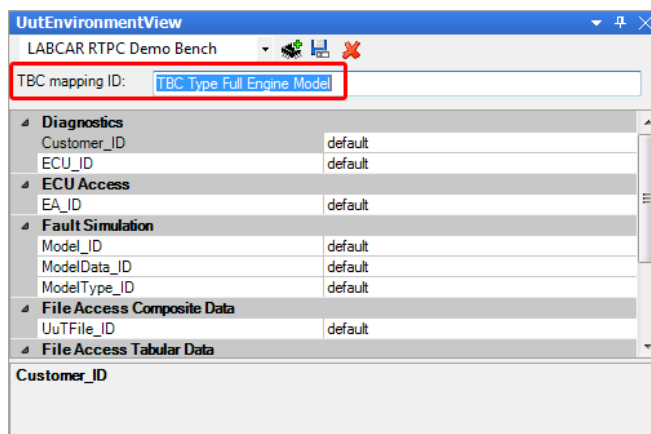
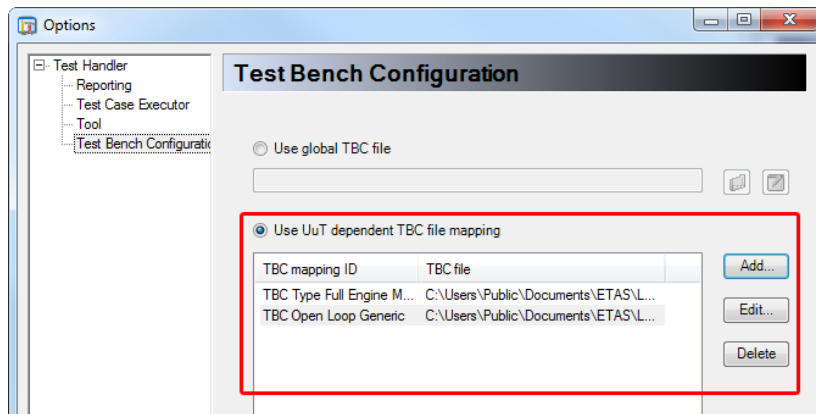


Thus, as soon as the Configuration Wizard finishes his work the customer is able to run his first test project at an operable test bench.

The complete detailed instruction manual is placed in the user's guide at chapter '4.5.1 The Configuration Wizard.'

3.2 Test Bench Configurations can be fully determined by "Unit under Test"

Users can choose to specify the Test Bench configuration depending on the UuT selected for testing:



A "TBC mapping ID" can be specified in the Environment of each UuT.

If activated, the Test Handler maps the "TBC ID" to a specific Test Bench Configuration used for this test.

The API of the UuT Server is extended to allow automated setting of the "TBC ID"

3.3 The SUT Mapping Editor

The SUT Mapping Editor is used by both, LABCAR-OPERATOR 5.x and LABCAR-AUTOMATION V3.3 ff., and is able to treat both file formats, the LABCAR-OPERATOR text file and the LABCAR-AUTOMATION format as well.

3.4 Access to INCA with or without LABCAR-OPERATOR

INCA can be addressed from within a test case with or without using LABCAR-OPERATOR. Using both, INCA and LABCAR-OPERATOR, you can use additional functions, e.g. the synchronous datalogging features, which synchronizes the time stamps of both loggings (model and ECU).

The usage without LABCAR-OPERATOR is called 'INCA standalone'.

The following examples show the different configuration entries and descriptions. As this accesses only the ECU, a synchronization of the time stamps is not necessary and not possible.

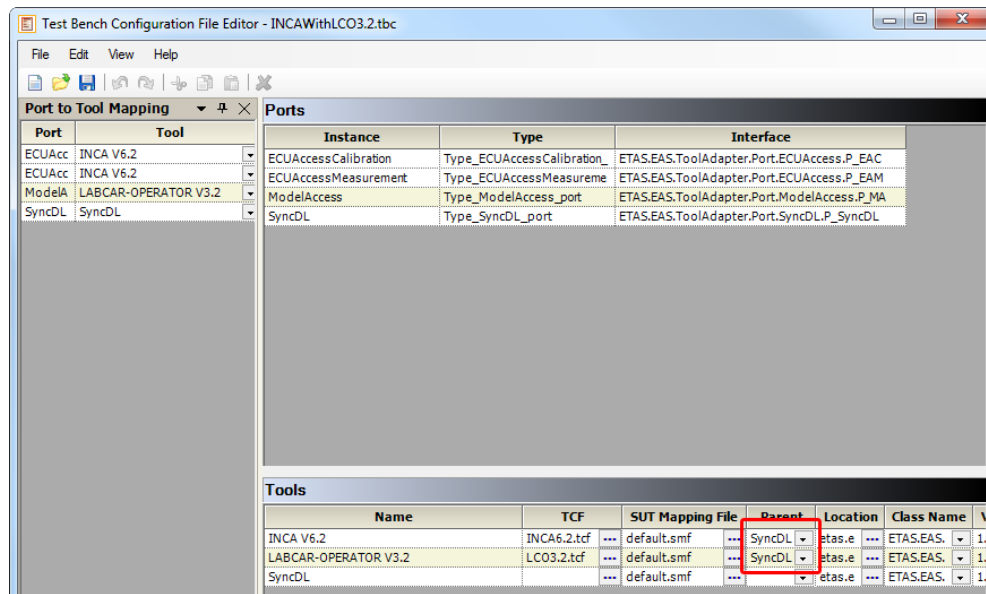
3.4.1 Using INCA with LABCAR-OPERATOR

When using INCA together with LABCAR-OPERATOR you have to look first to your LABCAR-OPERATOR version. The handling until LABCAR-OPERATOR V3.2.5 is different to the newer versions.

If both, INCA and LABCAR-OPERATOR, are installed and your Test Case is using the **ModelAccess** Port to LABCAR-OPERATOR and the **ECUAccess** Port to INCA, the configurations are like shown below.

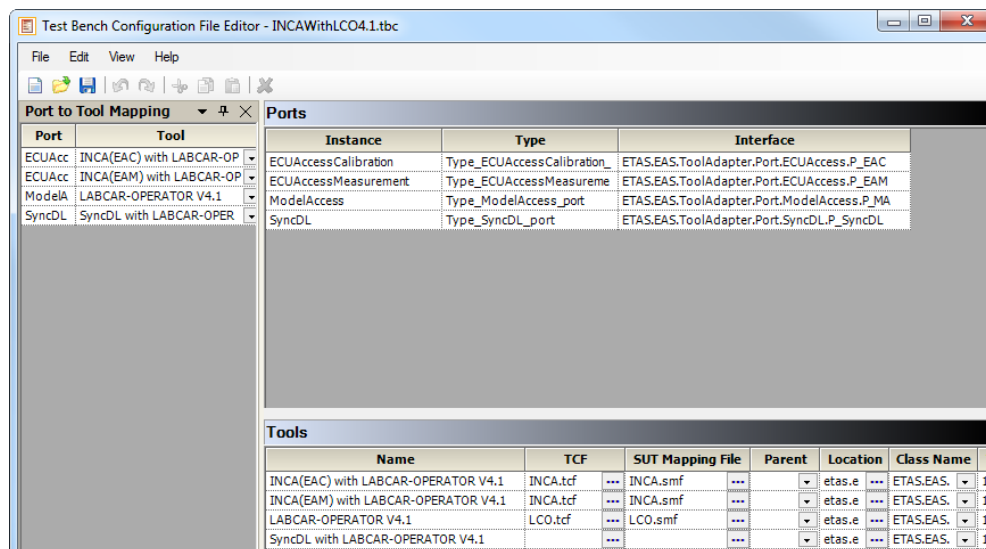
- **LCO3.2.5** and **INCA5.4** or **INCA6.2** are installed

Here you see a possible Test Bench:



- If you like to use the synchronous datalogging feature, then for **INCA** and **LCO** Tool in the parent column the **parent** port **P_SyncDL** have to be selected.
- The name of the ports (e.g. P_EAC and P_MA) have to fit to your test case names.

- **LCO4.1** (or higher) and **INCA5.4** or **INCA6.2** are installed

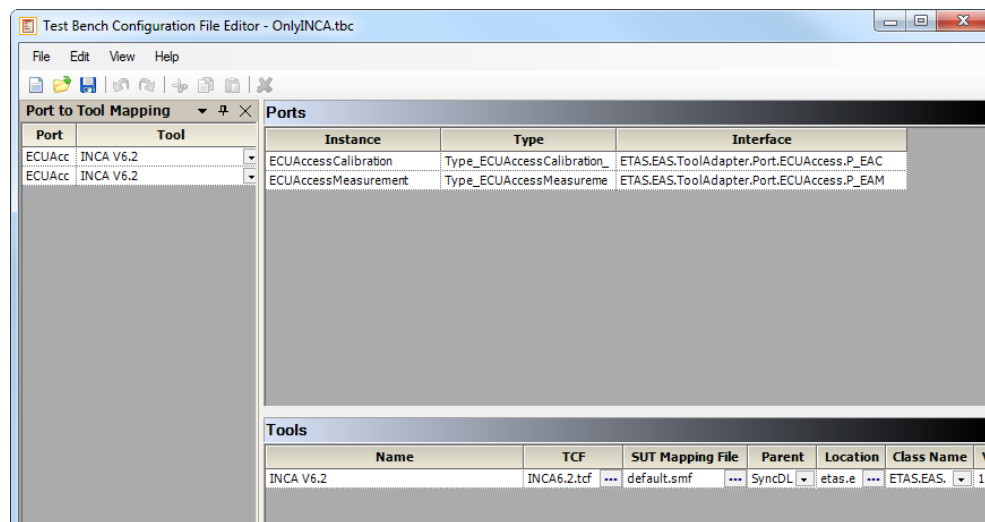


- **ModelAccess** port to LABCAR-OPERATOR has to be configured **before** **ECUAccess** port to INCA is configured inside the test case.
- If you use a LABCAR-OPERATOR version which is higher than V4.1 please adapt the version number of the example entries above accordingly.

3.4.2 INCA Standalone

Using INCA without LABCAR-OPERATOR is called 'INCA Standalone' handling, as no LABCAR-OPERATOR has to be available installed at your PC.

Test Case only using the **ECUAccess** Port (INCA), here a possible Test Bench.



- As only one signal source is available a synchronous datalogging is not provided here.
- In case LABCAR-OPERATOR V3.2.5 is installed as well, a LABCAR-OPERATOR window pops up and will be closed immediately. This does not harm the execution w/o the LCO.
- Even if LABCAR-OPERATOR V4.1 is installed at your PC as well, do NOT use the Tool template 'INCA(EAx) with LABCAR-OPERATOR' Vx.x, as this would lead to an error.

3.4.3 Necessary Installations for the use of INCA standalone

Both, INCA and LABCAR-OPERATOR, are installed

- INCA5.4/INCA6.2. and LCO5.0 ff are installed
 - [Install INCA Standalone from the CD.](#)
- INCA5.4/INCA6.2. and LCO4.1 are installed
 1. [Install INCA Standalone from the CD.](#)
 2. Unregister the old API. Default Installation Path: "C:\Program Files (x86)\ETAS\LABCAR-OPERATOR4.1\LABCAR-API" (regsvr32 /u /s "<LCO4.1Installation PATH>\LCO3API.dll")

3. You will no longer be able to use synchronous data logging, though you still can log both signal sources. But you have to treat them as individual, separate data source.
- INCA5.4/INCA6.2. and LCO 3.2.x are installed on your PC and you want to measure with INCA without LABCAR-OPERATOR:
 - No further Installation is necessary.

Use of INCA5.4/INCA6.2 without installed LABCAR-OPERATOR application

1. Install INCA Standalone from the CD.
2. Copy all dlls from installation CD folder:
 \Data\ThirdParty_AVC++7SP1.Redistributables to your ETAS directory of shared components: C:\ETAS\LABCAR-CCI-Standalone3.2\System32

Use of INCA7.0/INCA7.1 without installed LABCAR-OPERATOR application

1. Install INCA Standalone from the CD.
 (\Data\INCA Standalone\INCAXX\INCAAddOn_XML4LabCar.exe) depends on the INCA Version.

