Tachograph Software Test Tool

HENRIK CARUANA

KTH Information and
Communication Technology

Bachelor of Science Thesis
Stockholm, Sweden 2012

TRITA-ICT-EX-2012:269

Tachograph Software Test Tool

– Development and evaluation of test automation regarding the user interface component of the SE5000 tachograph

**ROYAL INSTITUTE
OF TECHNOLOGY**

Henrik Caruana
2012-09-25

Kungliga Tekniska Högskolan, KTH
School of Information and Communication Technology (ICT)
Examinator: Bengt Molin, bengtm@kth.se
Handledare: Anders Sundholm, företag Stoneridge Electronics,
anders.sundholm@stoneridge.com
Författarens e-postadress: caruana@kth.se
Utbildningsprogram: Högskoleingenjörsutbildningen Elektronik och Datorteknik, 180p

SAMMANFATTNING

Det här kandidatexamensarbetet beskriver utvecklingen av ett program som testar en mjukvarukomponent i SE5000 färdskrivaren och utreder om det är värt att vidareutveckla den här testmetodiken.

En av produkterna Stoneridge Electronics utvecklar är färdskrivaren SE5000.

En digital färdskrivare är ett elektroniskt system för att spela in kör- och vilotid för förare som kör kommersiella fordon. Fordonshastighet, körsträcka och andra systemrelaterade parametrar loggas också. Data sparas i ett minne inuti enheten och på förarspecifika smartkort.

Mjukvaran i enheten, SE5000, består av flera separata komponenter och man vill testa varje komponent för sig på mjukvarunivå. Genom att testa på mjukvarunivå förkortas återkopplingstiden för utvecklarna och eventuella buggar upptäcks tidigare.

Hög testtäckning är ett tydligt kvalitetstecken och är en trygghet för Stoneridge Electronics när enheten ska bli testad och funktionscertifierad av Transportstyrelsen.

Användargränssnittskomponenten ansvarar för att producera rätt utdata som svarar på förarindata och fordonsrelaterade parametrar. Tidigare har testning av användargränssnittskomponenten genomförts på systemnivå, dvs. mjukvaran har integrerats med hårdvaran och hela systemet testas genom ett extern gränssnitt. Tidsåtgången för att genomföra tester på det här sättet är väldigt stor och det är ett problem då man vill ha hög testteckning.

Stoneridge Electronics Justusprojekt vill kunna genomföra tillförlitliga, automatiserade tester på mjukvarunivå regelbundet eller vid mjukvaruförändringar. Det är också önskvart att etablera ett stort testfallsbibliotek som kan köras utan den stora tidsåtgången associerad med manuella tester. Dessa testfall bör vara enkla och snabba att implementera.

ABSTRACT

This thesis work describes the development of a program that tests a software component in the SE5000 tachograph and investigates if this test methodology and setup is worth further investigation.

One of the products Stoneridge Electronics develop is the tachograph SE5000.

A digital tachograph is an electronic system for recording driving and rest times for drivers and co-drivers of commercial vehicles. Vehicle speed, distance travelled and other system-related parameters are also logged. Data is stored in a memory inside the unit and on driver-specific smart cards.

The software of the SE5000 tachograph consists of several separate components that SE wishes to test separately on software level. By testing on software level the feedback loop is shortened and allows developers to catch possible bugs much earlier in the process.

High test coverage is a clear sign of quality and is a reassuring for Stoneridge Electronics when the unit is tested and a function certificate is issued by the Swedish Transport Agency.

The user interface component is responsible for delivering data in response to driver and vehicle input and stimulation. Previously the user interface testing has been performed on system level, i.e. software has been integrated with the hardware and the system has been put under test by using external interfaces of the target. The test execution time consumption is a problem when performing user interface tests on system level.

It is the projects wish to be able to perform reliable automated testes at a software level in order to run the test on a regularly basis or at software change. It is also desirable to establish a great variety of tests that can be executed without huge time consumption. These test cases should be easy and quick to implement.

ABBREVIATIONS

Definition	Explanation
SE	Stoneridge Electronics
SW	Software
UI	User Interface
TC	Test Case
TE	User Interface Test Engine
PC	Personal computer
VU	Vehicle Unit
DLL	Dynamic Link Library (filename extension)
HW	Hardware
GUI	Graphical User Interface
DDS	Driver Decision Support
RDI	Read Data Identifier
EXE	Executable (filename extension)
XML	Extensible Markup Language (filename extension)

Table of Contents

1. Introduction

1.1 Background

Stoneridge Electronics (SE) is a leading supplier to the automotive, truck, bus and off-road markets. Their products range from telematics systems, driver information systems, and tachographs through electronic control modules and power distribution center to multiplex systems and cockpit switch modules.

One SE product is the digital tachograph SE5000. A digital tachograph [1] is an electronic system for recording driving and rest times for drivers and co-drivers of commercial vehicles. Vehicle speed, distance travelled and other system-related parameters are also logged. Data is stored in a memory inside the unit and on driver-specific smart cards.

The way tachographs work is strictly regulated by law. In Sweden the Swedish Transport Agency is responsible for issuing a so called function certificate [2] consisting of the following three tests.

- Product compliance with Annex1b
- Product compatibility with all existing smart cards
- Product security standards are adequate

The certification is time consuming and expensive so it is desirable to have the product as tested and free from bugs as possible before the actual certification takes place.

The problem is that testing all the components and all possible scenario permutations in the vehicle unit (VU) is very time consuming. This leads to selectivity among tests and/or ineffective use of resources. Neither of which contribute to the effective development and quality of the product.

One of the most time consuming sets of tests are the Manual Entries. A manual entry is performed when there is unaccounted time for the driver. Unaccounted time occurs when the time in the VU and the withdrawal time of the driver card differ more than 1 minute. When this occurs the driver has to register what activity he/she has engaged in during the unaccounted time period.

Preparing and performing this particular type of test on system level is prone to errors and extra time demanding.

If testing could be automated then test cases (TC) could be created once and then put to regression. This would lead to higher test coverage and productivity, thus allowing for fewer bugs to slip through to system verification and a more efficient development.

Automated tests performed at software (SW) test level decreases the running time to practically zero.

All components producing any kind of measurable output can be subjected to this method of testing which is a big incentive for further investigation.

1.2 Purpose

The purpose of this thesis work is to develop a test tool and evaluate a test setup and methodology for testing SW components in order to automate components tests in the tachograph.

The SW component chosen to be evaluated is the user interface (UI) component. It is responsible for producing the data needed for the display and in response to manual entries.

1.3 Delimitations and definition of done

It is not expected to create a full test setup in order to test and evaluate the complete functionality of the UI component.

This thesis is limited to producing a proof of concept program that can create some basic test cases that are evaluated by the test engine set up by SE.

The following shall be achieved before the thesis work is considered done:

- The source code of the test tool is committed in software versioning and revision control system (from now referred to as version system)
- The source code of the test cases is committed in the version system
- Source code documentation has been approved by the SE5000 software project and committed in the version system.
- Test tool manual has been approved by the SE5000 software project and committed in the version system.
- A detailed documentation of the test setup has been approved by the SE5000 software project and committed in the version system.
- A detailed documentation of test cases has been approved by the SE5000 software project and committed in the version system.
- It shall be possible to rebuild the project successfully when the test tool source code is checked out from version system.
- A report is delivered to the SE5000 software project that states the appropriateness of using the test tool and the described test setup in order to perform software test of the UI software component. The report should also contain pros and cons of the test setup.

1.4 Test method and framework

The encapsulated loose coupled SW component UI needs to be compiled together with a stubbed environment that simulates the environment that communicates with the UI component on target. The stubbed environment also needs to supply interfaces that are externally accessible so that output from UI might be evaluated. The SW component is compiled together with its stubbed environment and test framework in order to create a dynamic link library file (DLL). See figure 1. The purpose of the test framework is to enable testing of the UI component when executing in the simulated environment.

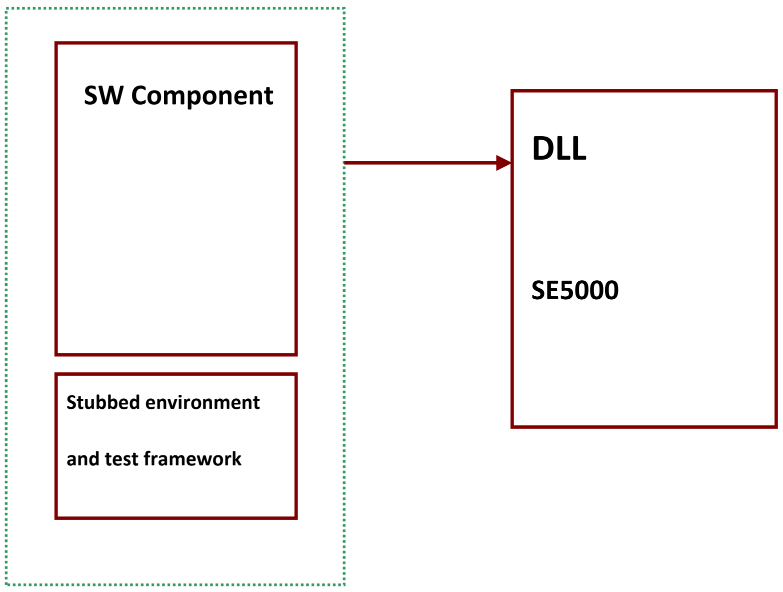


Figure 1. Use the compiler used for software test on HOST environment to create a dll-file.

