CHALMERS
UNIVERSITY OF TECHNOLOGY

Development of a Vehicle User Interface Testing Platform

Master's thesis in Systems, Control and Mechatronics
Thesis EX081/2016

Jonas Karlsson

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

CHALMERS
UNIVERSITY OF TECHNOLOGY

Abstract

A software framework has been designed and coded in order to provide a platform for testing vehicle user interfaces and the properties of the closed loop controlled system that the driver and vehicle makes up and how the user interface influences the driver. The report describes the software and how it works and it provides a reference for how to use it. The test results show some of the strengths and flaws of the software. The report also provides a reference of how the software can be improved in a future project.

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

CHALMERS
UNIVERSITY OF TECHNOLOGY

Acknowledgements

I have worked hard and put much effort into this project. However, it wouldn't have been possible without the support of my family, friends, co-workers on the Chalmers vehicle simulator and some of the staff at Chalmers.

I would like to thank Jonas Sjöberg, professor at the department of signals and systems at Chalmers and project manager for this project for having much patience with me and helping me during the project.

Alberto Morando helped me much with the Chalmers vehicle simulator when he did his thesis on it, parallel to mine.

Bruno Augusto, employee of VTI, was a great help when the Chalmers vehicle simulator had issues that needed fixing.

My friend Josefine Olsson kept coming with me to test the project at the Chalmers vehicle simulator despite numerous failed attempts for various reasons.

My other friend Ludvig Lam helped me test the project and let me borrow his gaming steering wheel and computer for some tests when I didn't have access to the simulator.

Last but not least, my mother who has always been there for me.

Thank you all for making this possible.

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Contents

Introduction 3
- Background 3
- Contributions 3
- Vehicle Simulation and this Software 3
- The Chalmers Vehicle Simulator 3

Specification 5

Design of the software 6
- The software 7
  - Tasks 8
  - Alarms 9
  - Scripts 9
  - Variables 10
  - Conditions 10
- MainForm 11
  - The message filter 11
- UDPTools 11
- UDPFaker 11
- Utilities 11
- VariableManager 12
- TaskManager 12
  - The script reading process 14
  - Task script tags 16
- AlarmManager 17
  - The script reading process 18
  - Alarm script tags 21
- SoundManager 23
  - Using the Windows Multimedia API 23
- ErrorHandler 23

Testing 24
- Live tests in the Chalmers simulator 24
- The accuracy of the timing in the logs 30

Discussion 33
- The software and live tests in the Chalmers Simulator 33
- The timing of the software 33
- The structure of the TaskManager vs. the AlarmManager 34
- Why a custom scripting language as opposed to a pre-existing one? 34

Conclusion 36
- The software 36
- Timing 36

Possible improvements 37

Bibliography 38

Appendix 41

CHALMERS
UNIVERSITY OF TECHNOLOGY

Appendix Contents

Appendix - Word List **43**
Appendix - Test results **44**
Live tests in the Chalmers simulator 44
Test 3, log 1 44
Test 3, log 2 45
Core Matlab script 49
Script to extract event data 50
Script to extract speed and lane data 50
Script to plot data 50
Software member reference **53**
Members of MainForm 53
Members of UDPTools 56
Members of UDPFaker 57
Members of Utilities 59
Members of VariableManager 63
Members of TaskManager 64
Members of AlarmManager 70
Members of SoundManager 75
Members of ErrorHandler 79
Appendix - The software code **80**
MainForm 80
UDPTools 86
UDP Faker 92
Utilities 94
VariableManager 101
TaskManager 105
AlarmManager 119
SoundManager 138
ErrorHandler 142

Introduction

Background

The software was requested to be made in order to have a platform for testing user interfaces. One of the methods used for testing user interfaces when this project was started was to create each new user interface as a Flash application or similar. This means that it's hard, if not impossible, to get all the features required for a thorough test.
A platform for testing user interfaces can in itself contain the features for testing while only the user interface itself and the scenario has to be built. This makes it less time consuming, easier and more accessible to people with less programming knowledge to be able to make testable user interfaces.

Contributions

The software was made in the shape of a framework. The software uses Visual Basic .Net as its programming language and provides a set of modules that handles the testing and network communication of the user interface.
Tasks and alarms are ways for the designer to provide interaction for the test driver for the test. They are made by creating scripts that the software reads, meaning that the same user interface executable can have any number of sets of different tasks and alarms that are interchangeable.
The software outputs logs of the events happening in the software as well as when and where the test driver clicked on the screen.
It features a networking module that can connect to any simulator that is able to output information via UDP. This makes it possible for the user interface to read the accessible variables in a simulator, such as the speed of the vehicle, and use it in the interface as well as in the tasks and alarms.

Vehicle Simulation and this Software

The driver and the vehicle forms a closed loop controlled system, where the driver takes the role as the controller and the vehicle is the system to be controlled. The properties of this system are crucial for the safety and quality of the performance of the system. Hence it is important to evaluate and understand these properties and take them into account already at the design phase when constructing a vehicle and its interface to the driver.
This project develops software that, in conjunction with a simulator, provides a toolset to design and run tests in a safe environment with the purpose to measure the properties of the driver and to measure the quality of the part of the interface that is the touch screen and how it influences the driving performance.
The developed software is demonstrated and verified on smaller experiments with a test driver. The results are given in the section Testing (p. 24) and discussed in the section Discussion (p. 33).

The Chalmers Vehicle Simulator

While the software has been designed to make it easy to connect to different simulators, it was designed for, and tested on the Chalmers Vehicle Simulator.
The Chalmers Vehicle Simulator is a moving base simulator. That is, the cabin moves with feedback from the simulation to simulate acceleration. The cabin consists of the driver side

quarter of a Volvo S80 mounted in a metal frame, with a projector screen in the front, displaying the graphics of the simulation. The side view mirror on the driver side has been replaced by a screen that can show the simulated side view mirror camera angle of the simulation. The CAN bus of the Volvo is connected to the simulator, allowing for control of the dashboard.
The vehicle simulator has been a platform for several projects that has built upon its functionality or used it for research. This project adds the possibility of placing a touch screen anywhere in the simulator.
(Sjöberg, Fredriksson, & Falcone, 2013)

Specification

This piece of software is a framework for creating applications with which operators are able to run tests, mainly focused on vehicle touch screen user interfaces, with test drivers. The framework provides functionality to the testing platform by giving the designer access to pre-programmed functions and modules in Visual Basic .Net:

- Triggering and scripting of events
  - Alarms
  - Tasks
- Tracking user inputs
  - Screen position coordinates
  - Time
  - What object was clicked
- Tracking the timing between events, arbitrary points in time and/or user inputs
- Playing multiple audio sequences at the same time
- Logging
  - System information
  - Event information
  - Arbitrary variables or other information
- Networking between an external vehicle simulator and the software
  - Wireless UDP connection via IP addressing through a LAN or a WAN
  - Pre-programmed functions to automatically create and receive network packets of selected arbitrary numeric variables
- Messages identifying errors made in scripts

Design of the software

The language used for the software is Visual Basic .Net and the programming was done with Visual Studio from Microsoft.