3.5 Standalone Diagnostic with INCA 7.1

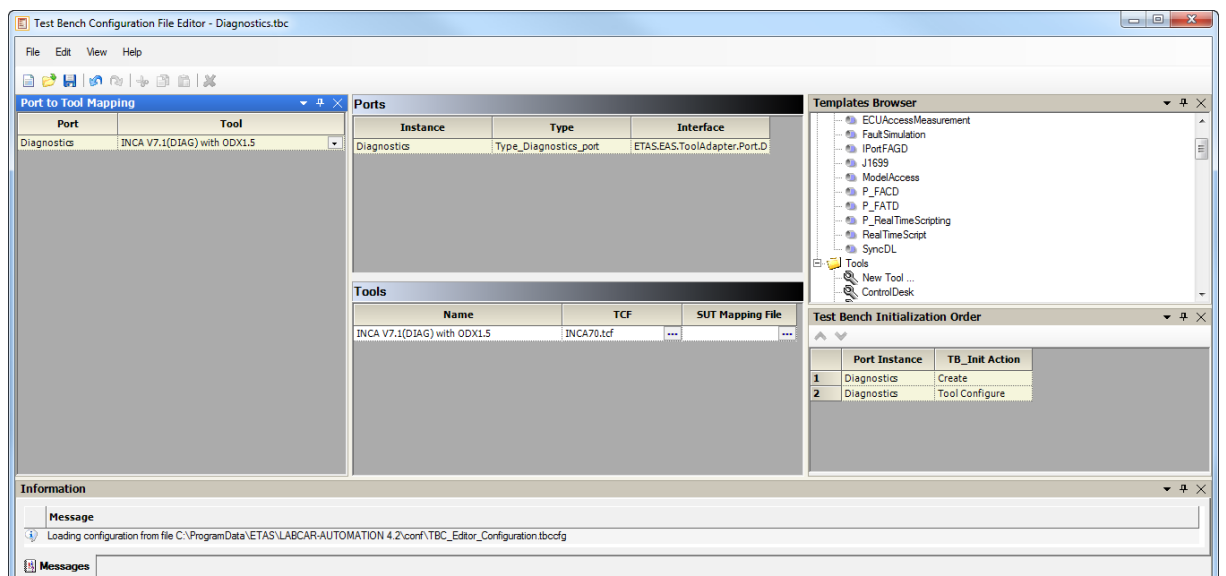
3.5.1 Pre-requisitions

A working installation of INCA and ODX-Link must be available on the computer.

3.5.2 Preparing the test bench configuration

Test bench configuration file

In your test bench configuration file you will need to define an instance of a Diagnostic port and connect this Port to the Tool *'INCA V7.1(DIAG) with ODX-LINK'*



Hint:

It is not possible to use the tool 'ODX (Diagnostics) with LABCAR-OPERATOR V5.x.'. See next chapter.

3.6 Standalone Diagnostic with LABCAR-OPERATOR

In the older LABCAR-AUTOMATION version it was possible to use the tool 'ODX (Diagnostics) with LABCAR-OPERATOR V5.x.' also for a standalone Diagnostic. That means that we are using the Diagnostic Port without a Model Access from LABCAR-OPERATOR. This functionality will be removed in one of the next version. If you still need this feature modify one/all of ODX files:

```
%ProgramData%\ETAS\LABCAR-AUTOMATION
4.2\conf\TBC\ETAS\Tools\ODX_DiagnosticsPort_EE[xx].xml
```

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <objects xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:vs="http://schemas.microsoft.com/Visual-Studio-Intellisense"
3     xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.springframework.net" xsi:schemaLocation="http://www.springframework.net C:\spring-objects.xsd">
4 <object id="ODX_DiagnosticsPort_EE35" type="ETAS.LCA.TA.ODX14.DiagnosticsPort, ETAS.LCA.TA.EEEnvironment">
5 <property name="EEEnvironment" ref="ODX_EE35"/>
6 <!-- Please use the INCA Standalone Tooladaptex instead! Alternative you can set this flag to true to have the old behaviour of the Tooladaptex.-->
7 <property name="UseDirectIncaDiagnostics" value="false"/>
8 <property name="SIPAgents">
9 <list element-type="ETAS.LCA.SAR.Core.Interfaces.Agents.I
10 <!--<ref object="AnalyseAgent" /-->
11 <ref object="LoggingAgent" />
12 <ref object="StateAgent" />
13 <ref object="DIAGToolConfigurationAgent" />
14 <ref object="SutMappingAgent" />
15 <ref object="ParameterAgent" />
16 </list>
17 </property>
18 </object>
19 <object id="ODX_EE35" type="ETAS.LCA.EnvironmentExperiment35.APIWrapper.EE_Environment, ETAS.LCA.EnvironmentExperiment35.APIWrapper" singleton="true">
20 </object>
21 </objects>

```

3.7 Flashing with INCA and PROF

3.7.1 Pre-requisites

A working installation of INCA and the PROF Tool must be available on your personal computer. In addition all necessary accessories to flash the ECU must be available, i.e. hex or s19 together with a2l-file, an INCA device to connect to and flash the ECU, like ES59x, and an INCA database providing a fitting workspace. For the PROF-Tool a configuration is necessary.

Hint:

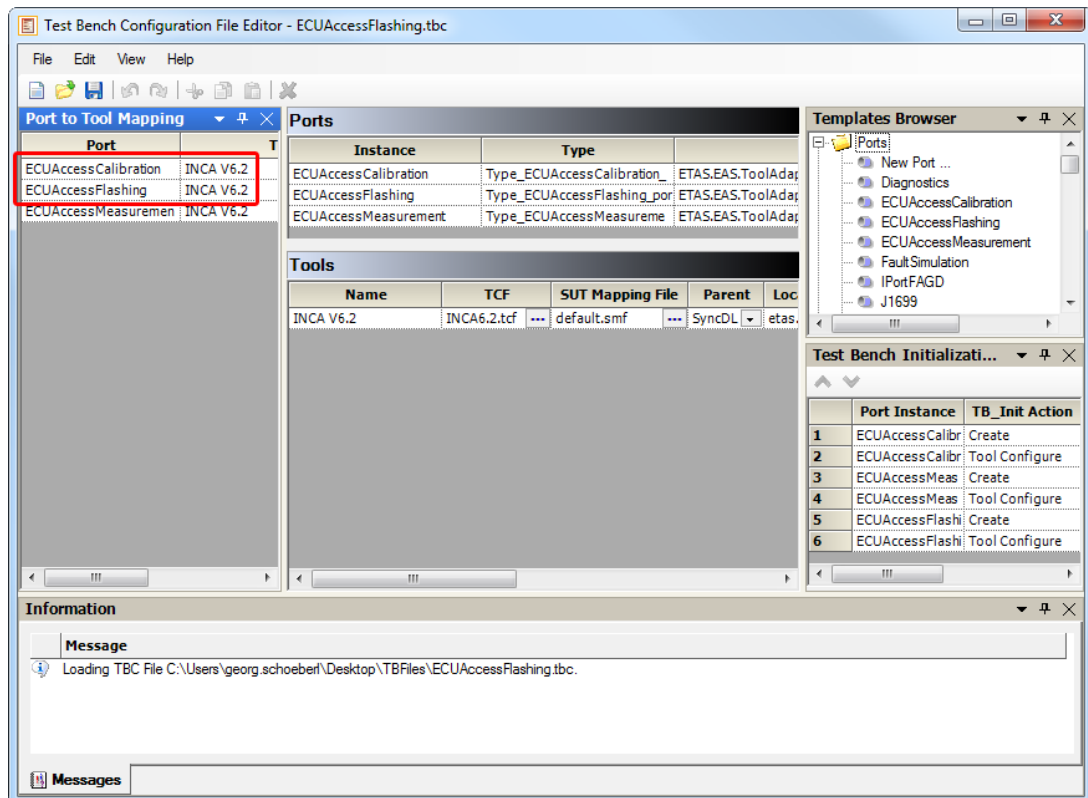
Make sure the ECU can be flashed by hand with the INCA database, workspace and PROF-Configuration you are going to use for your automated flashing.

3.7.2 Preparing the test bench configuration

Test bench configuration file

In your test bench configuration file you will need to define an instance of an ECU Access Flash port.

Make sure you choose the right INCA version, which is working with your PROF version smoothly.



Tool configuration file

In your tool configuration file you have to give the same information you provide for any other ECU access. As there are (see Figure 1)

- the INCA Database to open
- Used Measurement Device
- Used Experiment
- Folder in database containing the used experiment
- Used Workspace
- Folder in database containing the used workspace
- SuT Mapping File to be used

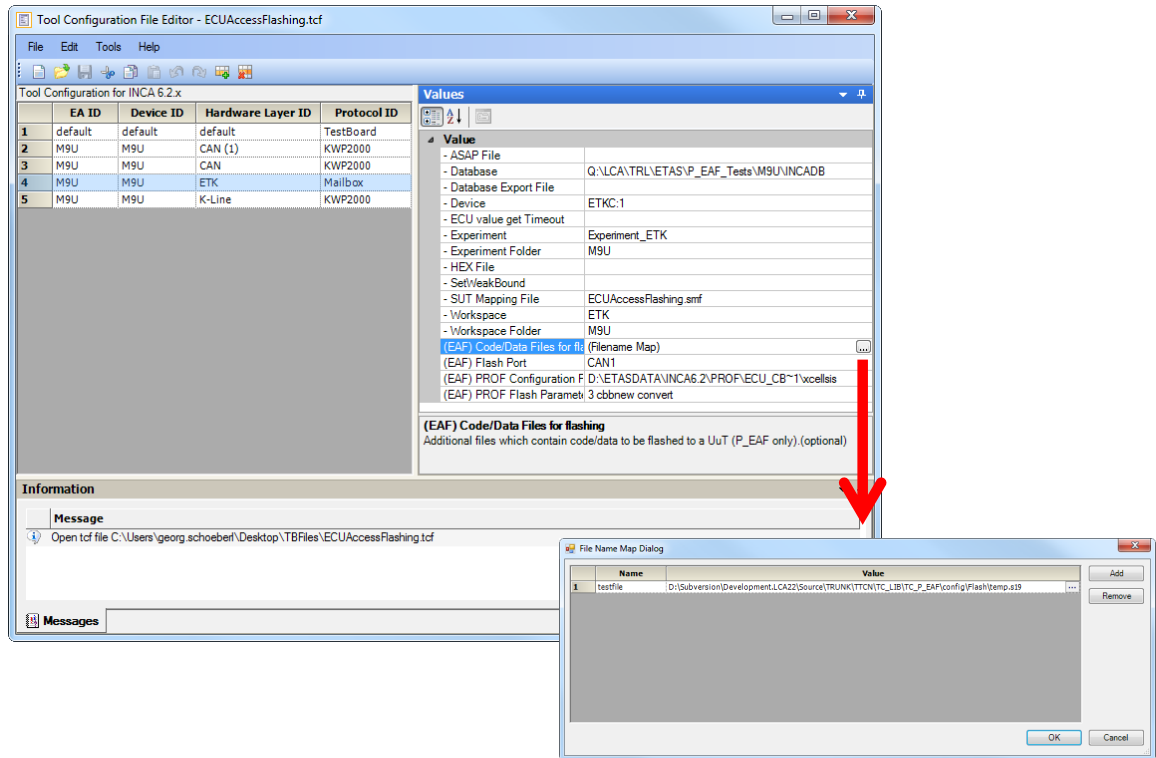
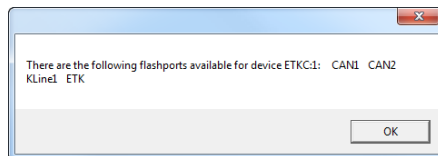


Fig.1: Tool configuration file prepared for the EAF port

In addition you have to give the information necessary for flashing

- file you want to use for flashing is given in (EAF) Code/Data Files for flashing (see Figure 1 and also "File Name Map Dialog ")
- The Flash Port , i.e. the port used by INCA to flash the ECU. You can find this information
 - in the INCA hardware configuration in your flash port option. (See Figure 2). But be aware that depending on the device chosen this information might be misleading.
 - by using the visual basic script GetFlashPortNames.vbs. It allows you to determine the flash ports available in the device used by your experiment.



You then have to choose the one used by your PROF tool.

- If you are using ODX have a look for your physical link device. You can find it in the vehicle information table (VIT). You can use ODX-CONFIG to check.
- extract the information from the target server log file (TgtSvr.log) Depending on the INCA version it is stored in
 - C:\ETAS\LogFiles\TgtSvr
 - or
 - C:\ETAS\LogFiles\ProcessLogsV2
 Have a look for the line containing the string "StorePermanentlyToTarget", e.g.

- Logfile excerpt:
 12-05-2006 17:59:52 API: StorePermanentlyToTarget (CAN1,
 C:\DOCUME~1\...\Temp\flshtmp1.hex 0 cbbnew convert
 D:\ETASDATA\INCA5.3\PROF\FCU_CB~1\xcellsis)
 where CAN1 is the Flash Port entry, 0 cbbnew convert is the flash parameter and
 D:\ETASDATA\INCA5.3\PROF\FCU_CB~1\xcellsis is the PROF configuration file
 entry.

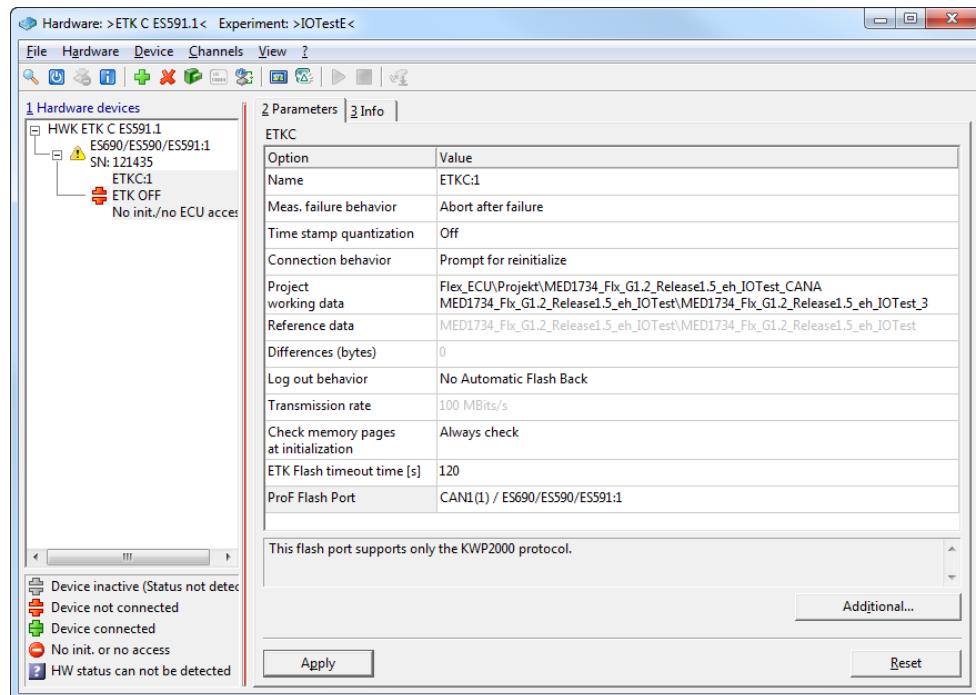


Fig. 2: Hardware dialog in INCA

- The (EAF) PROF Configuration File is holding the path to the PROF layout file (the filename has no extension (see Figure 3) and Figure 1

Name	Größe	Typ	Geändert am
cbbnew.pri	4 KB	PRI-Datei	12.08.2009 15:45
cbbonly.pri	2 KB	PRI-Datei	12.08.2009 15:45
common.pri	2 KB	PRI-Datei	12.08.2009 15:45
convert.pri	2 KB	PRI-Datei	12.08.2009 15:46
e_mess.pri	2 KB	PRI-Datei	12.08.2009 15:46
fcu_id.dll	80 KB	Application Extension	12.08.2009 15:46
FCU_ID.TMP	1 KB	TMP-Datei	12.08.2009 15:45
flash.pri	3 KB	PRI-Datei	12.08.2009 15:45
install.ini	1 KB	Configuration Settings	12.08.2009 15:45
noconv.pri	1 KB	PRI-Datei	12.08.2009 15:45
PROFINCA.CFG	2 KB	CFG-Datei	12.08.2009 15:45
progdata.pri	2 KB	PRI-Datei	12.08.2009 15:45
prognew.pri	4 KB	PRI-Datei	12.08.2009 15:46
xcellsis	6 KB	SpeedDial	12.08.2009 15:45
XCELLSIS	1 KB	Datei	12.08.2009 15:45
xcellsis.prm	2 KB	PRM-Datei	12.08.2009 15:45

Fig. 3: The layout file is without any extension; therefore the type is Datei or file respectively

- the (EAF) PROF Flash Parameter is holding the parameter string given to the PROF program when it is called by INCA.
 The parameters used to flash your ECU can be extracted from the Logfile (see page

27) or you can take it from the layout file directly (see Figure 4).

In both cases your parameters for the here shown example will be: **0 cbbnew convert**