The shared library file (DLL-file) is then integrated with a Microsoft Dialogue Application in order to send events to the UI component and to manually evaluate its behavior and output. The application should have a display where the graphical user interface is shown.

Further the application should implement a graphical user interface so that the user can make settings and send events to the UI SW Component that is put under simulation.

The application should also create a log of the simulation with actions taken by the tester, and also references to the expected output (display image). See Figure 2.

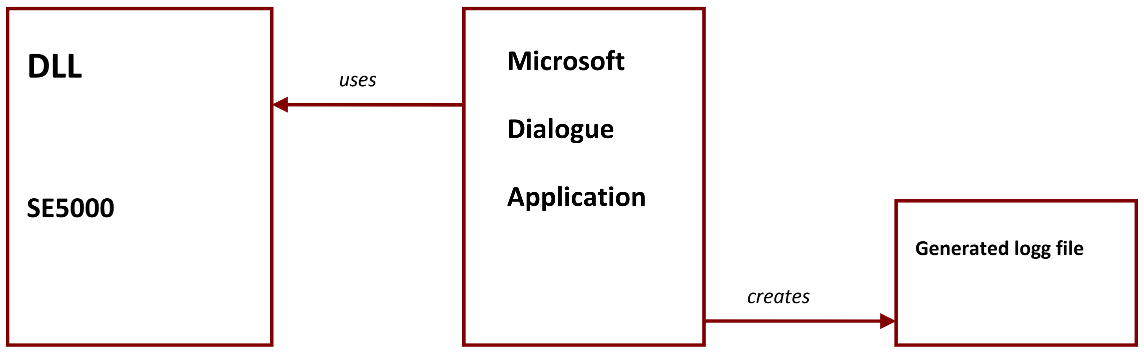


Figure 2. A Microsoft Dialogue Application uses the DLL-file to simulate the UI and generates a logg-file.

The Generated logg-file is a readable script that can be used by a script interpreting test engine that access the UI test component and uses the test environment in order to evaluate the expected result, see Figure 3.

The test framework uses the CUnit test framework in order to perform evaluation and generate an xml test report.

Hence the Generated logg-file will be used as a TC put to regression tests of the UI after once simulated. The created log file should also be possible to replay, using the application that communicates with the DLL.

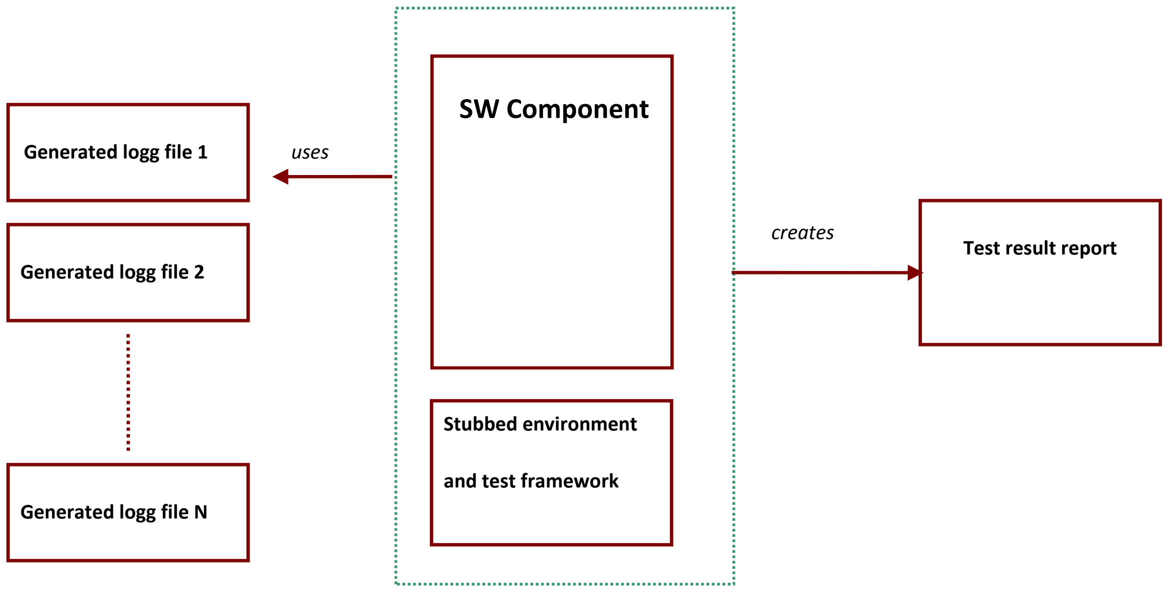


Figure 3. The SW Component UI with its test environment uses the generated scripts as test input.

1.5 Project time plan

The project method for this thesis is agile and iterative.

The author was responsible for the graphical user interface (GUI) so there was no clear requirement specification regarding the final design of the program. This resulted in design changes and modifications along the way to adapt to new features being implemented as well as input and suggestions from the customer, SE.

The test engine and test framework were developed simultaneously with the test tool. This made it difficult to estimate when different features would be implemented. As soon as new features were implemented in the test engine and framework they were also implemented in the test tool.

The only definite project dates are listed in table 1 below.

Table 1. Preliminary breakdown

Date	Duration	Activity
June 1st		Project start
		Familiarization with C# and subversion
		Architectural design of test tool
		Definition of script language and test case structure
		Handle dummy binary data file from UI component
		Communicate with test environment
		Proof of Concept
August 29th		Half time presentation and demo
		Documentation towards SE
		Documentation towards KTH
		Refinement of design and further development of functionality
September 30th		Handover of project

The features in table 1 above were modified and condensed as the work progressed. At the end of the project they could be summarized discussed in Chapter 3 The Development Process.

2. Frame of reference

This thesis work is an evaluation of a new approach of testing the SW in the UI component of the tachograph.

The UISWTestTool is built to work with the UI Test Engine which is a script based test engine for the Activities component in the SE5000 tachograph. The test engine and framework is described in the section 1.4 Test method and framework

The UI Test Engine has been developed alongside the UISWTestTool by developers at Stoneridge Electronics.

3. The Development Process

The development the UISWTestTool has been iterative. The core function and workflow of the program was decided early on in cooperation with SE. The functionality of the program gradually increased as the UI Test Engine was developed and the design of the program was modified many times.

As the progress of the UISWTestTool, design and function, relied almost completely on the development of the UI Test Engine it was difficult to predict the rate at which the project would progress.

The sections below describe the different stages of the project.

3.1 Familiarization with C# and setup

The language C-Sharp (C#) was chosen for development of the UISWTestTool since it was going to be run as a Microsoft Dialogue Application on PCs using Windows as Operating System. It is the author's opinion that C# is intuitive and has a simple mode for developing applications.

To get more familiar with C# tutorials and the MSDN .NET Library [4] were very helpful. The tutorial site most frequently visited was The New Boston [5]

The following programs were used for this project

- SharpDevelop [6] for C# development
- Notepad++ [7] for C# development
- UMLet [8] for UML diagrams.
- GIMP [9] for handling pictures
- Microsoft Office Word 2010 [10] for documentation

3.2 Program architecture

It was decided that the program would be made into two separate modules to make certain parts of the program reusable. One script generating- and one UI specific module.

The script generating module may be used in the future if this test methodology is adopted for other SW components in the SE5000 tachograph or possibly other projects.

The UI module would include a GUI for controlling the input and output going to and from the UI test framework. It would include buttons matching the ones on the tachograph panel and ways to set data in the stubbed environment in the framework.

3.3 Script generation

This approach of testing is built around scripting each test to make it possible to reproduce exactly the same tests automatically.

When the program is started you are presented with several fields of a script information form to fill out. This information is used to create a test case with a certain folder structure. A folder with the test case name will be created at the location you choose. In that folder a .txt script file and a folder named dispRef are also created.

The .txt file will contain all scripted commands sent to the UI Test Framework in that exact order as well as the test specification in a header, including requirements specification and purpose of the specific test.

The folder dispRef will contain all the binary reference pictures needed to verify the test case.

3.4 Managing display data

The data needed to represent the display is 293 bytes. When it is fetched from the test environment it is stored in a byte array in the dialogue application. The content is interpreted and displayed in the top display in the program with each pixel clearly distinguished in order to spot errors and inconsistencies.