The code is divided into several different modules that each have a different objective. In this section follows explanations of central concepts in the software as well as a list of the modules and a short explanation of their objective. A reference list of the members of the classes can be found in the appendix section Software member reference (p. 53), where the members are explained in detail.

For writing the code for the software, the Microsoft Developer Network, or MSDN, was accessed multiple times in order to research the way the programming language and its functions work. Specifically the .NET framework class library and documentation was used. (MSDN, p. Documentation Library)(MSDN, p. .NET framework class library)

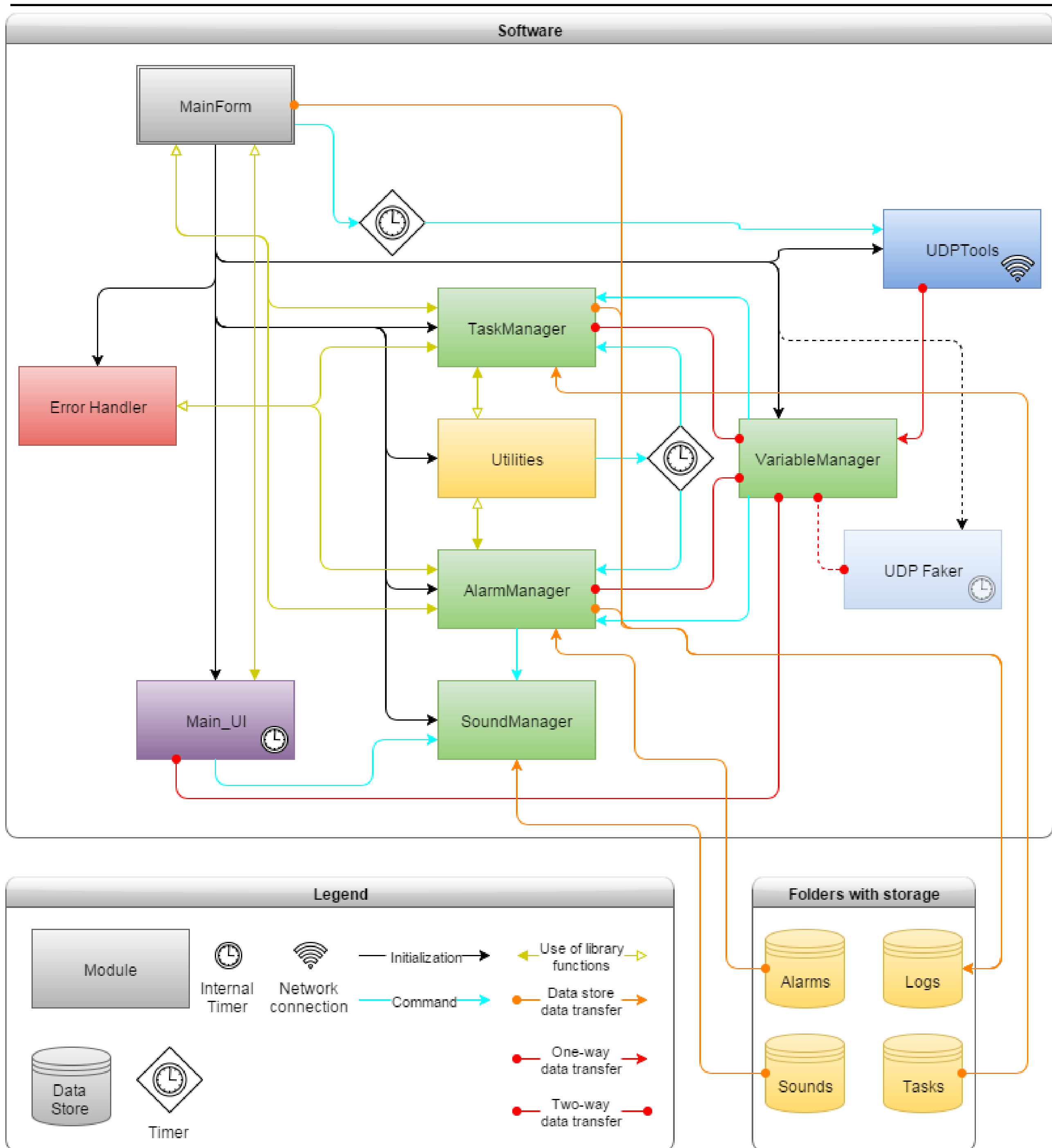

Figure 1: The software and how the different modules interact

The software

The software is designed as a framework for designers to design a user interface in.
The designer uses the modules within the framework in order to do testing of the efficiency of the human machine interaction of the user interface.
There exists a module for creating scripted tasks for the test driver to do while driving in a simulator. Tasks are elaborated on in the Tasks section (p. 8), but in short, they're a way for the designer to give an objective to the driver and see if and when the driver fails or succeeds.

There is also a module for creating scripted alarms. Alarms are somewhat similar to tasks but are rather ways for the designer to notify the driver or have them react to things and are elaborated on in the Alarms section (p. 9).
There is a system in place for registering any mouse click (or touch on the touch panel) the driver does while testing. The position and time of them as well as what control, or part of the software, was clicked is logged. This ties into the tasks and alarms in that the clicks can be used as a trigger.
The software has a UDP module that can communicate with any simulator that has the possibility to send (and receive) data via UDP. When the UDP module has been modified to use the same packet structure as the simulator, data can be sent between the software and the simulator to be used for the test. An example would be to display the speed of the vehicle on the user interface and use the speed to trigger alarms and tasks. On the other hand, the software could for example, if the simulator programmed to accept the command, turn on or off the traffic in the simulation based on circumstances in the software. See the UDPTools section (p. 11) for more information.
Log entries are made whenever a mouse click is registered or an alarm or task changes its status. There are also system information, such as error messages or startup log entries made. Custom logs can also be made to for example log the speed of the vehicle or any other accessible variable.

Tasks

A task is a scripted objective for the driver to do while testing.
A task has 3 different events that changes the status of the task. The task can be triggered, failed and ended.
Conditions are defined by the designer to determine when a task is triggered, failed or ended. The conditions are elaborated on in the Conditions section (p. 10), but can be for example that a certain control in the user interface is clicked or a certain variable is changed to a certain value.
The trigger event starts the task and a task can only be failed or ended after it has been triggered. When the task has been triggered, it can either be failed or ended afterwards, whichever happens first. The failing and ending events are optional, but the task will remain active until one or the other happens and cannot be reactivated until then.

- In order to trigger the task, all of the conditions for triggering it need to be fulfilled.
- In order to fail the task, any one of the conditions for failing it need to be fulfilled.
- In order to end the task, or complete it, all the conditions for ending it need to be fulfilled.