```

1 3 1 1 "File #name:      " 1 1 1 1 3
2  "*.S19"
3
4
5 2 1 1 "#Baud rate:      " 1 1 1 6 4
6  "KWPonCAN" "0"
7  " 10.4 kBaud" "1"
8  "125.0 kBaud" "2"
9  "250.0 kBaud" "3"
10
11 2 1 1 "#Action:         " 1 1 1 5 4
12 "Program ECU cbb + program + data" "cbbnew convert D:\ETASDATA\INCA5.3\PROF\FCU_CB~1\xcellsis"
13 "Program ECU program + data"      "prognew convert D:\ETASDATA\INCA5.3\PROF\FCU_CB~1\xcellsis"
14 "Program ECU data"                "progdata convert D:\ETASDATA\INCA5.3\PROF\FCU_CB~1\xcellsis"
15 "Program ECU cbb"                 "cbbonly convert D:\ETASDATA\INCA5.3\PROF\FCU_CB~1\xcellsis"
16
17
18

```

Figure 4 Contents of PROF layout file

SuT mapping file

Is the same you use for your ECU measurement access.

3.7.3 Lines in code

Initialization of the port:

```

private IPortEAF m_portEAF = null;
private IPortEAC m_portEAC = null;
private IPortMA m_portMA = null;

/// <summary>
/// Registers the ports.
/// </summary>
private void RegisterPorts()
{
    m_portMA = Factory.GetPortMA("ModelAccess");
    m_portMA.Timeout = -1;

    m_portEAC = Factory.GetPortEAC("ECUAccessCalibration");
    m_portEAC.Timeout = -1;

    m_portEAF = Factory.GetPortEAF("ECUAccessFlashing");
    m_portEAF.Timeout = -1;
}

```

And to perform some flash action add the following parts to your test performance method:

```
#region ECU access flashing
```



```
Reporting.SectionBegin("Flash access");
m_portEAF.Configure("device2", "default", "default");
m_portEAF.DeviceFlash("workingpage");
// workingpage is the test label for the working page in INCA
// the testlabel is resolved by the SuTMapping

if (m_portEAF.ChecksumIsValid())
{
Reporting.SetInfoText(0L, "Flashed ETK successfully", 6L);
}

// The following flash command flashes the ECU with a given hex
or s19 file
//m_portEAF.UuTFlashFile(@"C:\Users\Public\Documents\ETAS\LABCAR-AUTOMATION 4.1.0\Examples\Test Bench Configurations\ATCL
Example NEW Test Bench for LC050\Demo03.hex");

// The following flash command flashes the ECU with the working
or reference page
//m_portEAF.UuTFlash("workingpage");

Reporting.SectionFinished("", new Verdict(VerdictCode.Pass));