It is possible to save the content of the byte array display buffer as a binary reference file with a uniquely identifiable name in the dispRef folder. When saving the content of the display buffer a command is entered into the script containing the command EVALUATE DISPLAY and the name of the binary reference file. At run time the test engine will compare the content of the display buffer with the content of the associated binary reference file and determine if they differ in any way.

A possibility to choose and open up a specific binary reference file is also available in order to inspect faulty binaries.

3.5 Communication with UI Test Framework

Communication with the test framework is done through a DLL interface. The entire test environment consists of several parts. It contains the UI component, the test framework, the stubbed control environment and some tests and initializations. This test environment is developed by the Stoneridge Electronics engineers and is described in section 1.4.

The DLL is imported into the UISWTestTool making several functions available.
These are the functions currently available

- *ui_dll_init():*	Initializes the entire test environment
- *ui_dll_pushScriptCommand():*	Pushes script commands to the test framework
- *ui_dll_getDisplayBufferData():*	Gets display buffer byte for byte

3.6 Proof of Concept

The final goal of this thesis work was to produce a proof of concept and evaluate if the method and test setup for testing the SW of the UI component was possible and if it was worth further development.

The proof of concept would consist of a program where a test case is created and the top level menus are cycled through and verified visually. Commands are to be recorded in the associated TC script and each menu is to be evaluated using the evaluate command.

The test case is then to be run by the test engine both using valid and corrupt commands and binary reference files.

3.7 Documentation and presentation

The documentation and presentation consist of a User Manual for the program, a Technical Description of the program, the thesis report and two presentations.

The User Manual contains information on the intended use of the program and how to use it as well as images and descriptions of the view of the different program tabs.

The Technical Description contains information about what kind of input the program can handle and what output it produces.

The User Manual and the Technical Description are both internal documents for SE.

The thesis report will be public and both KTH and SE will receive a copy.

3.8 Development beyond the scope and handover

All remaining time after the proof of concept and documents are done and have been approved will be spent on additional features and functionality to the UISWTestTool as well as revising the documents.

Handover of project will consist of a code review with a Stoneridge Electronics developer and a briefing where interested parties are informed where the tool is checked in and where the relevant documents are located.

4. Results

4.1 Current status

The result of this thesis work is a program called UISWTestTool created using C# as a Microsoft Dialogue Application. The program tests the software of the UI component in the SE5000 tachograph by communicating with a test environment containing the UI component, a test framework and a stubbed control environment.

The scripted commands sent to the test environment result in data being sent from the UI component, in this case to the display. The program records all commands sent in a script and can save the content of the display buffer as a binary reference file.

When the UI test engine runs the scripted tests it interprets the content of the script line by line and either sends the command to the UI component, sets new data in the stubbs, compares the current content of the display buffer with the content of the associated binary reference file or ignores the line if it is a comment.

The source code of the program is available in the SE Tools trunk of their repository. Once checked out it is possible to rebuild the program and run it on a windows host PC.

A User Manual as well as a Technical Description of the program exist in their document repository.

4.2 Architecture

In this section it is described how the UISWTestTool connects to the surrounding environment.

The vehicle unit consists of a HW and a SW part. The SW part is made up by many separable components. The UI component is integrated in the SW test environment along with stubbs, a test framework and some basic tests. The SW test environment can be compiled into two different files a DLL and EXE. UISWTestTool uses the DLL to communicate with the test environment and create TCs. The EXE version of the TE runs all added TCs and produces an XML result file that can be interpreted by the developers.

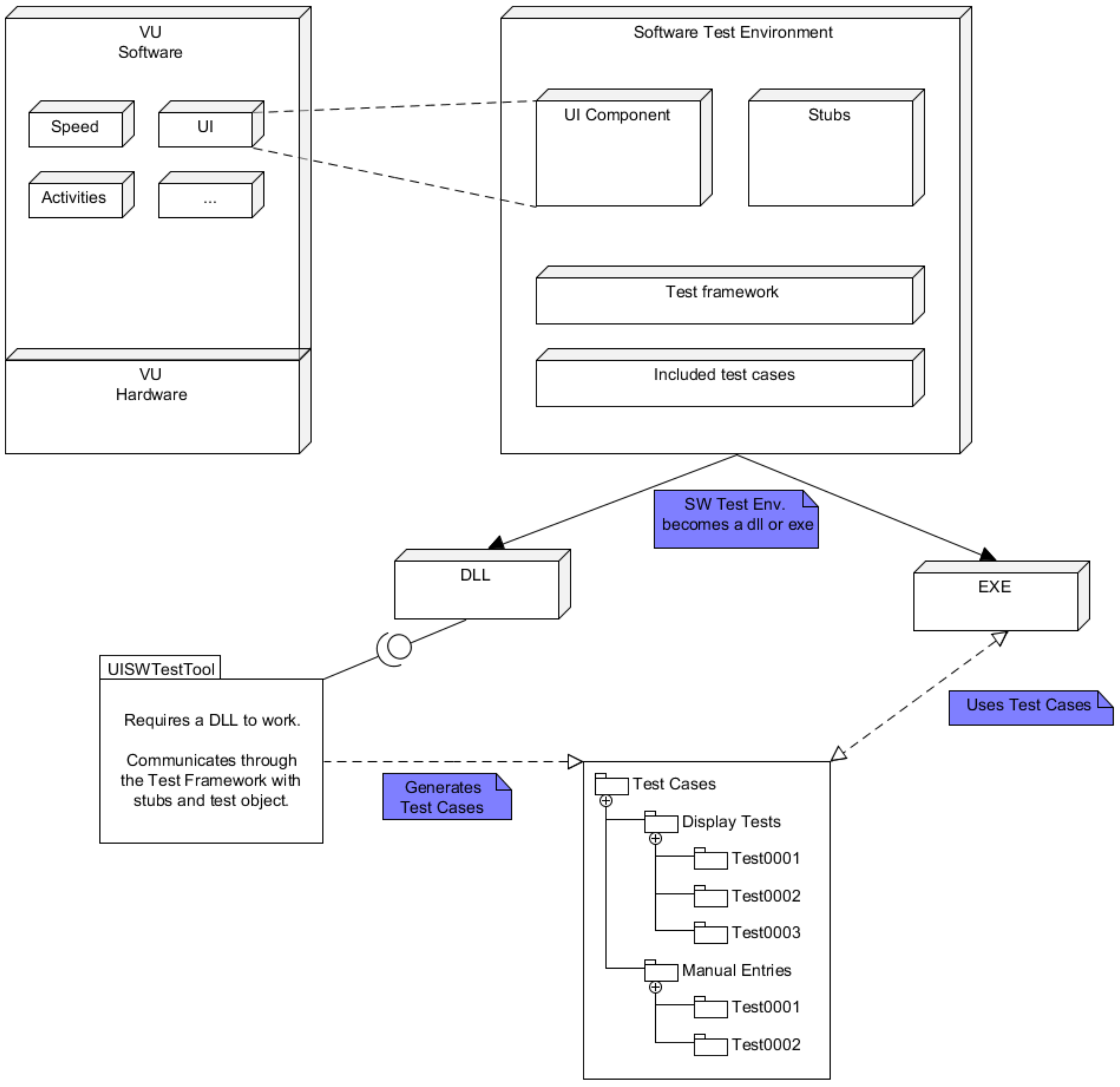


Figure 4. How the test setup and program are connected

4.3 Test case creation work flow

The activities diagram below illustrates the work flow of the UISWTestTool when creating a TC