The tasks can also modify variables when they are triggered, failed or ended. For example when the conditions to trigger the task has been met, the task is triggered and when that happens, the example variable "state" could be set to the value 1 by the task. The other tasks can then trigger their different events when the variable "state" reaches a certain value. Parts of the user interface can also change depending on the value of the variable.
Any variable in the VariableManager can be changed or used in this way. Variables will be elaborated on further in the Variables section (p. 10).
A log entry is made when a task is loaded, triggered, failed or ended.
See the

TaskManager section (p. 12) for a reference list of the script tags that can be used for a task script. Scripts are elaborated upon in the Scripts section (p. 9).

Alarms

An alarm is a scripted event for the driver to react to or receive information through while testing.
An alarm has 3 different events that changes the status of the alarm. The alarm can be triggered, confirmed and handled.
Conditions are defined by the designer to determine when an alarm is triggered, confirmed or handled. The conditions are elaborated on in the Conditions section (p. 10), but can be for example that the speed variable, if available, has reached a certain threshold or that a certain time has passed.
The trigger event starts the alarm. Then the confirm event is meant to be used for the driver to tell the software that the existence of the alarm is acknowledged. The handling event is meant to be used to deal with the reason for the alarm.
The confirmation and handling events are optional, but if used, the events must be done in a sequence where the alarm is first triggered, then confirmed and lastly handled.
If confirmation and handling aren't used, the alarm stops being active right after it has been triggered and can then be used again.

- In order to trigger the alarm, one of the conditions for triggering it need to be fulfilled.
- In order to confirm the alarm, one of the conditions for confirming it need to be fulfilled.
- In order to handle the alarm, all of the conditions for handling it need to be fulfilled.

A sound loaded in the SoundManager can be scripted to play when an alarm triggers. There is also the option to show a message box with text about the alarm.
The alarms can also modify variables when they are triggered, confirmed or handled. For example when the conditions to trigger the alarm has been met, the alarm is triggered and when that happens, the variable "fails" could be incremented by one. Parts of the user interface can also change depending on the value of the variable.
Any variable in the VariableManager can be changed or used in this way. Variables will be elaborated on further in the Variables section (p. 10).
A log entry is made when an alarm is loaded, triggered, confirmed or handled.
See the AlarmManager section (p. 17) for a reference list of the script tags that can be used for a task script. Scripts are elaborated upon in the Scripts section (p. 9).

Scripts

Scripts are external files that control some of the behavior of the software.
The same user interface can have different script files loaded for different purposes. The software is built with the idea that scripts for tasks and alarms control what is tested in the user interface while the user interface only provides the functionality a user interface is meant to. Some modifications may still be needed within the user interface however. But how the test is done programmatically is ultimately the designer's decision.
Tasks and alarms can be scripted, as described in the Tasks (p. 8) and Alarms (p. 9) sections. In order to do this a series of tags are used to describe what the task or alarm is supposed to do. A tag is a piece of text within square brackets, for example [ThisIsATag].

Start tags and end tags enclose different categories in the script, for example the info category or the fail category of a task or the confirm category of an alarm.
Tags within the categories describe the related information, conditions or commands.
A tag sometimes needs different sections of information in a single tag, for example condition tags. In the condition tags, the first section needs to describe what type of condition it is while the second and possibly third section describes the details of that type of condition. In order to do this, a colon separates different sections of the tag, for example [section1:section2:section3].
Comments can be written in the script to explain what the script does to anyone reading it. Comments are prefixed with the character ' and are not read by the software.
Empty lines in the script are not read by the software.
The following is an example of a category, info, in a script where a tag describes the name of the script:

```
[info]
        [name:Script1] 'This is a comment
[/info]
'This is also a comment [and this tag is not read because it's in the comment]
```

The example has 3 tags, a start tag for info, a tag inside the info category describing the name and the end tag for info.
A reference list for what tags can be used how and where can be found in the TaskManager (p. 12) and the AlarmManager (p. 17) sections for tasks and alarms respectively.

Variables

The VariableManager stores and handles a set of variables defined by the designer. The difference between these variables and the normal variables defined in the rest of the code is that these variables each are a set of key-value pairs in a list. The key is the name of the variable and the value is the value it's storing.
This system is in place so that scripted events are able to use named variables that are searchable by name during runtime. This means that the designer can target variables to modify or read by giving a command or condition in the script and naming the variable as the target for it.
The variables sent or received via UDP are also variables stored this way.
See the VariableManager section (p. 12) for more information of how it works.

Conditions

Conditions are requirements for triggering different events in the scripts and the alarms. They are written as tags in the scripts in the categories they are supposed to generate requirements for triggering. For example a condition written in the confirmation category of an alarm means that it generates a requirement for confirming the alarm.
There are different kinds of conditions that generate different kinds of requirements.
A variable condition means that a variable needs to have a certain value depending on the condition. For example the variable condition varLess means that the variable needs to be less than, but not equal to the given value in the condition.
A click condition means that a certain control in the software needs to be clicked for the event

to trigger.
A timer condition means that the event can trigger after a certain time has passed.

MainForm

MainForm is the starting module and as such, is created, loaded and shown as the software starts. It is a form and its main objective is to initiate the other modules when the software starts up. MainForm also handles the detection of the click location and detecting what objects are clicked. The module also hosts functions for checking what objects are contained in a form and if an object with a certain name exists or not. It gives the read and write commands to the UDPTools module, with the help of a timer, in order to handle data transfer via UDP. MainForm contains the code that handles logging of data as well.

The message filter

In order to be able to detect when and where the user interface is clicked using the mouse or the touchpad, this module implements a message filter. The message filter interrupts the Windows messages in the software and lets the software modify or remove them before optionally returning them.
This gives the software access to the messages sent when anything on the user interface is clicked. The position of the click and the object that was clicked can be extracted from the message and forwarded to other parts of the software.
In order to implement the message filter, the module has to implement the IMessageFilter interface. A message filter must be added using the method Application.AddMessageFilter. In order to receive the messages, the module has to implement the PreFilterMessage interface function. (MSDN, p. IMessageFilter Interface)

UDPTools

UDPTools is the module that hosts methods that handle the UDP connection. The module works like a library in that it doesn't do anything by itself. The timer that triggers the sending and receiving of the UDP data is in the MainForm and that calls the methods sendUDP and receiveUDP in this module. The sendUDP and receiveUDP uses previously set up UdpClient type classes as input parameters in order to connect. In this case, those are named publisher and subscriber and are hosted by the MainForm module as class variables. The publisher deals with sending UDP and the subscriber deals with receiving UDP.

UDPFaker

This module fakes a UDP connection by directly overwriting the variables in the VariableManager as if a connecting device would have sent variables to this software. This module is for testing the scripts and the UI by setting different variable values, which will trigger the scripts and UI accordingly as if a device would have given this software the variables via UDP.