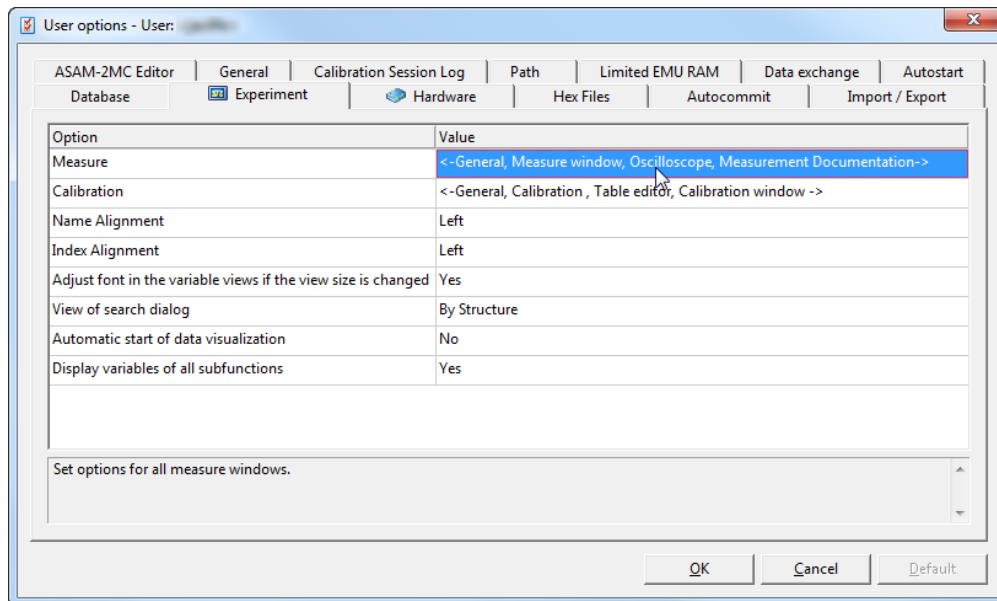
#endregion ECU access flashing
```

You can find the full test case in the LABCAR-AUTOMATION examples.

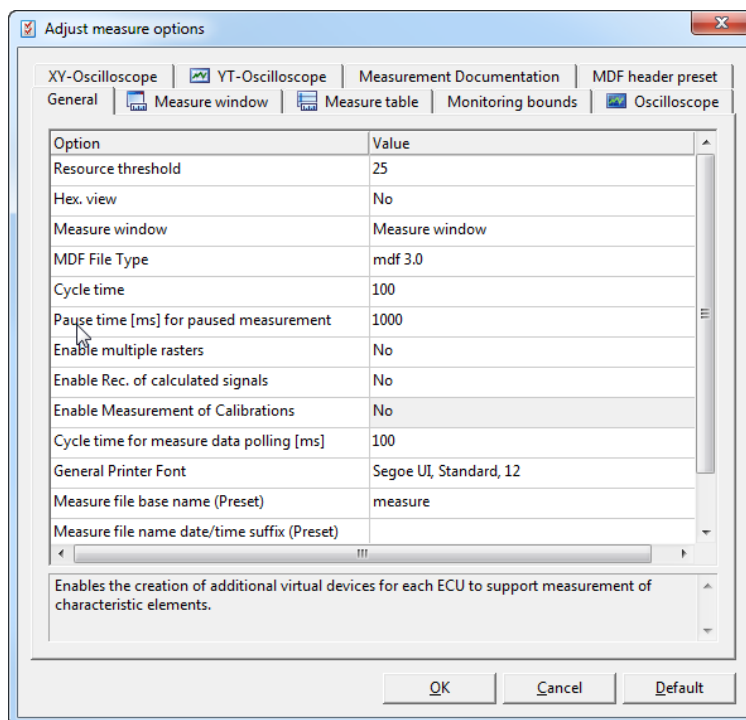
3.8 Parameter recording for Sync DL

To use the INCA parameter recording for the SyncDL you have to set an user option inside INCA to enable it.

Open your INCA and select in the Menu Bar: Options -> User Options and activate the tab 'Experiment'.



After double click at value for measurement in this window the panel for the change of the measurement options open.



In the Change Measurement options – window select the tab ,General' and look for the entry ,Enable Measure Calibration devices' and set it to ,yes'.

3.9 Failure Simulation Load Cut off

At the FS-Port for failure simulation it is possible to cut off the load via test case for each failure set separately.

To use this functionality you've to reference the ATCL (Etas.Eas.Atcl.Interfaces.dll) build version 1.0.0.4 and to use the Load Flag property of the ErrorDefinition.

The load cut off for the complete port by a parameter in the tool configuration file is still available.

This option setting (unless it is empty) is overwriting any settings inside the test case!

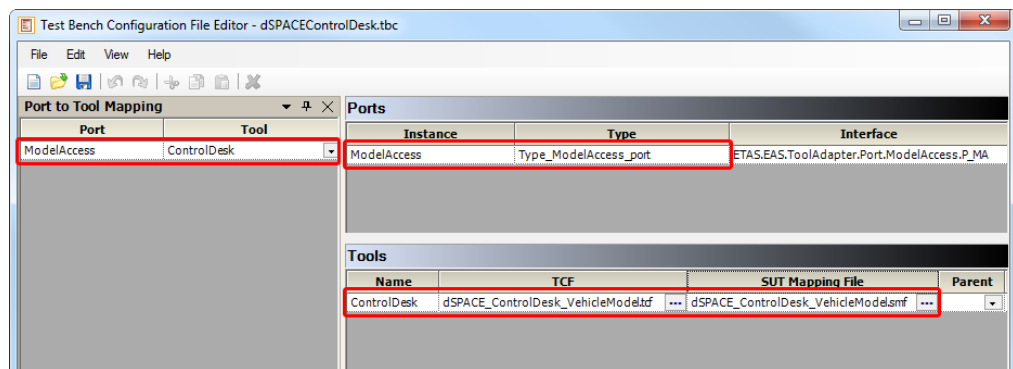
3.10 Connection to dSpace Test Bench

Connection to dSpace test benches is very easy as ETAS provides already a Tool Adapter for it. This Tool Adapter enables the connection to dSpace ControlDesk 3.2.1

Configuration

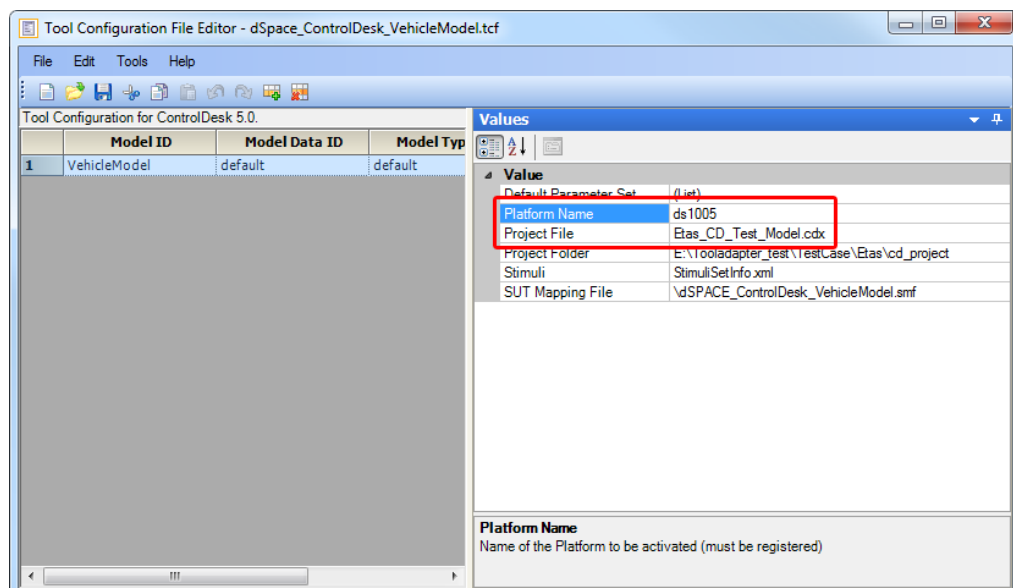
The changes in the test bench are tool configuration related only:

- the Model Access – which is used for the access via LABCAR-OPERATOR as well - has to bind to the Tool Adapter for dSpace ControlDesk like shown in the following picture:



Thus the Model access Port (in this example called 'ModelAccess') with all its functionality ("signatures") is available now for the test cases.

- the Tool Configuration File has to point to the dSpace Control Desk project and to the platform which is used. These are the main differences to the Tool Configuration File of the LABCAR-OPERATOR.



The dSpace ControlDesk Project file has the extension .cdx

The SuT Mapping File is slightly different to the one for LABCAR-OPERATOR due to the different syntax in the dSpace ControlDesk. SUT Mappings naturally differ in tool labels (“Right hand side” of mapping). Additionally not only labels may differ, but also object and data types.

Test Label	Tool Label
Engine	IC/Pedal2NEngine/Limit_EngineSpeed/n_Engine
Accelerator	IC/Accelerator/Value
AcquisitionTask	Acquisition
AirConditionTorque	HW//IC.Pedal2NEngine.AirConditionTorque.physical.const
AirConditionTorque.mode	HW//IC.Pedal2NEngine.AirConditionTorque.physical.mode
AirConditionTorque.sg	HW//IC.Pedal2NEngine.AirConditionTorque.physical.sg
AirConditionTorque.sgchannel	HW//IC.Pedal2NEngine.AirConditionTorque.physical.sgchannel
EngineSpeedLimit	IC/A_IdleController/A_IdleController/EngineSpeedLimit
Idle	IC/A_IdleController/A_IdleController/Idle
PIdle	IC/A_IdleController/A_IdleController/PIdle
StartAirCondition	IC/Pedal2NEngine/AirConditionM...

Test Label	Tool Label
A_Throttle	Model Root/Diesel Engine Vehicle Model/Vehicle Model/Engine...
a_Vehicle	Model Root/Diesel Engine Vehicle Model/Vehicle Model/Vehicle...
Accelerator	Model Root/Diesel Engine Vehicle Model/Driver/Accelerator_...
Accelerator_	Model Root/Diesel Engine Vehicle Model/Driver/Accelerator_...
Accelerator__NUMBER(1)_	Model Root/Diesel Engine Vehicle Model/Driver/Longitudinal C...
Accelerator__NUMBER(2)_	Model Root/Diesel Engine Vehicle Model/Vehicle Model/Acceler...
Accelerator__NUMBER(3)_	Model Root/Diesel Engine Vehicle Model/Vehicle Model/Engine...
AcceleratorC	Model Root/Diesel Engine Vehicle Model/Driver/D_Shifting/Acc...
AcceleratorC__NUMBER(1)_	Model Root/Diesel Engine Vehicle Model/Driver/D_Shifting/Acc...
AcceleratorControl	Model Root/Diesel Engine Vehicle Model/Driver/D_Shifting/D_S...
AcceleratorDelayed	Model Root/Diesel Engine Vehicle Model/Driver/AcceleratorDel...

Example of a LABCAR-OPERATOR SuT Mapping File (left) / dSpace ControlDesk related SuT Mapping File (right)

Use the example TBC, TCF or SMF out of the examples folder at C:\Users\Public\Documents\ETAS\LABCAR-AUTOMATION 4.x\Examples\Test Bench Configurations\Sample Files

Differences between LABCAR-OPERATOR and dSPACE ControlDesk

- Signal Generation is limited to one object by dSPACE ControlDesk
- Datalogging functionality is limited by dSPACE ControlDesk
- Some Data Types are missing in ControlDesk
 - boolean,
 - char, string
 - bitstrings; octetstrings, hexstrings
- Some Variable Objects are missing but exist in ControlDesk
 - “Map”/”2D-Table”
 - “Curve”/”1D-Table”
- Initial variable creation by “ModelValueSelect” is not necessary in dSPACE Test Bench

4 For Testers

4.1 Changing the layout of the report

There are two possibilities to change the layout of your report. Just have a short view about the technique behind.

The reports are stored as xml format and are formatted by the browser you use to display. This can be the Microsoft Internet Explorer or our LABCAR-AUTOMATION Report viewer.

Browsers use style sheets (xslt files), referenced to format the input for the display. The LABCAR-AUTOMATION Report viewer copies these style sheets into each report directory, basing on its default option settings.

4.1.1 Changing report layout after creation per report

To change the layout of one single report, this is possible after the creation of the report (and only then). You just have to change the style sheets, copied to the same location like the report:

<your LCA project location>\TestRuns\Report <date and time> \Standard Report\styles

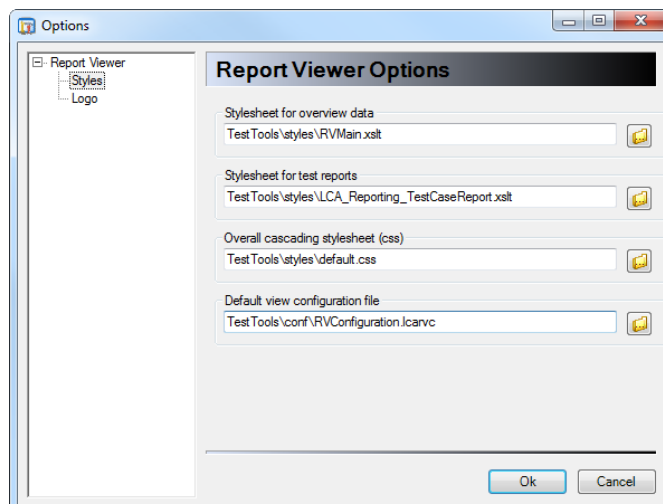
You have to reopen the report for viewing again after changing the style sheets.

4.1.2 Changing report layout for all reports

To change the layout of all further reports you have to change the style sheet located in the program folder of LABCAR-AUTOMATION (C:\Program Files (x86)\ETAS\LABCAR-AUTOMATION4.x\TestTools\styles) and to adapt the option inside the report viewer.

Please restart the report viewer after changing this option

For existing reports the styles are not adapted, you have to do it manually



5 **Complex features**

5.1 **Offline Project generation**

5.1.1 **Typical Use Case**

LABCAR-AUTOMATION projects are a collection of several artifacts working together during test management and execution: test case, parameter set, units under test, test bench configuration, tool configurations. On folder level they are well structured and one file defines the complete content of the project.

With respect to cost and time savings projects are designed to be reapplied for different test targets, test benches, test phases or with only different parameter sets. If only one of the artifacts has to be changed and the other ones should be kept and shall be consistent overall it is useful to manage the artifacts at a central place.

The project creation is separated out of the definition of the single peaces and often the request is there to automate it.

The Project Generator is the choice for acting like this.

5.1.2 **Features of the Project Generator**

Having all artifacts of a LABCAR-AUTOMATION project available at any place in the folder structure the Project Generator assembles the project out of it controlled by a so called Project Management File (.pmf).

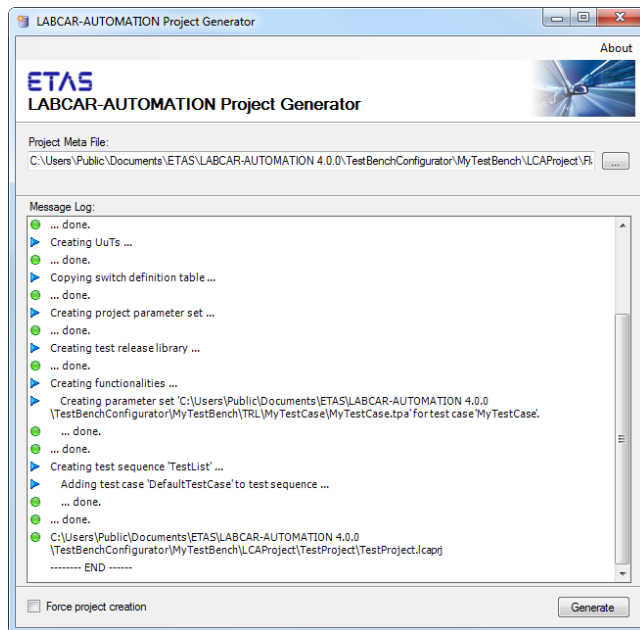
5.1.3 **Feature Handling**

To provide all inputs to the Project Generator a Project Meta File (.pmf) has to be created. It is structured in XML and contains the link to all LABCAR-AUTOMATION project artifacts. All of the artifacts have to be created before, e.g. by creating a new or changing an existing LABCAR-AUTOMATION project.

The format follows a given schema. The complete schema is delivered within the installation directory at: C:\Program Files (x86)\ETAS\LABCAR-AUTOMATION 4.x\TestTools\XMLSchema\lca_pmf_v1.1.xsd

This .pmf file is the only input to be entered at the dialogue of the Project Generator to enable the generation. With the option 'Force project creation' the Project Generator is instructed to overwrite an existing project.

Just press 'Generate' and it starts.



5.2 Report Structure 'Abstract Section'

5.2.1 Feature Description

Most of the information during test run are collected, reported and viewed according to the chronological cycle of the test.

Nevertheless a summary of the results – more detailed than the verdict only – is interesting to be seen aggregated and right at the top of the report or of a dedicated section, even if the information are partly originated at a late point in time of the test run.

This the Abstract Section in the report allows for.

5.2.2 Test case development for use of the Abstract Section

The Abstract Section is a special section inside the report. The test case has to border its report parts for this special section with the two signatures.

- Reporting.SectionAbstractBegin
- Reporting.SectionAbstractFinished

The entry instructions for the Abstract section can be used several times per test case

And it also can be nested inside of other sections. In this case the abstract section content relates to the section and is shown at the top of this section. One abstract section per report section is possible.

The abstract section has to be realized in the following structure elements:

- TestCaseBegin/TestCaseEnd
- SectionBegin/SectionFinished
- TestEntityCalled/TestEntityFinished
- FunctionCalled/FunctionFinished

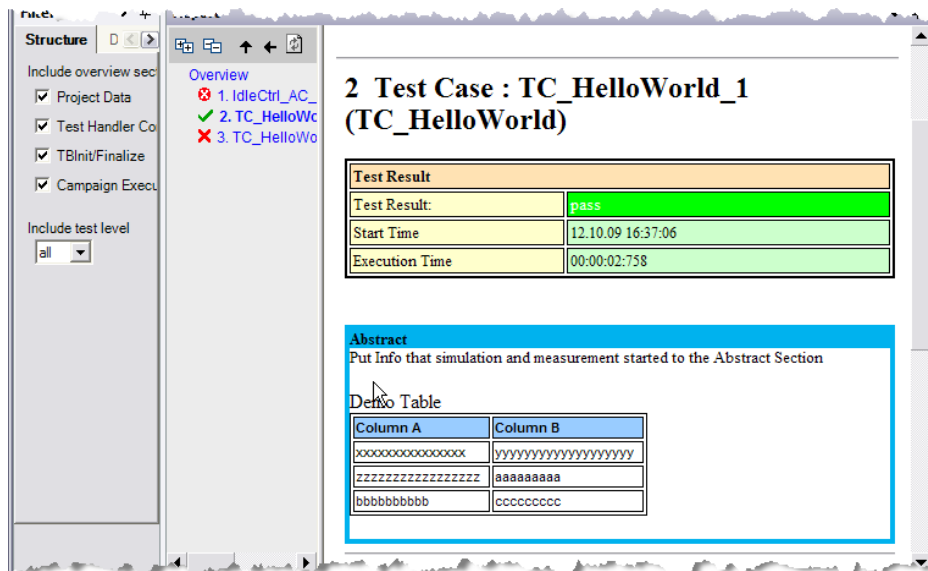
The abstract section allows the following structure elements/methods inside:

Name	Description
AddPlot2Report	Overloaded.
AddTableToReport	Add the table to the report.
FunctionCalled	Overloaded.
FunctionFinished	Overloaded.
SectionBegin	Adds a new Section to the report.
SectionFinished	Finishes a section in the report.
SetErrorLevel	Set the error level.
SetErrorLink	Sets the error link.
SetErrorText	Sets the error text.
SetErrorTextAndValue	Sets the error text and value.
SetHeaderLink	Sets the header link.
SetHeaderText	Sets the header text.
SetHeaderTextAndValue	Sets the header text and value.
SetInfoLink	Sets the info link.
SetInfoText	Sets the info text.
SetInfoTextAndValue	Sets the info text and value.
SetLink	Set a link.
SetResultLink	Sets the result link.
SetResultText	Sets the result text.
SetResultTextAndValue	Sets the result text and value.
SetText	Set some Text.
SetTextAndValue	Set some text and some values.
SetWarningLink	Sets the warning link.
SetWarningText	Sets the warning text.
SetWarningTextAndValue	Sets the warning text and value.
StateCalled	Overloaded.
StateFinished	Overloaded.
TestEntityCalled	Overloaded.

TestEntityFinished Overloaded.

5.2.3 Report View with Abstract Section

The Abstract Section is shown by the report viewer at the top of the report or the section. Using the overview page of a test case you find the content of the Abstract Section at the top as chapter 'Abstract':



Using the link in the tree to a section of the test case you find the content of the section related abstract at the chapter 'Abstract' at to top of this dedicated section:

5.3 Improved Data logging functionality

5.3.1 Typical use case

If customer uses standardized LABCAR-OPERATOR projects, which have to be used for the simulation and tests it might be useful to predefine Dataloggers for values which have to be logged in any case. Depending on the use case the test case or simulation sequence may add some values to be logged for a more detailed view on the behavior.

1.1 Section : Initialization

Test Result	
Test Verdict:	pass
Test Result:	Initialization

Abstract
All parameter registered successful. Find details below

The test case will be initialized

Parameter A Test Parameter registered

Parameter Upper Case conversion registered

Parameter Sleep Time before Evaluation registered

1.2 Section : Simulation and Measurement

Test Result	
Test Verdict:	pass
Test Result:	Simulation and Measurement

Abstract
All parameter read successful

The effort for the basic Dataloggers have not to be spent twice. However the test case is able to log all necessary details without overloading the LABCAR-OPERATOR project with too much Dataloggers.

5.3.2 Feature Description

Former LABCAR-AUTOMATION releases create their own Datalogger in the LABCAR-OPERATOR project when a test runs, independently of the Dataloggers which already exist in the LABCAR-OPERATOR project.

With the newest version of LABCAR-AUTOMATION it is now possible to access Dataloggers that exist in the LABCAR-OPERATOR project already.

This feature is optional, by means that it can be selected, if the LABCAR-OPERATOR Dataloggers shall be visible and usable inside the LABCAR-AUTOMATION test cases, or not.

5.3.3 Feature Handling

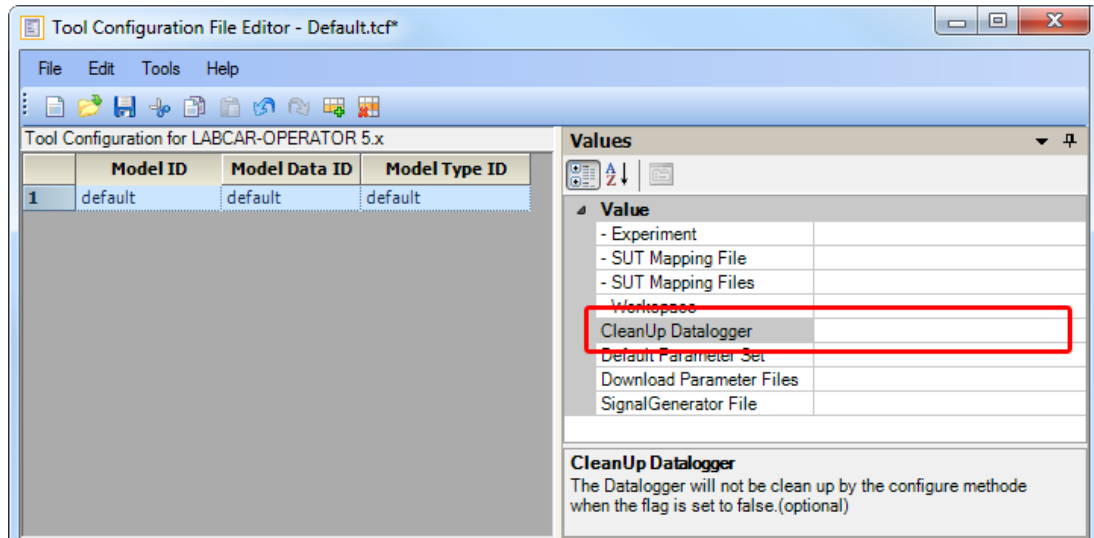
The new feature is provided by LABCAR-AUTOMATION as a new entry inside the tool configuration of the model access. It is only available from LABCAR-OPERATOR V5.0 onwards.

There are no code changes necessary for the option to use the LABCAR-OPERATOR project dataloggers in the LABCAR-AUTOMATION test case as well. It is only controlled by the tool configuration of the LABCAR-OPERATOR.

If the test case uses the same value for logging like already defined by the LABCAR-OPERATOR project and if the use of LABCAR-OPERATOR project dataloggers is enabled, LABCAR-AUTOMATION merges both dataloggers automatically.

5.3.4 Test bench configuration for LCO project dataloggers

The tool configuration template of the LABCAR-OPERATOR V5.0 contains a new entry 'CleanUp Datalogger'. It accepts the values 'True' and 'False'.



In case 'False' is entered, the Dataloggers defined in the LABCAR-OPERATOR project are visible and usable by LABCAR-AUTOMATION as well.

To ensure the same behavior like in the past, the default value of this entry is 'True'. Leaving the entry blank has the same meaning like 'True'.

5.3.5 Hints

The entry field of this new entry is free text. Thus you can enter each string you want. The only value which is evaluated is 'False' or 'FALSE'. All other values are resulting internally to 'True'.

5.4 Maps and Curves

5.4.1 Typical use case

Signals which are recorded or used for stimulation are usually stored as a digital curve. It uses pairs of dedicated values for the time line (x-axis) and the amplitude (y-axis). A set of signals can be stored together as a map, using the same values for the time line. The synchronous datalogging is a representative use case for these maps.

5.4.2 Feature Description

LABCAR-AUTOMATION is able to work with curves and maps using two dedicated TypeSut – Types, called TypeSut1DFloatTable and TypeSut2DFloatTable.

5.4.3 Feature Handling

Reading measurement values or feeding parameter values as maps or curve these types can be mapped accordingly to the tool labels.

Access to the single values and the time line point are possible via index values.

5.4.4 Mapping rules

Test case type	SuT Mapping type
TypeSut1DFloatTable	Curve
TypeSut2DFloatTable	Map

5.4.5 Example