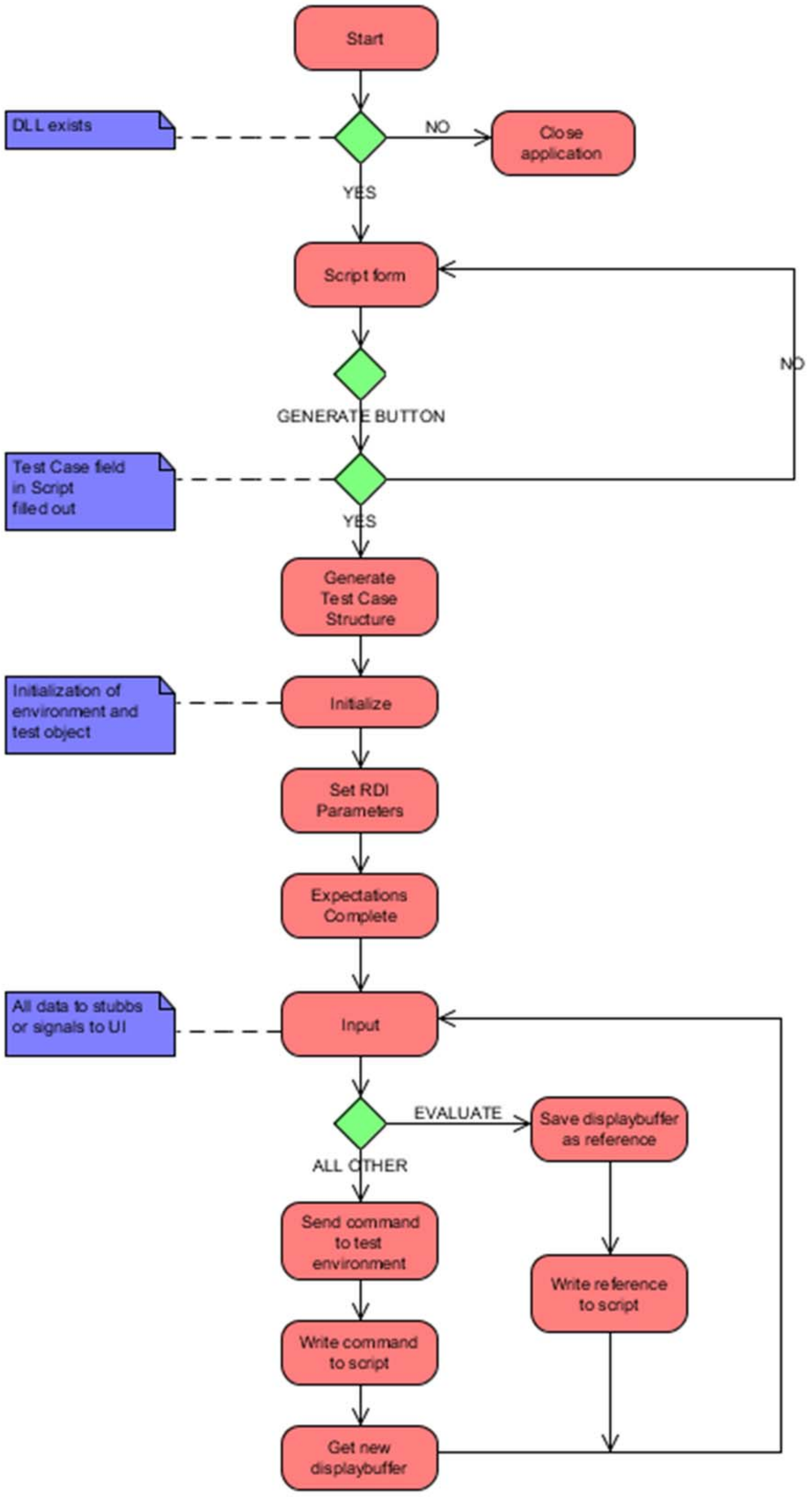
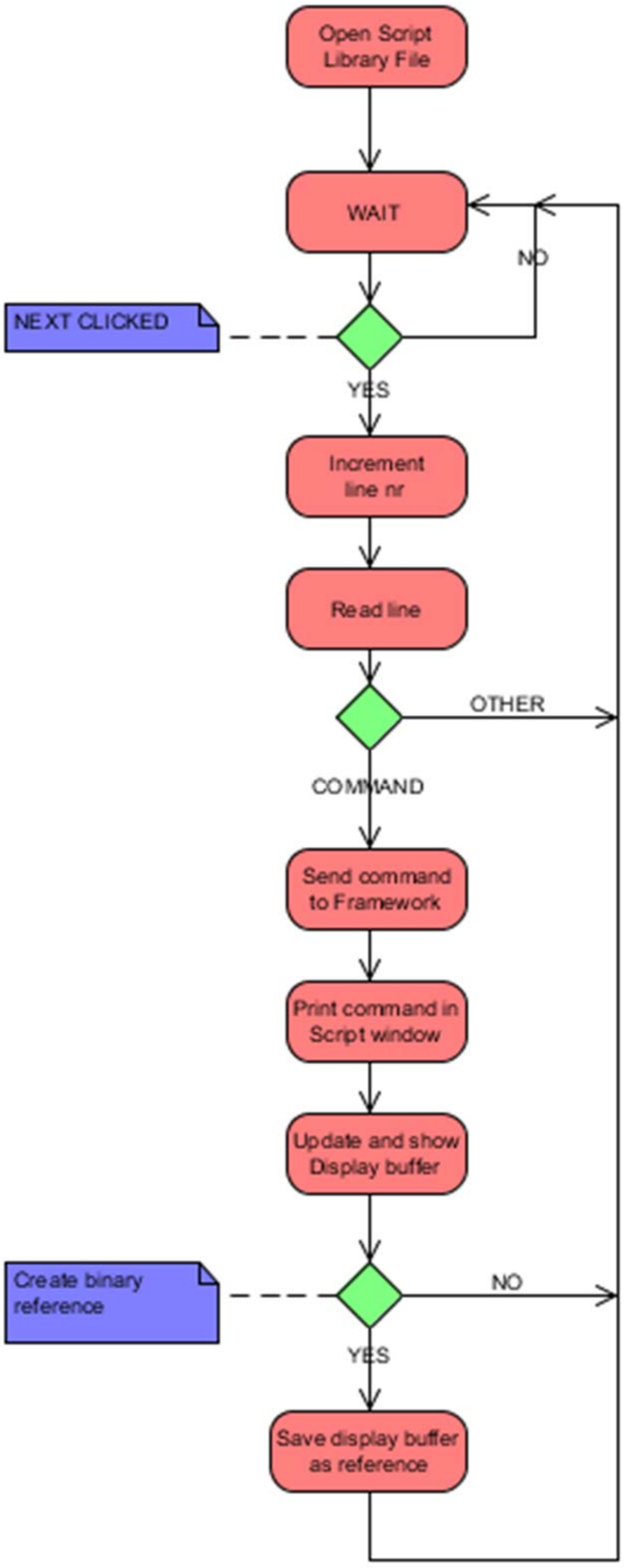

Start
DLL exists
NO
Close application
YES
Script form
NO
GENERATE BUTTON
Test Case field in Script filled out
YES
Generate Test Case Structure
Initialization of environment and test object
Initialize
Set RDI Parameters
Expectations Complete
All data to stubbs or signals to UI
Input
EVALUATE
Save displaybuffer as reference
ALL OTHER
Send command to test environment
Write reference to script
Write command to script
Get new displaybuffer

Figure 5. Program flow when creating a test case

4.4 Replay work flow

The activities diagram below illustrates the work flow of the UISWTestTool when using the replay function



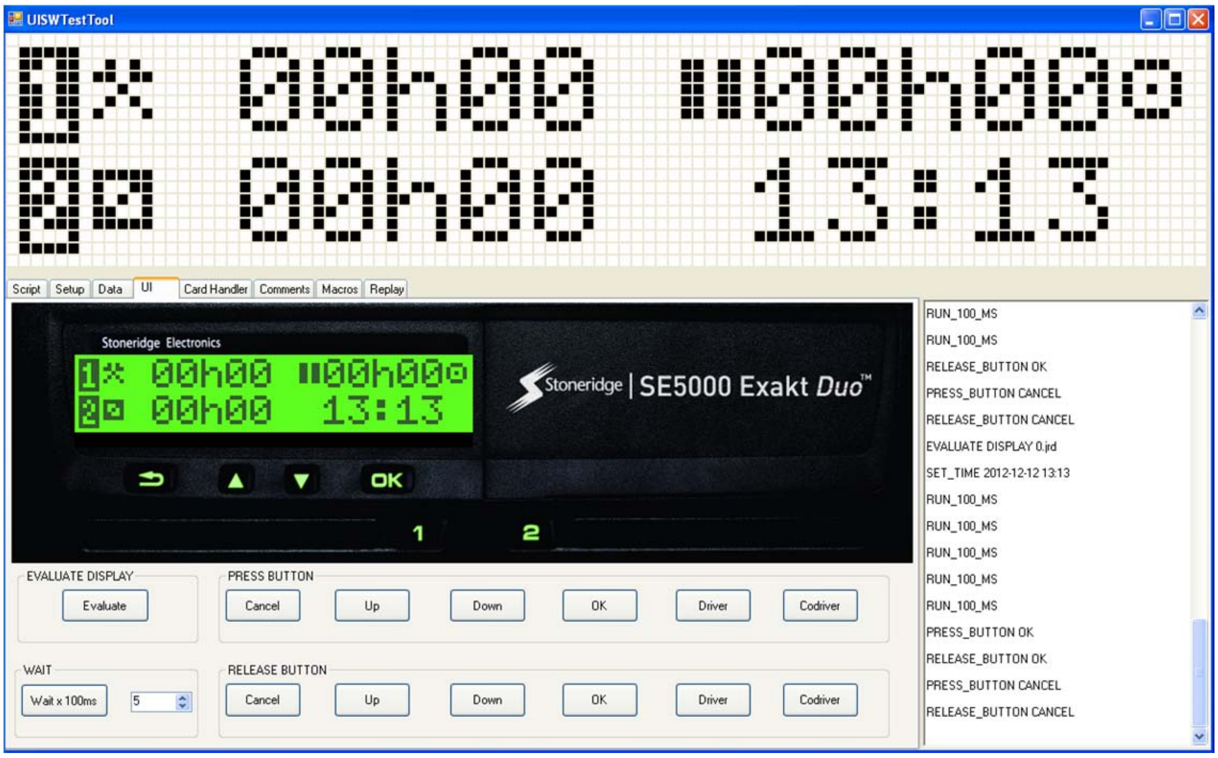
Open Script Library File
WAIT
NO
NEXT CLICKED
YES
Increment line nr
Read line
OTHER
COMMAND
Send command to Framework
Print command in Script window
Update and show Display buffer
Create binary reference
NO
YES
Save display buffer as reference

Figure 6. Program flow when using replay

4.5 Program views

This section shows and describes the different parts and views of the program.