Utilities

This module works as a library for methods and classes and hosts timers that are used by other modules. It contains methods that are used by the TaskManager and AlarmManager in order to load and interpret the scripts for the software and find errors in them. This module also contains methods to check if conditions for the tasks and alarms are properly met. A class named condition is hosted by the module and is used by the TaskManager and AlarmManager

to store information about conditions.
Many of the methods require a so called taglist as an input parameter. A taglist is an array of an arbitrary number of strings. These strings comes from when other parts of the software reads the scripts and each array is a single line, or tag, in a script. Each string in the array is a single keyword on the line. Those methods that require a taglist input are used in order to tell the software that a certain input is expected and the methods will validate that the input is what the designer expects it to be. An example of this is the readTextFromTaglist method that will validate that the tag in question indeed is a tag that contains a text string.
The part of the taglist called "identifier" is a value identifying the type of tag. It can for example be the type of condition or command. The identifier is used in the AlarmManager and TaskManager to filter the tags and identify them before the tag is validated, in order to know which type of validation is required.

VariableManager

This module stores and keeps track of the variables used by the TaskManager and AlarmManager and the helper methods in the Utility module. These are also the variables that can be sent via UDP. The difference between these variables and variables that you declare normally is that these are stored together with a string name. This means that the variables are searchable by their name in run-time.
The module has methods for creating, reading and manipulating the stored variables.
The variables are stored in a list called variableList.

TaskManager

This module reads, stores and handles the scripted tasks that the driver is to perform during the software use.
The module is initiated by creating a new instance of it and supplying it with a task script. One instance is created per script and this is done in the MainForm when MainForm is loaded. This process of loading the scripts is done in the constructor of the module. Each of the instances of the module are placed in a list in the MainForm to keep track of them.
When the module is created and supplied with the script, it reads the script by using internal methods and methods from the Utility module.
When done, based on the settings it can use a timer called UpdateTimer in the Utilities module to update the tasks or it updates the tasks when a variable is changed in the VariableManager or both. This is controlled by the variables updateWithTimer and updateOnDemand located in the TaskManager. These variables are public and can either be assigned in the code in the TaskManager at design-time or from other locations in the software during run-time.
The updating lets the task check if it's started, progressed, failed or ended based on the state of the variables.
There are also timer based conditions, which are handled via the UpdateTimer calling the method UpdateTimerCounters. This method updates the timer counters in the tasks with the duration progressed from the start of the timer. Once the timer reaches its target time, this method either instantly fails the task, if the timer condition was for failing or sets boolean flag variables accordingly if the timer was for ending or starting the task.
Clicking a control in the software UI, including the forms themselves, calls the method reportClick in the TaskManager, from the MainForm. This method lets the MainForm communicate to the TaskManager what control was clicked, which updates the state of the

tasks based on their click conditions, if any.
When the state is updated and all necessary conditions hold for either of the three cases of starting, failing or ending a task, the StartTask, FailTask or EndTask methods are called to respectively start, fail or end the task.

The script reading process

The scripts are read by the module in the constructor method when an instance of the module is created.

The text below, for the constructor method, is the same as the text for the readAlarm method in the AlarmManager in the script reading process.

The constructor method starts by reading the script file with the given file name from the input parameter into a temporary memory of the software as an array of raw text where each entry in the array is a line of text corresponding to the lines in the script file. It then loops through the rows of the script, reading them individually until it reaches the end of the file, in which case it stops looping and nothing more happens.

When it finds a row that is not a comment and is not empty, it checks if the first character is a "[", a beginning square bracket. This signifies that the line has a tag. If the line doesn't start with a square bracket, but any other character that isn't the comment sign, "'", the row is invalid and an error, ShowInitialSignError, is generated and the software closes.

If the line however does start with a square bracket, the software continues processing the line by checking for the position of the first ending square bracket "]". If it doesn't find one, the line is invalid and an error, ShowNoEndSignError, is generated and the software closes.

If an ending square bracket is found, the software continues and splits the string within the square brackets at the positions where there's a colon sign ":", putting the parts of the string into an array with one part per entry and checks that the number of entries are between and including 1 and 3. If the number of entries are incorrect, an error, ShowParamNumError, is generated and the software closes.

If the number of entries is correct, the software checks if the current category is none. If it is, the expected tag should be a valid category, for example info, making the tag [info]. If the current category is none and the read tag is not a valid category, an error, ShowTagError, is generated and the software closes. If it is valid, it sets the current category to the read category and starts the loop over.

If the current category is not none, the software reads the tag based on what category the current category is and extracts the information from the tag. If the tag isn't expected for the category, for example a condition tag being present in the info category, an error, ShowTagError, is generated and the software closes.

If the tag is expected, the software reads it into temporary memory by using the appropriate tag-reading method from the Utilities module. If the tag isn't valid according to the rules of the selected method in the Utilities module, an error is generated and the software closes. See the Utilities module (p. 11) for more information about the available methods and the errors generated.

If a tag is successfully read, the Utility module method returns the information in the tag and it is put into the memory of the instance of the module and the loop continues.

If the current category is not none and the software finds a category end tag for the current category, the software starts the loop over.

Below follows a flowchart of the.