```
// A very simple Test Case Example snippet

TypeSut1DFloatTable resultCurve =
(TypeSut1DFloatTable)Ports.ECUAccessCalibration.GetValue(new
TypeSut1DFloatTable("",
"One_D.STD_sint16_sint16.All_Curves_STD.Curves_Raster_1", "The Test
Label is mapped in the SutMapping File", new double[0], new
double[0], 0.0, 0, 0, 0, "", ""));

TypeSut2DFloatTable resultMap =
(TypeSut2DFloatTable)Ports.ECUAccessCalibration.GetValue(new
TypeSut2DFloatTable("", "Group_Two_D_1.Map_Mod", "The Test Label is
mapped in the SutMapping File", new double[0], new double[0], new
double[0], 0, 0, 0, 0, 0, 0, 0, "", "", ""));

// assign the first two amplitude figures to the curve
resultCurve.ValueY[0] = 100;
resultCurve.ValueY[1] = 200;

// define a map containing values from 0 .. map length for x- and y-
axis

// assign the x-axis figures to the map
int index = 0;
for (int i = 0; i < resultMap.ValueX.Length; i++) {
    resultMap.ValueX[i] = index;
    index++;
}
// assign the y-axis figures to the map
index = 0;
for (int i = 0; i < resultMap.ValueY.Length; i++) {
    resultMap.ValueY[i] = index;
    index++;
}
// assign the z-axis figures to the map
index = 10;
for (int i = 0; i < resultMap.ValueZ.Length; i++) {
    resultMap.ValueZ[i] = index;
```

```

        index++;
    }

    // download curve and map settings onto the ECU
    Ports.ECUAccessCalibration.SwitchSectionPage("WorkingPage");
    Ports.ECUAccessCalibration.SetValue(resultCurve);
    Ports.ECUAccessCalibration.SetValue(resultMap);

    // read curve and map back
    TypeSut1DFloatTable resultCurve2 =
        (TypeSut1DFloatTable)Ports.ECUAccessCalibration.GetValue(new
        TypeSut1DFloatTable("",
        "One_D.STD_sint16_sint16.All_Curves_STD.Curves_Raster_1", "The Test
        Label is mapped in the SutMapping File", new double[0], new
        double[0], 0.0, 0, 0, 0, 0, "", ""));

    TypeSut2DFloatTable resultMap2 =
        (TypeSut2DFloatTable)Ports.ECUAccessCalibration.GetValue(new
        TypeSut2DFloatTable("", "Group_Two_D_1.Map_Mod", "The Test Label is
        mapped in the SutMapping File", new double[0], new double[0], new
        double[0], 0, 0, 0, 0, 0, 0, 0, "", "", ""));

    // compare written and read figures of the curve
    if(resultCurve.ValueY.LongLength == resultCurve2.ValueY.LongLength)
    {
        for (int i = 0; i < resultCurve.ValueY.Length; i++) {
            if( resultCurve.ValueY[i] != resultCurve2.ValueY[i]) {
                Fail();
            }
        }
        for (int i = 0; i < resultCurve.ValueX.Length; i++) {
            if (resultCurve.ValueX[i] != resultCurve2.ValueX[i]) {
                Fail();
            }
        }
        Pass();
    }
    else {
        Reporting.LogExtension("Length of written and read back curve is
        different");
        Fail();
    }

    // compare written and read figures of the map
    if (resultMap.ValueY.LongLength == resultMap2.ValueY.LongLength)
    {
        for (int i = 0; i < resultMap.ValueY.Length; i++) {
            if (resultMap.ValueY[i] != resultMap2.ValueY[i]) {

```