4.5.1 Program layout



```
RUN_100_MS
RUN_100_MS
RELEASE_BUTTON OK
PRESS_BUTTON CANCEL
RELEASE_BUTTON CANCEL
EVALUATE DISPLAY 0.jrd
SET_TIME 2012-12-12 13:13
RUN_100_MS
RUN_100_MS
RUN_100_MS
RUN_100_MS
RUN_100_MS
PRESS_BUTTON OK
RELEASE_BUTTON OK
PRESS_BUTTON CANCEL
RELEASE_BUTTON CANCEL
```

Figure 7. Basic layout of UISWTestTool

The basic design of the program consists of three parts

Top display panel
In the top display panel the content of the display buffer is printed. Each pixel is deliberately made large and contrasted to make it easy to distinguish possible faults.

This display is always visible.

Control tabs
The tabs control the creation of TCs and input to the test environment and script.

Script window
Every command sent to the test environment is also printed in this text box in order to trace your actions without having to locate the script, open it up and see where in the test you currently are.

This text box is always visible.

4.5.2 Script tab

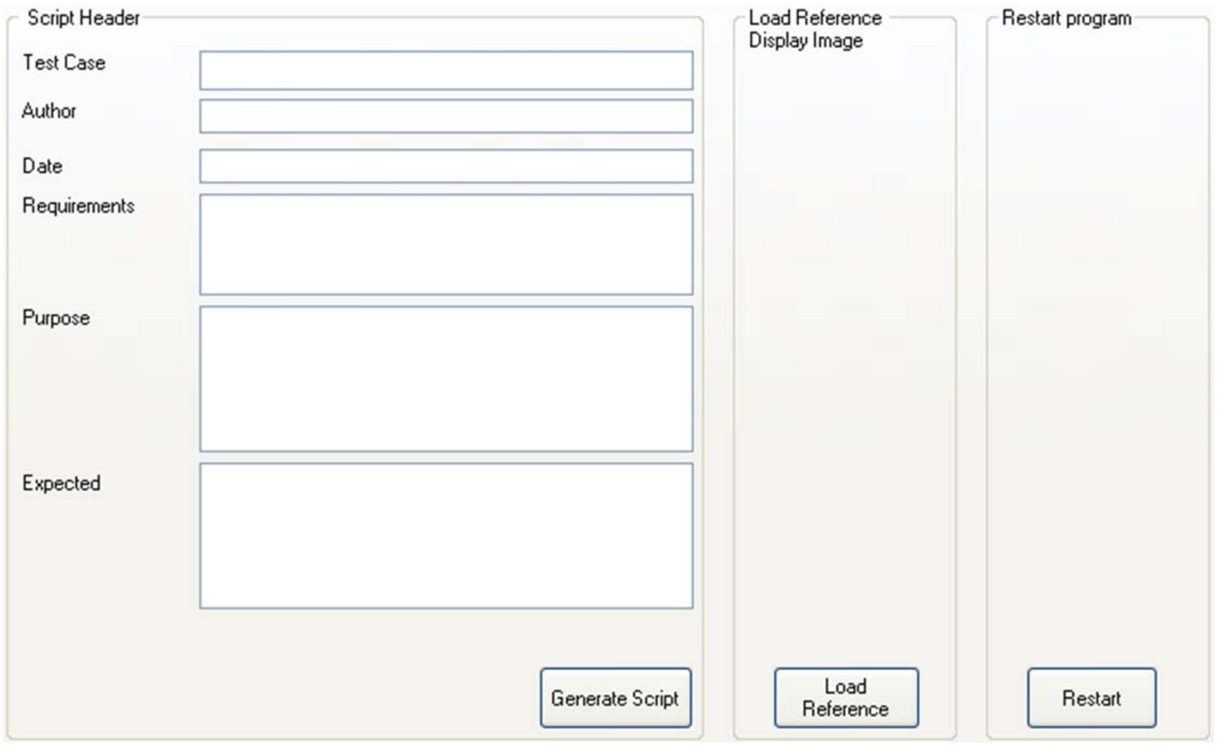


Figure 8. Script tab view in UISWTestTool

Script Header

The information entered here is used to create the TC structure.
Before “Generate Script” is clicked all other tabs are inactive.
When “Generate Script” is clicked a TC is created, the information from the script header fields are entered into the script and the fields are inactivated to prevent overrides.

Table 2. Script Header actions

Input	**Function**
Test Case	Content is used to generate TC folder and name of the script. Is also entered into the script
Author	Is entered into the script
Date	Is entered into the script
Requirements	Is entered into the script
Purpose	Is entered into the script
Expected	Is entered into the script
Generate Script	Generates folder structure for TC, creates the script with the correct name and writes the header information to it

Load Reference Display Image

This button opens up a file browsing dialogue that allows you to locate a binary reference file and display its content in the top display panel.

Table 3. Load Reference Display Image actions

Input	Function
Load Reference	Allows you to open a binary reference file and display it for inspection

Restart program

The button closes the application down, releases the DLL and restarts the application. The main purpose of this button is to speed up the process of creating new TCs.

Table 4. Restart program actions

Input	Function
Restart	Closes the application and then restarts again

4.5.3 Setup tab

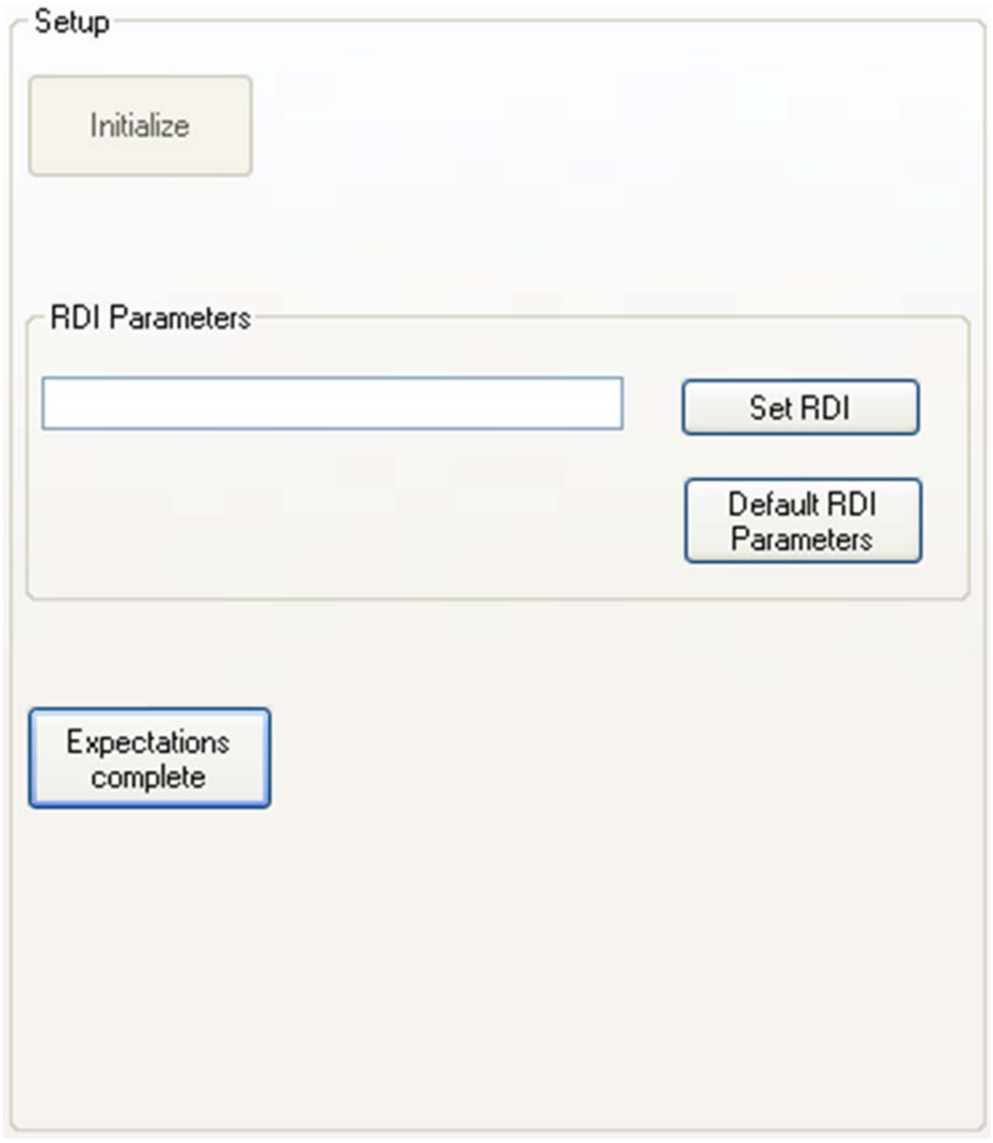


Figure 9. Setup tab view in UISWTestTool

This tab is used to initialize the test object and the test environment. Once initialized the button is inactivated.