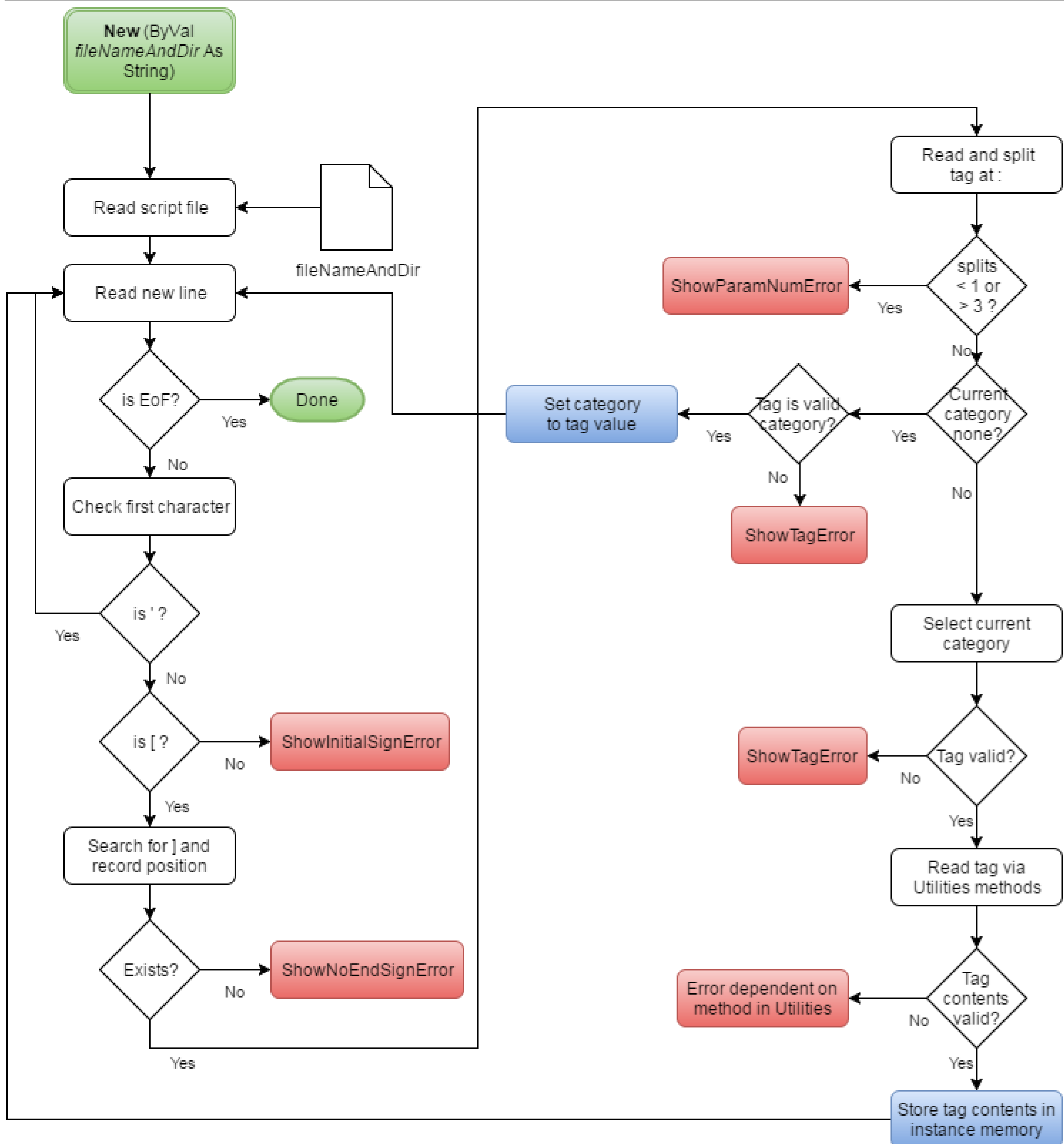

Figure 2: A flowchart of the script reading process for the tasks.

Task script tags

Category	Tag types	Tag structure	Example
none	info, trigger, fail, end, actionlist	[type]	[info]
info	name	[type:string]	[name:task1]
	/info	[type]	[/info]
trigger Handles triggering the task	varLess, varMore, varEqual, varMoreEqual, varLessEqual, varNotEqual	[type:varName:value]	[varLess:var1:3.14]
	setVar, addVar, subVar	[type:varName:value]	[setVar:var1:3.14]
	incVar, decVar	[type:varName]	[incVar:var1]
	click	[type:controlName]	[click:button1]
	timer	[type:time (s)]	[timer:13.37]
	triggerOnce	[type:boolean]	[triggerOnce:false]
	/trigger	[type]	[/trigger]
fail Handles failing the task	varLess, varMore, varEqual, varMoreEqual, varLessEqual, varNotEqual	[type:varName:value]	[varLess:var1:3.14]
	setVar, addVar, subVar	[type:varName:value]	[setVar:var1:3.14]
	incVar, decVar	[type:varName]	[incVar:var1]
	click	[type:controlName]	[click:button1]
	timer	[type:time (s)]	[timer:13.37]
	/fail	[type]	[/fail]
end Handles ending, or completing the task	actionlist	[type:boolean]	[actionlist:true]
	varLess, varMore, varEqual, varMoreEqual, varLessEqual, varNotEqual	[type:varName:value]	[varLess:var1:3.14]
	setVar, addVar, subVar	[type:varName:value]	[setVar:var1:3.14]
	incVar, decVar	[type:varName]	[incVar:var1]
	click	[type:controlName]	[click:button1]
	Timer	[type:time (s)]	[timer:13.37]
	/end	[type]	[/end]
actionlist Handles actions that need to be done in order	click	[type:controlName]	[click:button1]
	varLess, varMore, varEqual, varMoreEqual, varLessEqual, varNotEqual	[type:varName:value]	[varLess:var1:3.14]
	setVar, addVar, subVar	[type:varName:value]	[setVar:var1:3.14]
	incVar, decVar	[type:varName]	[incVar:var1]
	/actionlist	[type]	[/actionlist]

Tag type	Function
actionlist	(Outside categories) Start tag for actionlist category (In end category) Select if actionlist should be used or not
/actionlist	End tag for actionlist category
addVar	When the other necessary conditions are met, a value is added to a variable
click	Condition to click a control
decVar	When the other necessary conditions are met, a variable is decreased by 1
end	Start tag for end category
/end	End tag for end category
fail	Start tag for fail category
/fail	End tag for fail category
incVar	When the other necessary conditions are met, a variable is increased by 1
info	Start tag for info category
/info	End tag for info category
name	The name of the task
setVar	When the other necessary conditions are met, a variable is set to a value
subVar	When the other necessary conditions are met, a value is subtracted from a variable
timer	Condition where a certain time needs to pass
trigger	Start tag for trigger category
/trigger	End tag for trigger category
triggerOnce	The task will only trigger once. Default is true.
varEqual	Condition where a variable needs to be equal to a value
varLess	Condition where a variable needs to be less than a value
varLessEqual	Condition where a variable needs to be less than or equal to a value
varMore	Condition where a variable needs to be more than a value
varMoreEqual	Condition where a variable needs to be more than or equal to a value
varNotEqual	Condition where a variable needs to be anything other than a value

AlarmManager

This module reads, stores and handles the scripted alarms that the driver is to experience during the software use.
The module is initiated by calling the initializeAlarmSystem method, which is done in the MainForm when MainForm is loaded. initializeAlarmSystem is a wrapper method which calls the findAndAddAlarms method.
The findAndAddAlarms method reads the scripts from the Alarms folder one by one, using the readAlarm method. The readAlarm method in turn uses methods from the Utility module to fill an AlarmItem class with the information about the alarm. Then the cleanupAlarm method is called to clean the alarm up from minor scripting errors that won't cause any issues and mentions them in the log files. Lastly the alarm is saved in the AlarmList in this module and a log entry is made that the alarm has been loaded.
When done, based on the setting it can use a timer called UpdateTimer in the Utilities module

to update the alarms or it updates the alarms when a variable is changed in the VariableManager or both. This is controlled by the variables updateWithTimer and updateOnDemand located in the AlarmManager. These variables are public and can either be assigned in the code in the AlarmManager at design-time or from other locations in the software during run-time. The updating checks if the alarm has been triggered, handled or confirmed based on the new state of the variables.
There's also a timer based condition for triggering an alarm, which is handled via the UpdateTimer calling the method updateTimerTick. This method calls, among others, the updateTimerCounters method, which updates the timer counter in the alarm with the duration progressed from when the timer started. Once the timer reaches its target time, this method triggers the alarm.
Clicking a control in the software UI, including the forms themselves, calls the method reportClick in the AlarmManager, from the MainForm. This method lets the MainForm communicate to the AlarmManager what control was clicked, which updates the state of the alarm based on the click conditions, if any.
When the state is updated and all necessary conditions hold for either of the three cases of triggering, handling or confirming an alarm, the triggerAlarm, handleAlarm or confirmAlarm methods are called to respectively trigger, handle or confirm the alarm.