```
        Fail();
    }
}
for (int i = 0; i < resultMap.ValueX.Length; i++) {
    if (resultMap.ValueX[i] != resultMap2.ValueX[i]) {
        Fail();
    }
}
for (int i = 0; i < resultMap.ValueZ.Length; i++) {
    if (resultMap.ValueZ[i] != resultMap2.ValueZ[i]) {
        Fail();
    }
}
Pass();
}
else {
    Reporting.LogExtension("Dimension of written and read back map
is different");
    Fail();
}
Pass();
```

5.5 Real Time Tests support

5.5.1 Typical use cases

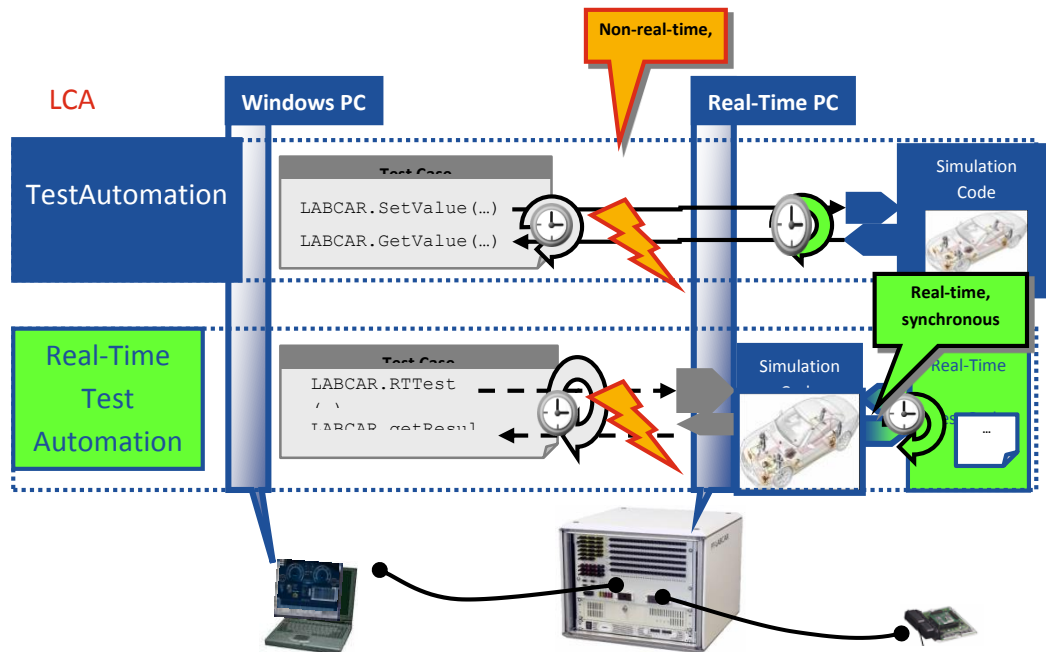
The strong networking of ECUs and other controlling elements in cars leads more and more to a time dependant interconnection. Especially the use case of a bus system where information are exchanged randomly and with a high frequency of messages leads to the necessity to observe the exact point in time as soon as messages are received.

Due to asynchronous design of communication between sensors, actors and different ECU's, the test of the behavior regarding reaction along the timeline becomes important. To realize such tests the target reaction has to be detected like in the reality – in real time.

The possibility to perform tests at the real time environment (Real Time PC) of a LABCAR covers this demand. Downloading new parts of code onto the Real Time PC during and thus expand the model during test execution is another quite exciting benefit of this feature.

5.5.2 Feature Description

In case action and reaction of several values calculated by the model the point in time of setting or measuring a value might impact the behavior. Without real time test possibility both actions are depending on the performance and signal transmission time, which can lead to unintentional delays. These parts of the test which are time critical in this sense should be performed as real time tests.



The Real Time PC (RTPC) of the LABCAR expects C-code. LABCAR-AUTOMATION is able to trigger the download of such code onto the RTPC, to set it into operation (activate) or to deactivate it. Precondition for usage of this feature are:

- LCO 5.0 as Model access tool
- Prepared Model (hooks introduced)
- LABCAR as the test bench with RTPC for Simulation
- Existing C-code for download

5.5.3 Feature handling

To use the real time feature the model has to be prepared for it. Models use 'hooks' to enable docking of alternative code. These hooks are the input and output interfaces for the new code.

First you have to provide the C-code for the download onto the RTPC. One possibility to develop such code is the LABCAR-ASC. LABCAR-ASC is an ASCET Add-On and comes with LABCAR-OPERATOR. It provides among others the ASCET Target "RTPC" and enables compilation on RTPC.

For more information on creating real time C-code and preparing the model please refer to the user guides of RTPC and LABCAR-OPERATOR.

When developing your test case you have to use a new port 'RealTimeScripting' to initiate the following actions

- Code Download
- Activation of downloaded code
- Deactivation of downloaded code

to use the Real Time Testing feature.

For executing LABCAR-AUTOMATION provides a special Tool Adapter (Test Bench Connector Real Time Test – LABCAR-TBCRT) for access to this feature. It has to be configured in the test bench configuration.

5.5.4 Test case development for Real Time testing

The port providing the access to the real time test functionality is provided by LABCAR-AUTOMATION with in the ATCL (Automotive Testing Class Library):

- Ports.RealTimeScripting

The creation and closing signatures are like usually

- Create
- ConfigureTool
- Configure
- Close

Special signatures for the real time test handling itself are:

- AddFiles – to pass the initial set of C-files to RTPC for later activation
- Activate – activation of the downloaded code at RTPC
- GetRTPCState – receive the current state of the RTPC
- Deactivate – deactivation of the downloaded code at RTPC

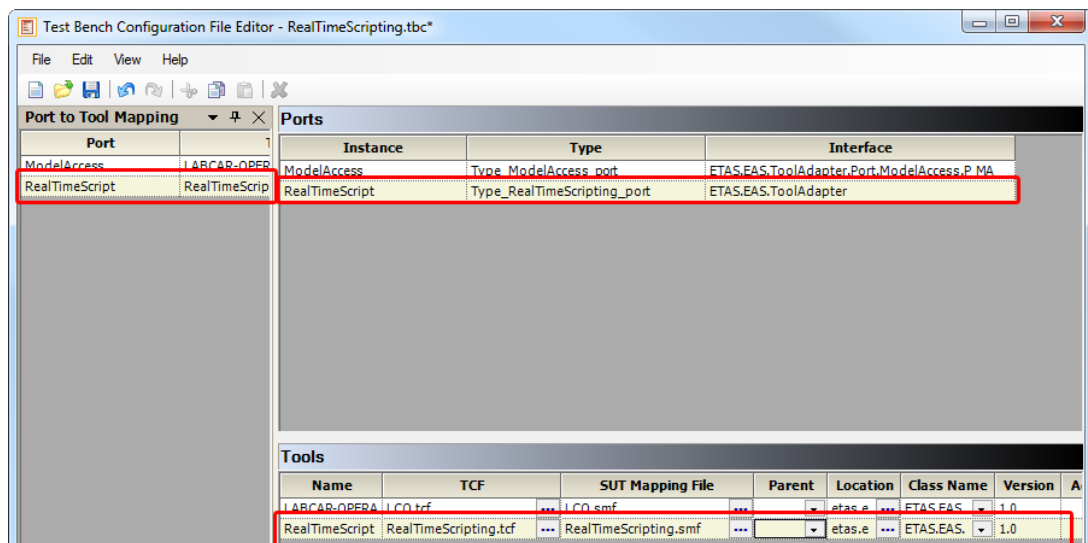
To get detailed information on the test case design find an example, delivered with the installation of LABCAR-AUTOMATION, at:

C:\Users\Public\Documents\ETAS\LABCAR-AUTOMATION 4.x\Examples\Test Case Development\TC_RealTimeScripting

5.5.5 Test bench configuration for Real Time testing

Within the LABCAR-AUTOMATION installation a tool adapter ,...' is provided . To use the feature the test bench has to configure it.

Example of a test bench configuration with LABCAR-OPERATOR 5.0 / Experiment Environment 3.2:



5.5.6 Hints

Datalogging with labels of the new code is not possible directly. The reason is the ignorance of the new code labels – and thus of the mapped test labels – during test bench

initialization. A loophole out of this drawback is to 'handover' the values to the original downloaded model and to log these model values.

5.6 Soft-Stop Function for Test cases

5.6.1 Typical use cases

Big test runs

In the case that the test run contains a lot of test cases and the test cases themselves have a long duration an erroneous behavior of one test case might 'kill' the complete test run. In this case it should be possible just to stop this single test case and continue with the next one.

The already performed test cases may not run again and the following test cases can run without any further actions.

Wrong test case coding

Test cases may include endless loops by accident. A test run stop would wait until this endless test case is finished, which will never happen. The 'Stop Test' is a way out of this.

5.6.2 Feature Description

In the past the stop button in Test handler stopped execution of the complete test run, but did not finish the actual running test case before. The process was killed hardly, thus the test bench left in an undefined, unpredictable state.

Now this additional choice is available via button push inside the Test Handler. The test case receives the information about this stop and is able to react on this button push. Thus the developer can decide, what has to be performed when the stop of the test case occurs.

5.6.3 Feature Handling

In the test handler the button 'Stop' is extended with a small menu. The tester can select the kind of stop which has to be carried out.



For faster reaction of the tester in front of the desktop the different stop activities can be triggered via short cut key combinations like shown in the picture right beside.

The test case which has been stopped with 'Stop Test Case' or Shift+F10 will have the status 'interrupted' in the Test Handler's execution overview:

5.6.4 Test case development for Soft Stop

The LABCAR-AUTOMATION library provides for its class TestCase a new virtual method StopTestCase. This Method is prepopulated with no statements and does simply nothing, except to set the verdict to 'inconc'.

To enable the test case to react on the event it just has to override this method with the appropriate actions.