You can also set RDI parameters from this tab. Those parameters are set in the stubbed environment that the UI component gets its information from.

You can give the command “Expectations complete” that signals to the VU that it has all the information it needs to start up.

Once “Expectations complete” has been pressed the button is inactivated. A duplicate button in the Macro-tab is also inactivated. All other tabs are now activated.

Table 5. Setup actions

Input	**Function**
Initialize	Initializes the test environment and the test object
Set RDI	Allows you to manually set whichever RDI parameter you want
Default RDI Parameters	Sets some RDI parameters at once.
Expectations complete	Sends the command EXPECTATIONS COMPLETE

4.5.4 Data tab

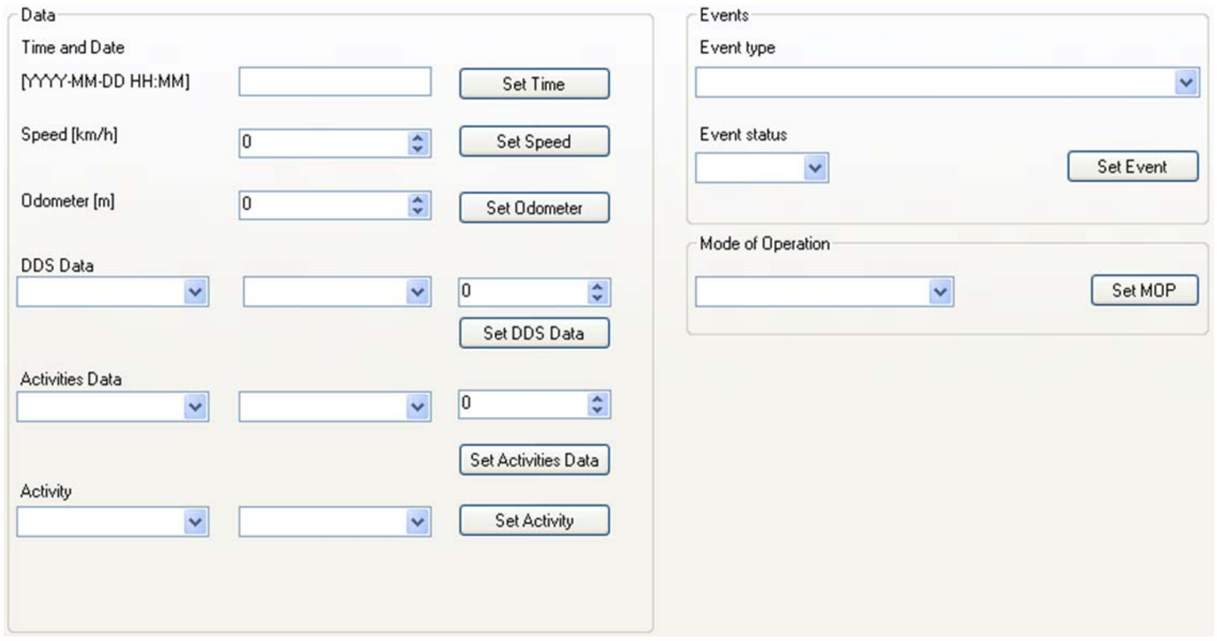


Figure 10. Data tab view in UISWTestTool

Data

You can set data in the stubs here

Table 6. Data actions

Input	Function
Set Time	Sets the time and date
Set Speed	Sets the current speed
Set Odometer	Sets the current odometer value
Set DDS Data	Allows you to set DDS Data for either driver or codriver
Set Parameter	Allows you to set data for either driver or codriver
Set Activity	Allows you to set current activity for either driver or codriver

Events

You can trigger all the different events possible and set them as active or inactive

Table 7. Event actions

Input	Function
Set Event	Trigger event from list and flag them as either active or inactive.

Mode of Operation

You can set the current mode of operation of the vehicle

Table 8. Mode of Operation actions

Input	Function
Set MOP	Sets current mode of operation

4.5.5 User Interface tab

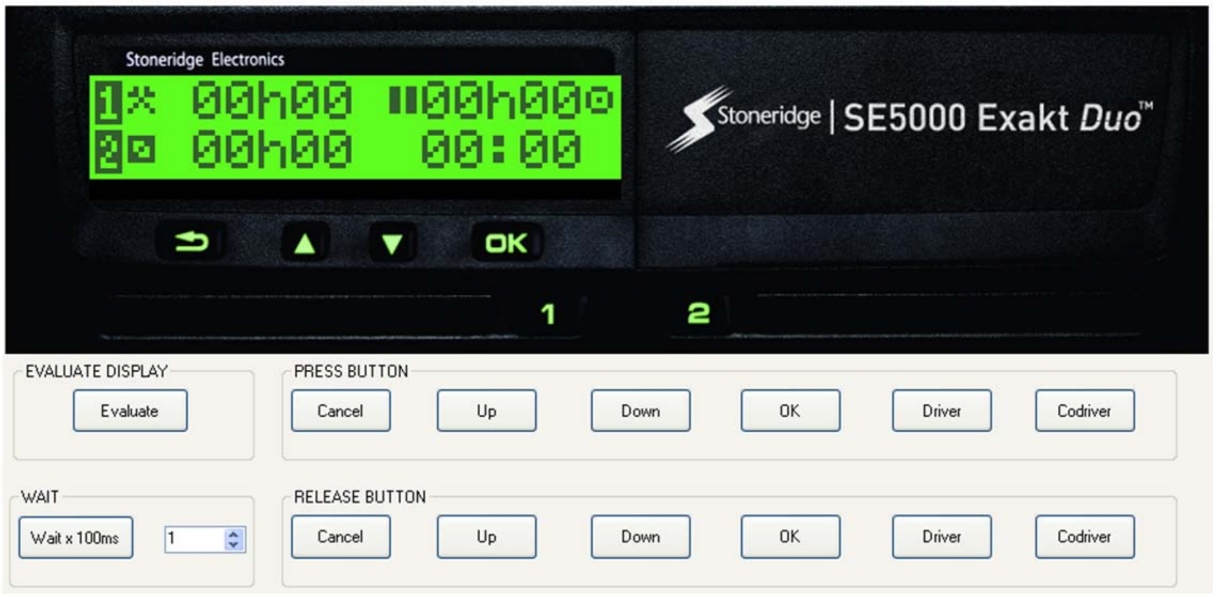


Figure 11. UI tab view in UISWTestTool

This tab gives you full control over what signals are sent and at what times to the test environment.

Table 9. UI advanced control actions

Input	**Function**
Evaluate	Saves the current content of the display buffer as a binary file in the folder dispRef and gives it a name unique for the TC
Wait x 100ms	Sends the command RUN_100MS the given amount of times
Press Cancel	Sends the command PRESS_BUTTON CANCEL
Press Up	Sends the command PRESS_BUTTON UP
Press Down	Sends the command PRESS_BUTTON DOWN
Press OK	Sends the command PRESS_BUTTON OK
Press Driver	Sends the command PRESS_BUTTON DRIVER
Press Codriver	Sends the command PRESS_BUTTON CODRIVER
Release Cancel	Sends the command RELEASE_BUTTON CANCEL
Release Up	Sends the command RELEASE_BUTTON UP
Release Down	Sends the command RELEASE_BUTTON DOWN
Release OK	Sends the command RELEASE_BUTTON OK
Release Driver	Sends the command RELEASE_BUTTON DRIVER
Release Codriver	Sends the command RELEASE_BUTTON CODRIVER