The script reading process

The findAndAddAlarms method, called from the initializeAlarmSystem is the root method for the script reading process.
When findAndAddAlarms is called, it generates a list of all alarm files and loops through them. For each alarm file, the method makes a new AlarmItem class instance and populates it by calling the readAlarm method. The readAlarm method does the script reading and returns the script in the shape of an AlarmItem. Then it calls cleanupAlarm to clean the AlarmItem up from minor errors and report these through the log. The software won't close because of those minor errors. Afterwards, the AlarmItem is added to the list AlarmList in the module and a log entry is made that the alarm has been read.
The text below, for the readAlarm method, is the same as the text for the constructor method in the TaskManager in the script reading process.
The readAlarm method handles the actual reading of the alarm from the file with the name from the input parameter. It starts by reading the script file into a temporary memory of the software as an array of raw text where each entry in the array is a line of text corresponding to the lines in the script file. It then loops through the rows of the script, reading them individually until it reaches the end of the file, in which case it stops looping and nothing more happens.
When it finds a row that is not empty and that is not a comment, it checks if the first character is a "[", a beginning square bracket. This signifies that the line has a tag. If the line doesn't start with a square bracket, but any other character that isn't the comment sign, "'", the row is invalid and an error, ShowInitialSignError, is generated and the software closes.
If the line however does start with a square bracket, the software continues processing the line by checking for the position of the first ending square bracket "]". If it doesn't find one, the line is invalid and an error, ShowNoEndSignError, is generated and the software closes.
If an ending square bracket is found, the software continues and splits the string within the square brackets at the positions where there's a colon sign ":", putting the parts of the string

into an array with one part per entry and checks that the number of entries are between and including 1 and 3. If the number of entries are incorrect, an error, ShowParamNumError, is generated and the software closes.
If the number of entries is correct, the software checks if the current category is none. If it is, the expected tag should be a valid category, for example info, making the tag [info]. If the current category is none and the read tag is not a valid category, an error, ShowTagError, is generated and the software closes. If it is valid, it sets the current category to the read category and starts the loop over.
If the current category is not none, the software reads the tag based on what category the current category is and extracts the information from the tag. If the tag isn't expected for the category, for example a condition tag being present in the info category, an error, ShowTagError, is generated and the software closes.
If the tag is expected, the software reads it into temporary memory by using the appropriate tag-reading method from the Utilities module. If the tag isn't valid according to the rules of the selected method in the Utilities module, an error is generated and the software closes. See the Utilities module (p. 11) for more information about the available methods and the errors generated.
If a tag is successfully read, the Utility module method returns the information in the tag and it is put into the memory of the instance of the module and the loop continues.
If the current category is not none and the software finds a category end tag for the current category, the software starts the loop over.
Below follows a flowchart of the process.