```
public override void StopTestCase()
{
    int count = 0;
    while (count != 10)
    {
        Reporting.SetText(0,0,"Stopp Test Case",0);
        Thread.Sleep(250);
        count++;
    }
}
```

This example just writes out ten times the "Stopp Test Case" string in time lags.

Typical Actions within this StopTestCase Method might be:

- Report, that the test case was stopped by an outside request
- Report the status of the test bench, e.g. the status of each port
- Measure and Log some important values
- Report the state of the test case
- Finalize the test bench, close all open ports or reset the test bench

5.6.5 Test bench configuration for Soft Stop

There is no special test bench configuration necessary for this feature.

5.6.6 Hints

To prevent the running test cases or test run directly following the interrupted test case from failure, each test case should provide a StopTestCase behavior by overriding the method. At least a reset of the test bench should be performed to prepare a clean test bench for the next test.

If the overridden method runs endless the Test Case will be not closed!

5.6.7 Difference between status shown in Test Handler and Verdict reported by test case

Test Handler status calculation and Verdict reporting by Test case

The Test Handler shows in the overview pane the Status of test case execution. If the test case finishes its operation normally the status is been build of its verdict (pass, fail or inconc). In the case the test case is not working (error in coding), the test bench was set up wrongly or the test case is interrupted this is shown in the Status column of the Test handler as values 'error' or 'interrupted'.

In opposite to this the Report shows the final result of the test case – its verdict. If no verdict could calculated at the end due to interruption or error this status is taken over as result.

What does this mean for this feature?

When a test case is stopped by the tester using the new Stop button, the test handler sets the status 'Interrupted' immediately as soon as the button was pressed and fires the appropriate event. This is done independently of the test case behavior (as the test case design is completely under developer's control).

In some cases the report contains a verdict pass or fail, even if the test handler shows 'Interrupted'.

Following two situations can lead to this rare case:

- The test case is 'almost' finished. In this glare situation the finish of the test case and the stop event may occur at the same point in time. They are overlapping each other, the event is fired but does not reach the test case anymore. LCA is not able to set the verdict to 'incon' (by default, defined in the provided virtual method) due to the stop event. The interrupt of the test case is marked in the Test Handler as 'interrupted' even if the test case has been finished normally.
- The test case has overridden the virtual method and do not set the verdict hardly to 'incon' but to another one (eg. 'pass' or 'fail'), depending on its state or results before. E.g. if there is a cool down of the engine performed after all test case steps. In this case the verdict-relevant test part has finished and has evaluated a valid verdict already, which should not be overwritten by an interrupt anymore.

5.7 Error Manager

5.7.1 Typical Use Cases

In some cases you like to check the counter of messages of one of the used tools of your test bench to check it inside the test case. If the tool underneath the LABCAR-AUTOMATION does not report e.g. a warning via its API – may be it is not relevant in the automated mode – the test case is able to check if a warning was produced.

5.7.2 Feature Description

With the Error Manager it is possible to read the counter of errors, warnings or information messages logged by a tool. Currently only the Experiment Environment version 3.2.1 supports this feature.

5.7.3 Feature Handling

The feature is available for programmed test cases, e.g. in C# or another .Net compatible software language. It can be used by referencing the necessary error logging library in your test case development project. This library provides you with the methods to

- Get the counter of Application Error Messages
- Get the counter of Application Warning Messages
- Get the counter of Application Information Messages
- Get the counter of Hardware Error Messages
- Get the counter of Hardware Warning Messages
- Get the counter of Hardware Information Messages
- Reset all Application Message Counters
- Reset all Hardware Message Counters

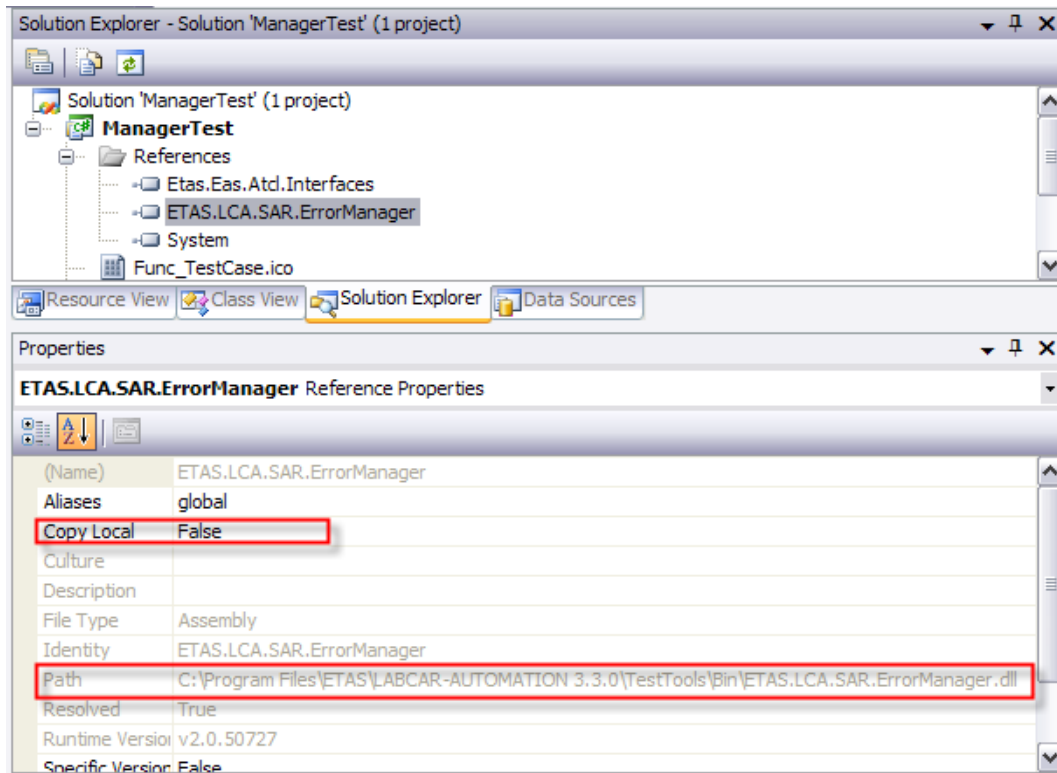
5.7.4 Test Case development with Error Manager

Library reference

Add the ErrorManager reference to your Test Case.

You find the ETAS.LCA.SAR.ErrorManager.dll to be referenced at C:\Program Files (x86)\ETAS\LABCAR-AUTOMATION x.y.z\TestTools\Bin, whereby C:\Program Files (x86)\ETAS\LABCAR-AUTOMATION x.y.z\ is your actual LABCAR-AUTOMATION installation folder.

In the following picture you find as example a Visual Studio Solution with a test project.



How to Use the ErrorManager in the Test Case

- Constructor

```
public IErrorManager errorManager
{
    get
    {
        return
        (IErrorManager) Factory.GetManager ("IErrorManager");
    }
}
```

- Available methods

```
- int applicationerror =
  errorManager.GetApplicationErrorMessageCount (Ports.ModelAccess);

- int applicationwarning =
  errorManager.GetApplicationWarningMessageCount (Ports.ModelAcc
```

```

    ess);

-   int applicationinfo =
    errorManager.GetApplicationInfoMessageCount (Ports.ModelAccess
    );

-   int Hardwareerror =
    errorManager.GetHardwareErrorMessageCount (Ports.ModelAccess);

-   int Hardwarewarning =
    errorManager.GetHardwareWarningMessageCount (Ports.ModelAccess
    );

-   int Hardwareinfo =
    errorManager.GetHardwareInfoMessageCount (Ports.ModelAccess);

-   errorManager.ResetAllApplicationMessageCount (Ports.ModelAccess
    );

-   errorManager.ResetAllHardwareMessageCount (Ports.ModelAccess);

```

5.7.5 Test Bench Configuration for Error Manager

There are no configurations necessary in the test bench to use the Error Manager.

As from the Example above (Chapter 4.7.4) the Port Name must fit to the name used in the test case – like usually for all calls to this port.

5.7.6 Hints

Due to the fact that only the Experiment Environment supports this feature, the method will throw an exception in case of any other tool is referenced by the given port (parameter of the methods).

5.8 Working with a Signal Generator

5.8.1 Typical use cases

To stimulate the model reproducible with a dedicated signal traces, thus reactions are comparable you need a possibility to store and ,re-call' this trace.

In some case it is necessary to stimulate different signals with a dedicated offset in time or started equally at exactly the same point in time. For this you should have a possibility to define these points exactly.

Third use case is the continuous realtime stimulation. The signal feed is created under real time condition.

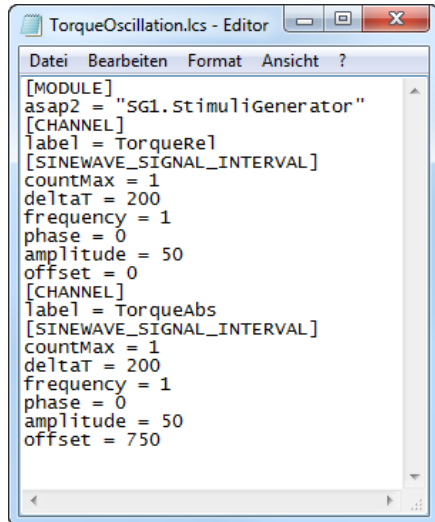
5.8.2 Feature Description

The feature uses different files to provide the input for the signal generation.

First of all you need the signal feed itself – the values which have to be set at certain points in time. This input file is a description of the signal. It can be created out of a recorded signal by LABCAR-OPERATOR (extension .lcs) or edited by hand.

The input of this file will be assigned to a signal generator. The Signal Generator, running at the real time environment, is build up the signal and connects it to the signal input at the model. Thus the model uses this as signal feed instead of the output of a device or other model parts.

The generation can be configured, started and stopped.



```

[MODULE]
asap2 = "SG1.StimuliGenerator"
[CHANNEL]
label = TorqueRel
[SINWAVE_SIGNAL_INTERVAL]
countMax = 1
deltaT = 200
frequency = 1
phase = 0
amplitude = 50
offset = 0
[CHANNEL]
label = TorqueAbs
[SINWAVE_SIGNAL_INTERVAL]
countMax = 1
deltaT = 200
frequency = 1
phase = 0
amplitude = 50
offset = 750

```

5.8.3 Feature Handling

The signal generator is part of the model access port. Thus the configuration information are included in the model access Tool Configuration File.

The set of information regarding signal to be used, acquisition task and file to be used are provided inside a stimuli set file additionally. This file will be automatically created by the configuration wizard at your test bench configuration folder at the subfolder \TBC and called per default StimuliSetInfo.xml. (see 4.8.5 Test bench configuration for the Signal Generator)

In case of LABCAR-AUTOMATION Standard Package is installed the name and storage location must not be changed!

In case of LABCAR-AUTOMATION's Shell Component 'Editors Package' is licensed, you can change name and storage location of this file and reference this directly in the tool configuration file.

Following steps are necessary when you like to use this feature:

- Provide configuration information for the signal generator
 - A traced signal – available as file of type .lcs (see 4.8.2 Feature Description)
 - A stimuli set information file, provided by Configuration Wizard or manually created.
 - If you do not use the Configuration Wizard you find an example at C:\Users\Public\Documents\ETAS\LABCAR-AUTOMATION 4.x\Examples\Test Bench Configurations\Demo Test Bench\LCO 5.0\StimuliSetInfo.xml*
 - A configuration information in the Tool Configuration File.

If you use the Configuration Wizard the Tool Configuration File is created and prefilled with the correct values and options.

- SuT Mapping Information in the SuT Mapping File to map signal name, channel, mode and acquisition task
- Configure the signal generator and connect it to signals to be stimulate
 - Test case steps to configure (defining signal name and stimuli set file)
 - Connection to the correct signal channels
 - Using LABCAR-OPERATOR V3.2.5:
The connection has to be done in the LABCAR-OPERATOR in the signal list manually before starting the automated test.
 - Using LABCAR-OPERATOR V5.0 or higher:
The connection has to be done inside the test case with the method SetModelValue. (see 4.8.4 Test case development for the Signal Generator). The values have to set the signal mode, the signal name and the channel name.
The separate connection to dedicated signals is possible only with the LABCAR-OPERATOR!
- Start/Stop the stimulation
 - Test case steps to configure start, pause (optional) and stop the signal generator. (see 4.8.4 Test case development for the Signal Generator).

5.8.4 Test case development for the Signal Generator

In case you programm your test cases with C# you can find here an example of the code:

- first part – configure the signal generator

```
m_maport.ConfigureSignalGenerator(
    "ACTorqueSignal", //SignalGenerator name <SG .../> from
    StimulisSetInfo.xml
    "pulse"); // SGSet name <SGSet .../> from StimuliSetInfo.xml,
    selects .lcs file
```

- second part – connect the signal generator to the correct channels

```
m_maport.SetModelValue(new TypeSutString("",
    SutMapping.SIGNALSIGNALGENERATOR, "", "ACTorqueSignal"));
m_maport.SetModelValue(new TypeSutString("",
    SutMapping.SIGNALSIGNALGENERATORCHANNEL, "", "Torque"));
// channel label inside the lcs file
// Available modes:
// 0 = constant
// 1 = stimuli
// 2 = model
// 3 = stim + model
// 4 = stim * model
```

```
TypeSutFloat mode = new TypeSutFloat("", SutMapping.SIGNALMODE,
"Mode value", 3.0, 0, 0, "");
```

```
m_maport.SetModelValue(mode);
```

- // third part – start and stop the signal generator

```
m_maport.Start();
m_maport.StartSignalGenerator("ACTorqueSignal");

Thread.Sleep(4000);

m_maport.StopSignalGenerator("ACTorqueSignal");
m_maport.Stop();
```

The separate connection to dedicated signals is possible only with the LABCAR-OPERATOR!

With LABCAR-OPERATOR V5.0 and higher it is mandatory to connect the signal generator after configuration

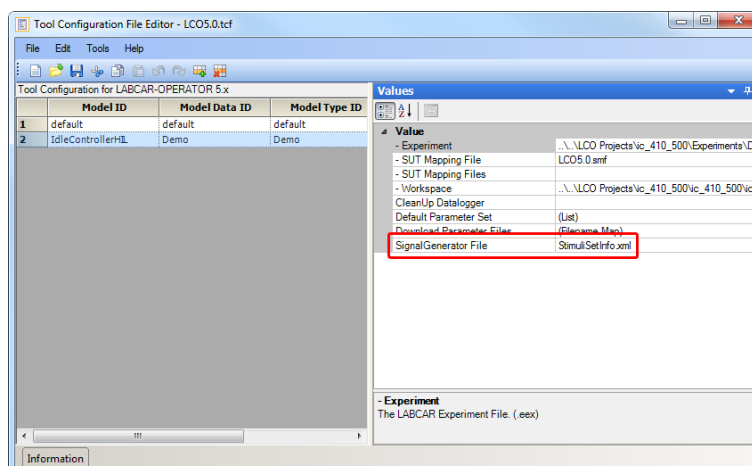
The information regarding the signal (name and signal set name) have to match the information of the stimuli information set. (see 4.8.5 Test bench configuration for the Signal Generator)

The information which channel name, signal generator name and signal mode has to be used for the signal to be stimulated you find in the signal list / signal center of LABCAR-OPERATOR. Please have a look into the User guide of the appropriate version of LABCAR-OPERATOR.

5.8.5 Test bench configuration for the Signal Generator

The configuration information is provided by the Configuration Wizard by default.

If you like to change this information or if you do not use the Configuration Wizard you need to install the Tool Configuration File Editor, which is part of the Editors Package.



For the format of the StimuliSetInfo file is like follows:

- StimuliSetInfo.xml (for LCO):