4.5.6 Card handler tab

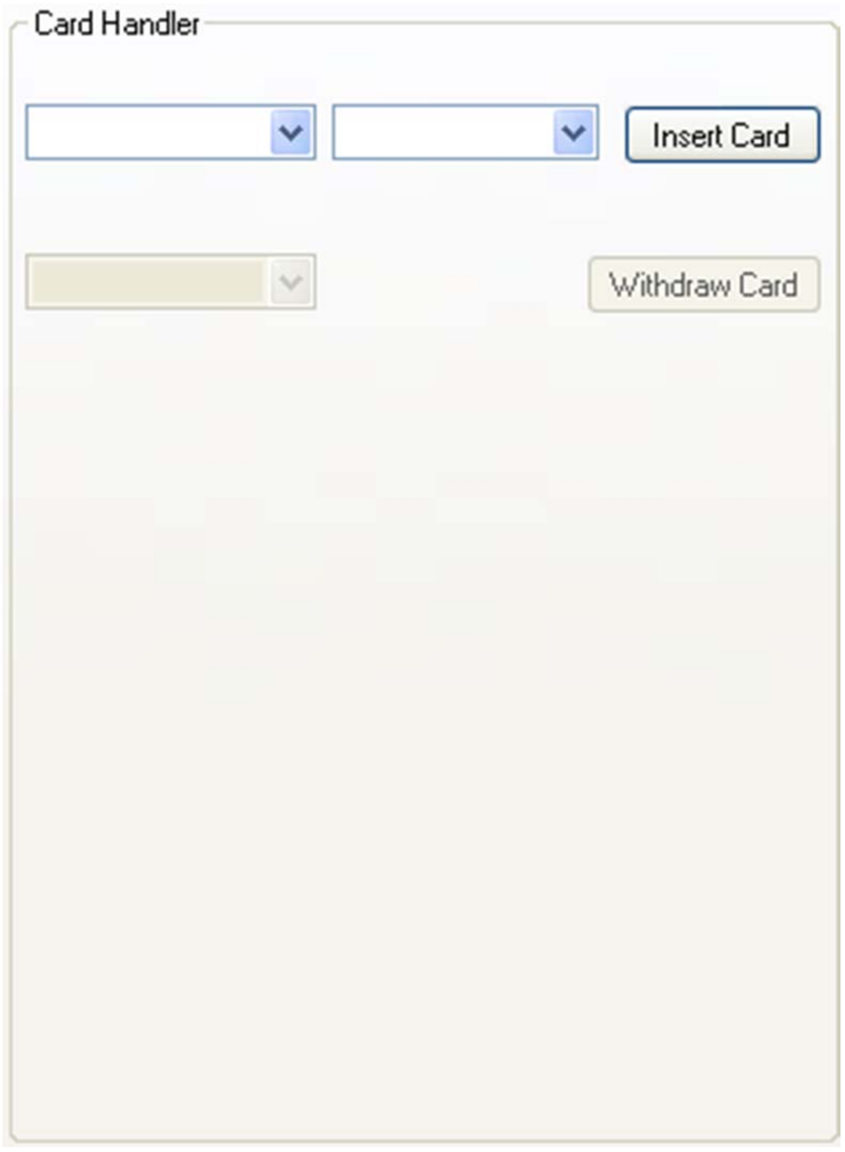


Figure 12. Card handler tab view in UISWTestTool

This tab allows you to handle insertion and withdrawal of smart cards.

Table 10. Card handler actions

Input	**Function**
Insert Card	You choose which slot and what type of card to insert.
Withdraw Card	You choose with slot you want to withdraw

4.5.7 Comments tab

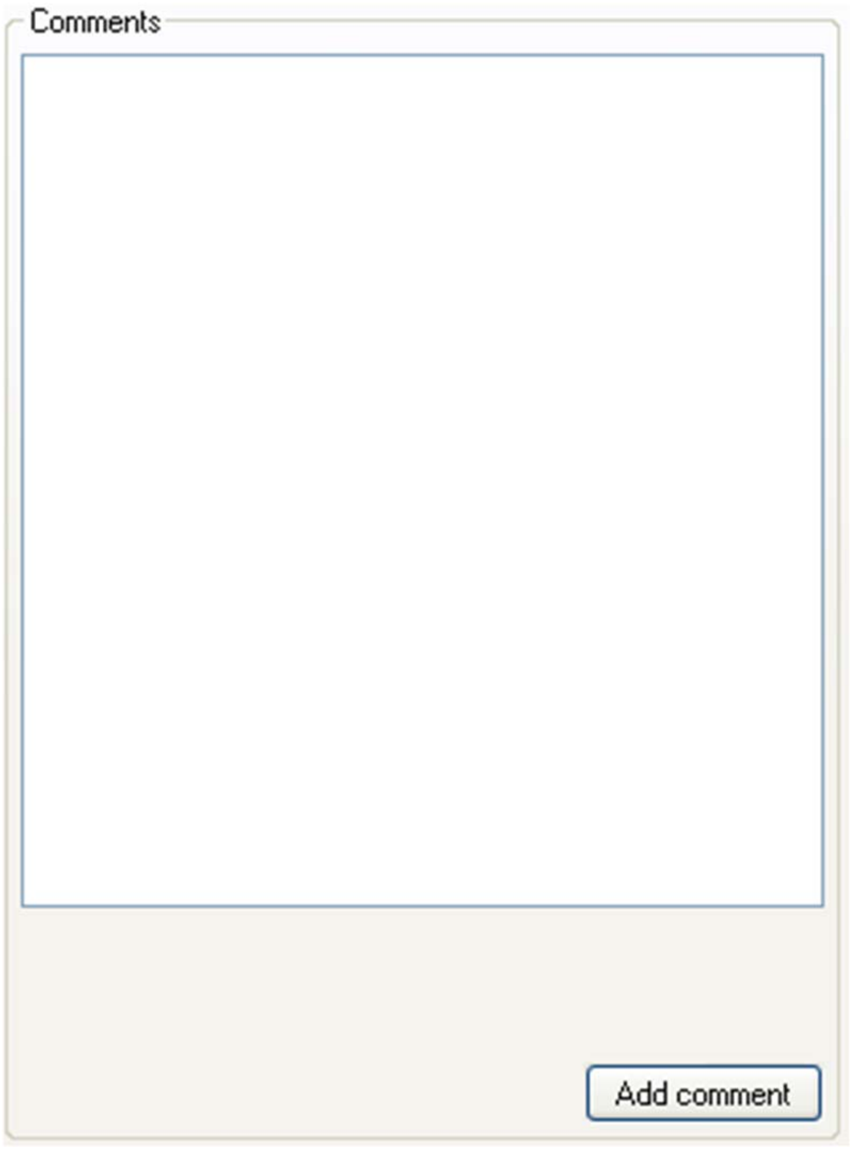


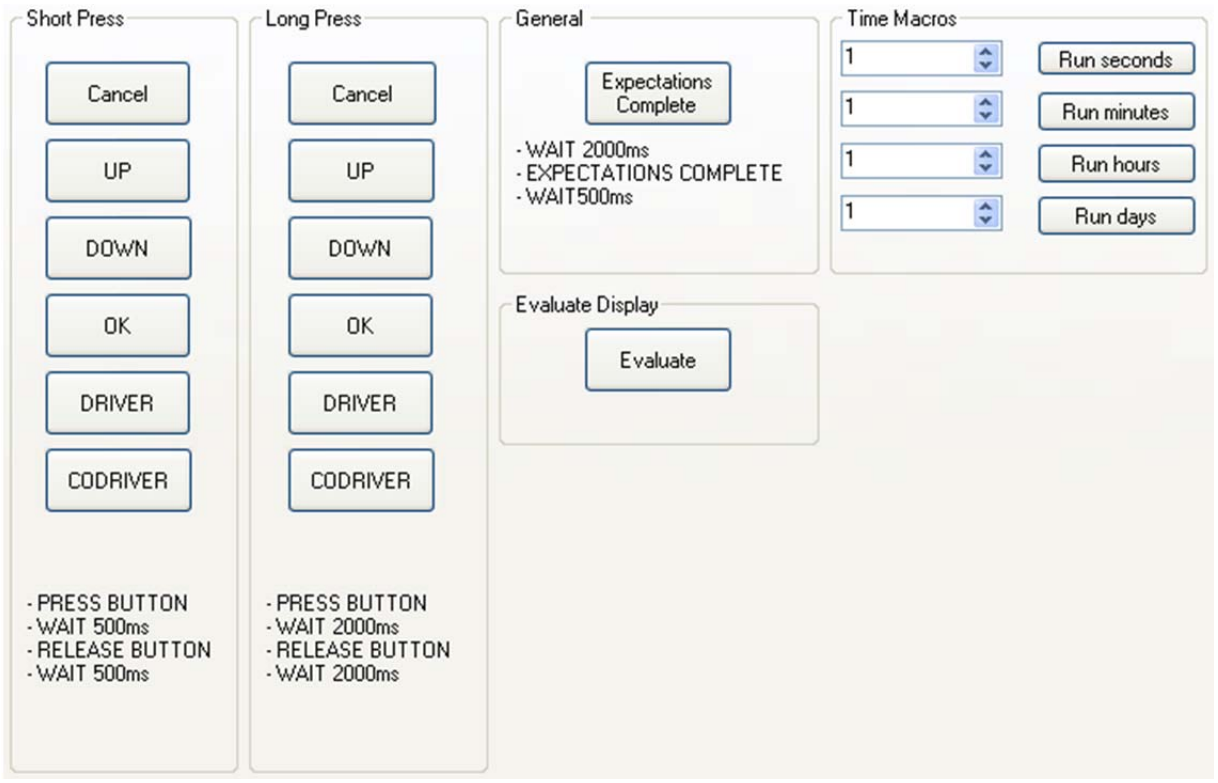
Figure 13. Comments tab view in UISWTestTool

This tab is for writing comments to make it easier and more understandable to read and review the final script.