```
<?xml version="1.0" encoding="utf-8" ?>
<File type="Stimuli Set Info" extension="XML" version="1.0">
  <SG name="ACTorqueSignal">
    <SGModelName name="SignalGenerator_0" />
    <AcquisitionTask name="TaskDVEModel" />
    <SGSet name="pulse" file="TorqueOscillation.lcs"/>
  </SG>
</File>
```

5.8.6 Configurable Signal Generator Input Mode Mapping

LABCAR-AUTOMATION 4.2.2 allows choosing between two different mapping behaviors for the Signal Generator input mode.

The mode can be defined by the newly introduced flag "Inport Mode" within the test bench configuration (tcf).

- **Inport Mode = False (Default Mode):** Mapping behavior represents for compatibility reasons the behavior of previous version of LABCAR-AUTOMATION as introduced with older versions LABCAR-OPERATOR Experiment Environment. This is the default behavior of LABCAR-AUTOMATION 4.2.2.
 - CONST = 0
 - STIMULI = 1
 - MODEL = 2
 - STIMULI + MODEL = 3
 - STIMULI * MODEL = 4
- **Inport Mode = True (Latest Mode):** Mapping behavior represents the latest behavior of the supported LABCAR-OPERATOR Experiment.
 - CONST = 0
 - MODEL = 1
 - MODEL_PLUS_CONST = 2
 - MODEL_MULT_CONST = 3
 - MODEL_PLUS_SIGNALGENERATOR = 4
 - MODEL_MULT_SIGNALGENERATOR = 5
 - SIGNALGENERATOR = 6
 - SIGNALGENERATOR_PLUS_CONST = 7
 - SIGNALGENERATOR_MULT_CONST = 8

Hint: This feature is only supported with the tool adapter for LABCAR-OPERATOR Experiment Environment 3.5 or higher!

5.8.7 Mappings

All information to be used inside the test case for the signal generator have to be provided accordingly in the mapping file. This is independent if you use C# as programming language or the Automation Sequence Builder to set up the test case graphically.

Mappings are necessary for

- Signal channel = <label>.sgchannel
- Signal mode = <label>.mode
- Signal generator= <label>.sg
- Acquisition task

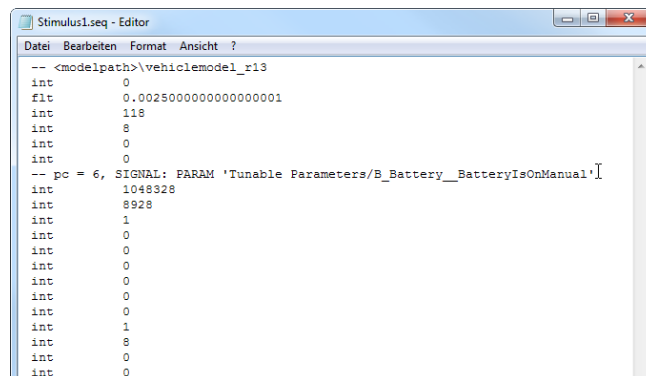
Whereby <label> is the special signal tool label of LABCAR-OPERATOR.

5.8.8 Working with dSpace

- StimuliSetInfo.xml (for ControlDesk)

```
<?xml version="1.0" encoding="utf-8"?>
<File type="Stimuli Set Info" extension="XML" version="1.0">
  <SG name="SG">
    <SGModelName name="SignalGenerator_0" />
    <MPSubSystem name="" />
    <SGSet name="start" file="Stimulus1.seq" />
    <SGSet name="stop" file="Stimulus2.seq" />
  </SG>
</File>
```

- Signal trace file - input for signal generator (see example picture right side)



5.8.9 Example with LABCAR-OPERATOR 3.2.5

To get a general idea please find here an overview of the used data in the different tool windows.

```

m_maport.ConfigureSignalGenerator("ACTorqueSignal", //SignalGenerator name <SG _> from StimulusSetInfo.xml
    pulse); // SGSet name <SGSet _> from StimulusSetInfo.xml selects .ics file

m_maport.SetModelValue(new TypeSutString("", SutMapping.SIGNALSIGNALGENERATOR "", "ACTorqueSignal"));
m_maport.SetModelValue(new TypeSutString("", SutMapping.SIGNALSIGNALGENERATORCHANNEL "", "Torque")); // channel label inside the ics file

// Available modes:
// 0 = constant
// 1 = stimuli
// 2 = model
// 3 = stim + model
// 4 = stim * model
TypeSutFloat mode = new TypeSutFloat("", SutMapping.SIGNALMODE "Mode value", 3.0, 0.0, "");

m_maport.SetModelValue(mode);

m_maport.Start();

m_maport.StartSignalGenerator("ACTorqueSignal");

Thread.Sleep(4000);

m_maport.StopSignalGenerator("ACTorqueSignal");
    
```

Name	Elec. Value	Phys. Value	Center
Physical Type	Stim + Model		
Value	Inf		
Signal Generator	SignalGenerator_0		
Signal Generator Channel	Torque		

Test Label	Tool Label	Type	Hierarchy
SignalSignalGeneratorChannel	HW//IC_Pedal2Engine_AirConditionTorque.physical.sgchannel	Measurement, Scalar Float	Parameter
SignalSignalGenerator	HW//IC_Pedal2Engine_AirConditionTorque.physical.sg	Measurement, Scalar Float	Parameter
SignalMode	HW//IC_Pedal2Engine_AirConditionTorque.physical.mode	Measurement, Scalar Float	Parameter
Signal	HW//IC_Pedal2Engine_AirConditionTorque.physical	Measurement, Scalar Float	Parameter
Accelerator	ICAcceleratorValue	Parameter, Scalar Float	Parameter
AirConditionTorque	HW//IC_Pedal2Engine_AirConditionTorque.physical.const	Parameter, Scalar Float	Parameter
PIdle	ICJA_IdleControllerJA_IdleController_P	Parameter, Scalar Float	Parameter
IIdle	ICJA_IdleControllerJA_IdleController_I	Parameter, Scalar Float	Parameter
EngineSpeedLimit	ICJA_IdleControllerJA_IdleController_n_Engine_max	Parameter, Scalar Float	Parameter
AirConditionTorque.sg	HW//IC_Pedal2Engine_AirConditionTorque.physical.sg	Parameter, Scalar Integer	Parameter
AirConditionTorque.sgchannel	HW//IC_Pedal2Engine_AirConditionTorque.physical.sgchannel	Parameter, Scalar Integer	Parameter
AcquisitionTask	Acquisition	Parameter, Scalar String	Parameter
AirConditionTorque.mode	HW//IC_Pedal2Engine_AirConditionTorque.physical.mode	Parameter, Switch, mode-switch	Parameter

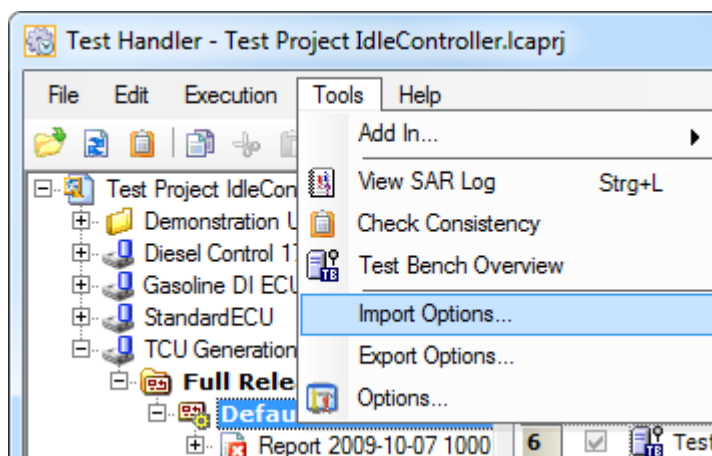
6 General issues

6.1 Test Handler Tool Options – across different installations

You may change your options in the Test Handler, e.g. for Reporting or UuT-TBC mappings. After installation of a new version it is rather conceivable you like to have these options available for the new version as well. Another requirement might be to save different tool option configurations for further reuse.

The functionality you need, is to import and export these options. You find the new functions in the Tools Menu.

To take over the content of the imported ToolOptions.conf file, a restart of the Test Handler is mandatory.



Additionally the options file ToolOptions.conf will not be deleted during uninstall of LABCAR-AUTOMATION from Version 3.4 onwards. Complete this file will not be overwritten during re-installation. Thus you have the options available right after re-installation or repair.

An import is not necessary.

If you like to force to overwrite the Test Handler Tool Options during re-installation or repair to the initial default values, please delete the ToolOptions.conf file before starting installation or repair.

For details see the Release Notes.pdf you find on your LABCAR-AUTOMATION Installation CD and at the ETAS program folder.

6.2 Silent Installation

LABCAR-AUTOMATION can be silently installed via the command line. Therefore call 'setup.exe /silent'.

Examples:

- To specify which feature to be installed use the "feature" option like 'Setup.exe /features="Core,Doc,Basic,TestDesign,ReportViewer,Examples,Shell,TestCreation,TestManager,TestHandler,Editors,Misc,ConfigurationWizard" /silent'
- To specify ALL Features to be installed use the below option
Setup.exe /features="ALL" /silent

- To Install ReportViewer alone, use the below option
Setup.exe /features="Core,Basic,ReportViewer" /silent

In the following list, the LCA feature names used for silent installation are listed:

⇒ *Please be aware, that every LCA installation must contain the core, otherwise the operation of LCA will fail*

LABCAR-AUTOMATION feature name	LABCAR-AUTOMATION components
Core	All core components of LABCAR-AUTOMATION including Tool Adaptors for INCA, LCO/EE and INTECRIO
• Doc	User manuals and product documentation.
• Basic	
○ TestDesign	The main LCA modules (required for operation): Executor (Engine), Automotive Test Case Library (ATCL)
○ ReportViewer	The Report Viewer to view the test reports.
○ Examples	The examples folder
Shell	
• TestCreation	Automation Sequence Builder
• TestManager	Test Manager
• TestHandler	Test Handler
• Editors	Editors Package
• Misc	Project Generator
○ ConfigurationWizard	Configuration Wizard
ToolAdapters	
• RealTimeScriptingAdapter	Real Time Testing Tool adaptor
• dSpaceAdapter	dSpace ControlDesk Tool adaptor

• ODXAdapter	Diagnose Tool adaptor for ODX Link (acc. INCA Version)
• ES4440Adapter	Fault Simulation Tool Adaptor (ES4440)

For an overview on the LABCAR-AUTOMATION components please refer to the LABCAR-AUTOMATION release notes, chapter 'LABCAR-AUTOMATION Packaging', delivered with this CD and placed in the Programs / ETAS folder after installation.

All the other additional Components, like Tool Adaptors for MLBA4, CANape, Excel etc. are not handled in the LABCAR-AUTOMATION-4.x.msi (resp. Setup.exe). Separate installation is required, the relevant installation files (.msi) are provided in the Installation CD.

⇒ *Please be aware, that the LABCAR-AUTOMATION is completely controlled by licenses. This means, although you might have installed a specific component, it would not work in case you do not own an appropriate license. In this case please contact ETAS sales department, to order to correct component license*

The different addons (tool adapters) can be silently installed as given below:
 "msiexec /i <addon_installer>.msi /passive"

6.3 License Management

Every component of the LABCAR-AUTOMATION is protected by use of a license.

During Installation you are able to select 'your' package. If you select another option or more than the available licenses grant, all selected components are installed, however they are usable only in a 'Grace mode'. This mode allows you to test the full functionality of the components for a time frame of 30 operational days. After expiration of this time the components without valid license do not work any longer.

For further particulars we refer you to the documentation "How to get a license file" listed in the tool section of the Installation CD and the user's guide to LABCAR-AUTOMATION.

Note:

Machine based licenses do not work together with Microsoft's "Remote Desktop Connection", but you can use "Remote Desktop Sharing" of Windows NetMeeting, or use VNC.

There is no issue with Remote Desktop with a server based license.

7 **ETAS Contact Addresses**

ETAS HQ

ETAS GmbH

Borsigstraße 14

70469 Stuttgart

Germany

Phone: +49 711 89661-0

Fax: +49 711 89661-106

WWW: www.etas.com

ETAS Subsidiaries and Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:

ETAS subsidiaries WWW: www.etas.com/en/contact.php

ETAS technical support WWW: www.etas.com/en/hotlines.php