Table 11. Comment actions

Input	Function
Add Comment	Comments are added to the script file and to the script window

4.5.8 Macros tab



- PRESS BUTTON
- WAIT 500ms
- RELEASE BUTTON
- WAIT 500ms

- PRESS BUTTON
- WAIT 2000ms
- RELEASE BUTTON
- WAIT 2000ms

- WAIT 2000ms
- EXPECTATIONS COMPLETE
- WAIT500ms

Figure 14. Macros tab view in UISWTestTool

This tab has macro buttons designed to make it faster and easier to navigate through the menus and set certain values.

Table 12. Macro button definitions

Definitions		
Expectations Complete - WAIT 2000ms - EXPECTATIONS COMPLETE - WAIT 500ms	**Short Press Buttons** - PRESS_BUTTON - WAIT 500ms - RELEASE BUTTON - WAIT 500ms	**Long Press Buttons** - PRESS_BUTTON - WAIT 2000ms - RELEASE BUTTON - WAIT 2000ms

Table 13. Macro tab actions

Input	**Function**
Evaluate	Duplicate of button in UI tab to make quick evaluation easier. Saves the current content of the display buffer as a binary file in the folder dispRef and gives it a name unique for the TC
Expectations Complete	See definition above. Only active if “Expectations Complete”-button in Setup tab has not been pressed. If this button is pressed the “Expectations Complete”-button in Setup tab is inactivated
Short Press Button Macro CANCEL	Cancel button, see short press definition above
Short Press Button	Up button, see short press definition above

Macro UP	
Short Press Button Macro DOWN	Down button, see short press definition above
Short Press Button Macro OK	OK button, see short press definition above
Short Press Button Macro DRIVER	Driver button, see short press definition above
Short Press Button Macro CODRIVER	Codriver button, see short press definition above
Long Press Button Macro CANCEL	Cancel button, see long press definition above
Long Press Button Macro UP	Up button, see long press definition above
Long Press Button Macro DOWN	Down button, see long press definition above
Long Press Button Macro OK	OK button, see long press definition above
Long Press Button Macro DRIVER	Driver button, see long press definition above
Long Press Button Macro CODRIVER	Codriver button, see long press definition above
Run Seconds	Simulates elapsed time in seconds. Amount specified in field.
Run Minutes	Simulates elapsed time in minutes. Amount specified in field.
Run Hours	Simulates elapsed time in hours. Amount specified in field.
Run Days	Simulates elapsed time in days. Amount specified in field.

4.5.9 Replay tab

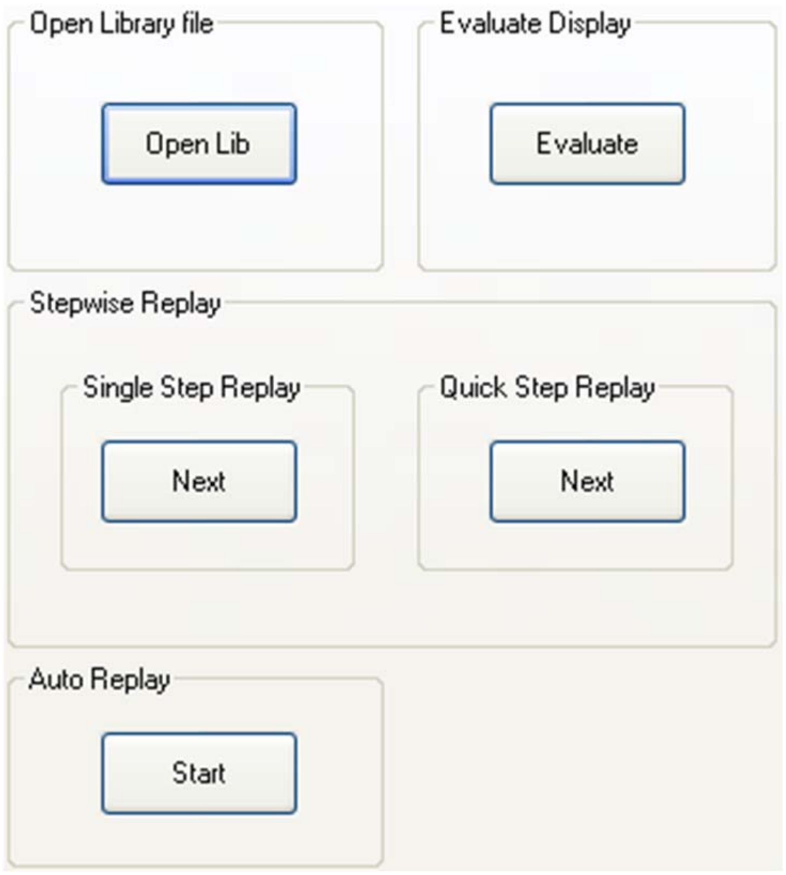


Figure 15. Replay tab view in UISWTestTool

In this tab you can open up a script and step through it in different ways and save the content as a reference file. It is mainly intended to be used to create permutations of a specific script. One example is stepping through all menus in several different languages. Create a new TC folder, rename it and the script and delete the reference binaries. You can then step through the script, evaluating the display and saving the new references in the new folder.

Table 14. Replay actions

Input	**Function**
Open Lib	Opens up a file browsing dialog that allows you to choose script to step through
Evaluate	Saves content of display buffer in folder dispRef of chosen script
Single Step Next	Steps through the script line by line sending the commands to the UI component. Stops after each command.
Quick Step Next	Steps through the script line by line sending the commands to the UI component. Stops after each evaluate.
Auto Replay Start	Automatically plays through the entire script

5. Discussion and Conclusion

This method of testing the software of the UI component is very effective and worth further development.

Since tests run on hosts it can never replace tests run on system verification level on target. But it can still help a great deal. It shortens the feedback loop for SW developers since they can see the effects of changes directly instead of having to wait for reports from the system verification engineers.

Once the test case is created and added to the test regression suite it is fully automated. Since it is fully automated all tests can be run every time instead of just a select few. This results in much greater test coverage at practically no additional cost.

Since tests are automated once they are created, developers no longer need to spend time manually testing them after each greater software change. This results in a much more efficient use of time.

Saving time on manual testing should result in higher productivity and better competitiveness for the company. This in turn should result in more development of better products at better prices.

What would happen if the display changes in a future version of the tachograph? As long as the script language is the same it is possible to open up an existing test case through the Replay tab and step through it and overwrite the old binary references with the new content of the display buffer.

6. Recommendations and future work

6.1 Recommendations

It seems appropriate that this program is further developed and refined. It will take quite some time to produce an extensive library of test cases but in the long run it will probably save a lot of time and money.

Testing the UI component has been done on system verification level with system verification requirements. There needs to be specific requirements for testing on SW level to cover the large amount of new TCs needed.

Deciding on a good TC structure is also important so that TCs can be more easily divided into smaller groups and better organized to keep track of what has been tested and what is yet to be tested.

6.2 Future work

There are many functions and features that have not yet been implemented in the UISWTestTool. One of these functions is loop control. It would be very useful to be able to play a script within a script. This is already supported by the UI TE but has not been implemented in this version of the program.

Further development of the UI TE would improve the usefulness of the UISWTestTool and this test methodology even further.

The following implementations to the TE should be considered

- Evaluate Manual Entries and the data it returns
- Evaluate Specific Conditions
- Evaluate changes of parameters via UI
- Support withdrawal of smart cards
- Support insertion of all card types
- Fully test DDS features
- Modify response from TE to determine if input is correct
- Save mismatching display buffer data when mismatch occurs
- Reinitiate a fresh version of UI component after each TC run

7. References

[1] http://en.wikipedia.org/wiki/Tachograph, 2012-09, Tachograph

[2] http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31985R3821:sv:HTML, 2012-09, function certificate regulation

[3] http://en.wikipedia.org/wiki/C_Sharp_(programming_language), 2012-09, C#

[4] http://msdn.microsoft.com/en-us/library/gg145045.aspx, 2012-09, MSDN .NET Library

[5] http://thenewboston.org/list.php?cat=15, 2012, C# tutorial

[6] http://en.wikipedia.org/wiki/SharpDevelop, 2012-09, SharpDevelop

[7] http://en.wikipedia.org/wiki/Notepad%2B%2B, 2012-09, Notepad++

[8] http://en.wikipedia.org/wiki/UMLet, 2012-09, UMLet

[9] http://en.wikipedia.org/wiki/GIMP, 2012-09, Gimp

[10] http://en.wikipedia.org/wiki/Microsoft_Office_2010, 2012-09, Microsoft Office Word

TRITA-ICT-EX-2012:269