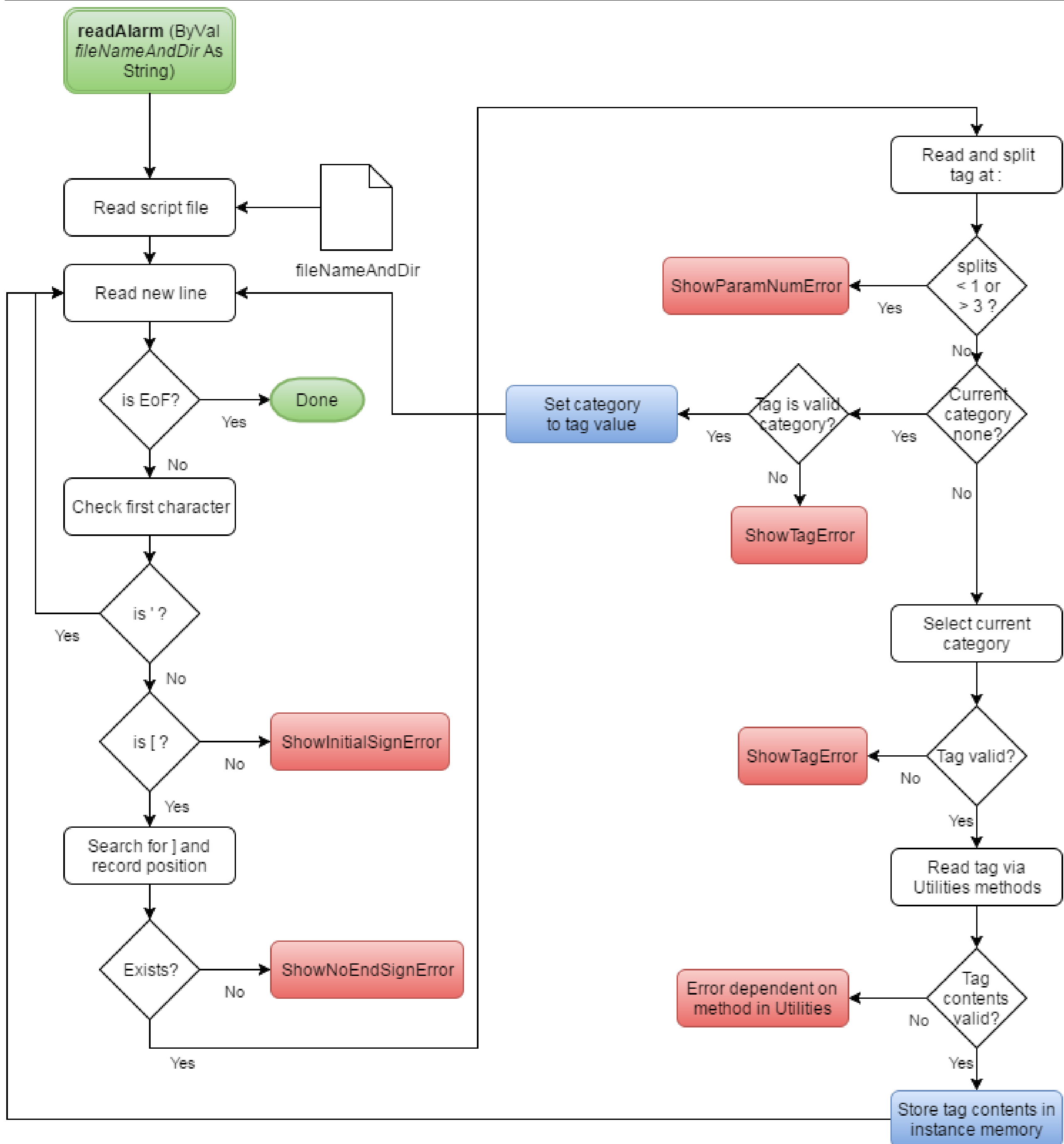

Figure 3: A flowchart of the script reading process of the alarms

Alarm script tags

Category	Tag types	Tag structure	Example
none	info, trigger, confirmation, handling, sound, messageBox	[type]	[info]
info	name	[type:string]	[name:task1]
	/info	[type]	[/info]
trigger	varLess, varMore, varEqual, varMoreEqual, varLessEqual, varNotEqual	[type:varName:value]	[varLess:var1:3.14]
Handles triggering the alarm	setVar, addVar, subVar	[type:varName:value]	[setVar:var1:3.14]
	incVar, decVar	[type:varName]	[incVar:var1]
	click	[type:controlName]	[click:button1]
	timer	[type:time (s)]	[timer:13.37]
	triggerOnce	[type:boolean]	[triggerOnce:false]
	/trigger	[type]	[/trigger]
confirmation	useMessagebox	[type:boolean]	[useMessageBox:true]
Handles confirming the alarm	varLess, varMore, varEqual, varMoreEqual, varLessEqual, varNotEqual	[type:varName:value]	[varLess:var1:3.14]
	setVar, addVar, subVar	[type:varName:value]	[setVar:var1:3.14]
	incVar, decVar	[type:varName]	[incVar:var1]
	click	[type:controlName]	[click:button1]
	/confirmation	[type]	[/confirmation]
handling	varLess, varMore, varEqual, varMoreEqual, varLessEqual, varNotEqual	[type:varName:value]	[varLess:var1:3.14]
Handles handling the alarm	setVar, addVar, subVar	[type:varName:value]	[setVar:var1:3.14]
	incVar, decVar	[type:varName]	[incVar:var1]
	click	[type:controlName]	[click:button1]
	/handling	[type]	[/handling]
sound	useSound	[type:boolean]	[useSound:true]
	soundName	[type:soundName]	[soundName:sound1.wav]
Handles sounds in alarms	doLoop	[type:boolean]	[doLoop:true]
	/sound	[type]	[/sound]
messageBox	useMessagebox	[type:boolean]	[useMessagebox:true]
	text	[type:string]	[text:This is some text]
	caption	[type:string]	[caption:This is a caption]
Handles message box	/messageBox	[type]	[/messageBox]

Tag type	Function
addVar	When the other necessary conditions are met, a value is added to a variable
caption	The window caption of the message box
click	Condition to click a control
confirmation	Start tag for confirmation category
/confirmation	End tag for confirmation category
decVar	When the other necessary conditions are met, a variable is decreased by 1
doLoop	Whether or not the sound played should loop until the alarm is confirmed or handled
handling	Start tag for handling category
/handling	End tag for handling category
incVar	When the other necessary conditions are met, a variable is increased by 1
info	Start tag for info category
/info	End tag for info category
messageBox	Start tag for messageBox category
/messageBox	End tag for messageBox category
name	The name of the alarm
setVar	When the other necessary conditions are met, a variable is set to a value
sound	Start tag for sound category
/sound	End tag for sound category
soundName	The name of the sound to be played
subVar	When the other necessary conditions are met, a value is subtracted from a variable
text	The body of text in the message box
timer	Condition where a certain time needs to pass
trigger	Start tag for trigger category
/trigger	End tag for trigger category
triggerOnce	The alarm will only trigger once. Default is true.
useMessagebox	Whether or not a message box should be used
useSound	Whether or not a sound should be played when the alarm triggers
varEqual	Condition where a variable needs to be equal to a value
varLess	Condition where a variable needs to be less than a value
varLessEqual	Condition where a variable needs to be less than or equal to a value
varMore	Condition where a variable needs to be more than a value
varMoreEqual	Condition where a variable needs to be more than or equal to a value
varNotEqual	Condition where a variable needs to be anything other than a value

SoundManager

This module stores and keeps track of the sounds used by the alarms and the software in general.
The module keeps a list of the sounds in the sound folder and hosts methods for loading the sounds into memory and to unload them from memory. There are also methods to control the playback and volume of the sounds and to check the status of the sounds.
The module utilizes the WinMM, or the Windows Multimedia API and uses the MCI, or Media Control Interface in order to send commands to the API to handle the playback of the sounds.

Using the Windows Multimedia API

The Windows Multimedia API lets the software control the windows built in media player in order to play sounds. This is done via Multimedia Command Strings, using the Media Control Interface for the WinMM.
For this to work, the function mciSendString from the winmm.dll is declared in the SoundManager class and then called in order to send the command string.
The strings that are sent are the commands to for example load the sound into the memory or to play the sound.
As a command string is sent, the function can also return information and some strings are used solely for that reason, such as requesting the volume level or other different kinds of statuses for the sound.
Although not implemented in this software, the WinMM can also send information back on its own to the software in order to for example tell the software that a playback has finished.
The strings used are expanded upon in each of the methods in the member reference list for this module.
(foedan, 2009) (MSDN, p. MCI)

ErrorHandler

This module stores the messages for a number of commonly referenced errors. Other modules call these methods to show a message box with the error to the user. This saves space and makes the code in the other modules more easily readable.

Testing

Live tests in the Chalmers simulator

Tests were made live with the Chalmers vehicle simulator in order to evaluate whether or not the software would work for end user testing with a test driver.
A user interface was made for the test as well as a set of tasks and alarms to create a scenario.
In addition to the standard features of the software, the user interface form was also equipped with a piece of custom code that constantly would write the speed of the vehicle and the position on the road, to a separate custom log. These variables are received via UDP from the simulator.
The following figures show screenshots of the user interface used:

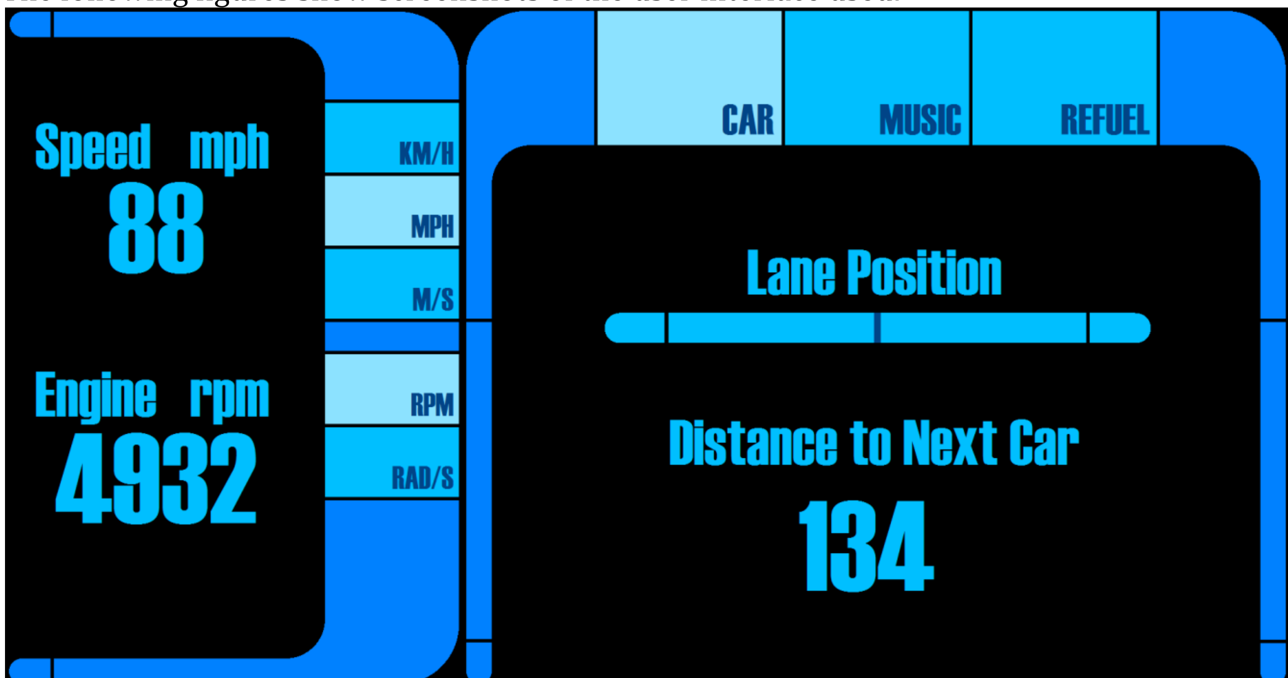

Figure 4: The user interface, with the Car tab selected

Figure 5: The user interface, with the Music tab selected

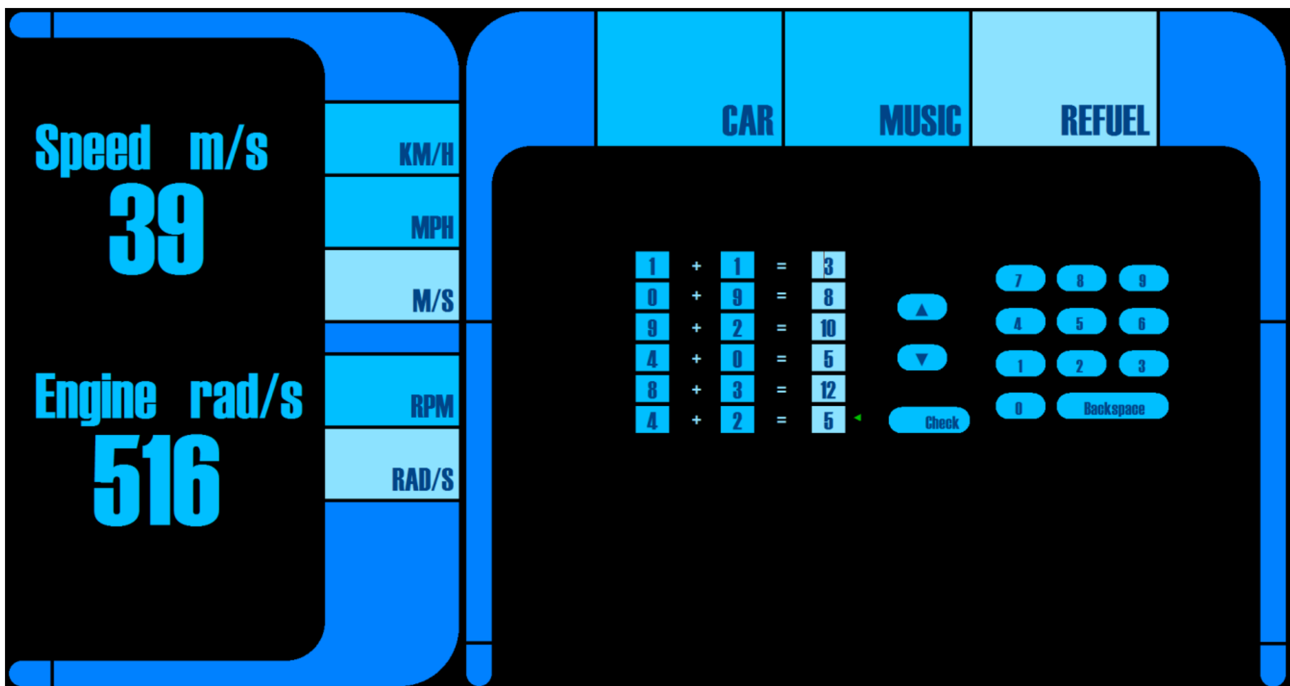

Figure 6: The user interface, with the Refuel tab selected

The driver was placed in the simulator and given a touchpad with the testing software running on it. At the time, there was no mount to place the touchpad in, so the test driver had to hold the touchpad while testing, which may have influenced the test. The main point was however not to get a performance test of the user interface or the driver, but to test if the software worked as intended in a live environment.
The scenario for the test was the following:

1. The driver starts driving and when the driver reaches the first car in front of them and drives within 18 meters of the it, the first task, FollowCar, triggers. The FollowCar task means that until the variable ScriptState becomes 4, the driver has to follow the car in front with a distance of between 5 and 20 meters and they must stay inside their lane on the road. ScriptState becomes 4 when the last task, Refuel, is completed. When FollowCar is triggered, the variable ScriptState is set to 1.
2. When ScriptState is set to 1, the next task, TurnOnStereo, triggers. In this task, the driver has to navigate the user interface to the Music tab where they have to turn the stereo on. This task can't fail, but is completed when the control PlayButton is clicked. When the task completes, the variable ScriptState is set to 2.
3. The next task, Delay, triggers when ScriptState is set to 2 and means that the driver has to wait for 30 seconds until it completes and sets ScriptState to 3. This task can't fail.
4. When ScriptState is set to 3, the final task, Refuel, triggers. This task is completed when the variable MathTest is set to 1. This is done by navigating to the Refuel tab of the user interface and solving the math problems. When all math problems are solved, the variable MathTest is set to 1 and the task completes and sets the ScriptState to 4.
5. When ScriptState is set to 4, if the task FollowCar has not yet been failed, it is now completed.

During the scenario, there are also 3 alarms that can trigger:

- Fuel is an alarm which triggers when ScriptState is set to 3. This is when the task Refuel triggers. The alarm plays a sound to notify the test driver that it is time to do the refuel task.
- FrontAlarm is an alarm which triggers to warn the test driver that they are closer than 10 meters from the car in front. The alarm is handled when the driver is 11 or more meters away from the car. During the time the alarm is active, it plays a sound.
- LaneAlarm is an alarm which triggers to warn the test driver that they are driving outside of the designated lane on the road. The alarm is handled when the driver is back on the designated lane. During the time the alarm is active, it plays a sound.

The results of the test were a set of logs, an event log and a speed log for each of the 3 tests. In order to demonstrate that usable information can be extracted out of the logs, a set of matlab script were made to plot one of the sets of logs into a graph. Speed and lane position is plotted over time and the events are plotted onto the graph at the time they happened. The event data was extracted and put into a table.
The raw logs for test 3, the test presented below, and the matlab scripts can be found in the Live tests in the Chalmers simulator subsection of the Appendix - Test results section. Because of the length of the logs, the other 2 logs are omitted from the report.
Below follows the event data and the figures.

Events triggered			
ID	Name	Type	Time (s)
0	FollowCar	Task	96.869
1	TurnOnStereo	Task	96.883
2	FrontAlarm	Alarm	100.728
3	Delay	Task	111.311
4	Refuel	Task	141.768
5	Fuel	Alarm	141.779
6	LaneAlarm	Alarm	241.559
7	FrontAlarm	Alarm	243.534

Tasks Completed		
ID	Name	Time (s)
1	TurnOnStereo	111.292
3	Delay	141.757
4	Refuel	242.113

Tasks Failed		
ID	Name	Time (s)
0	FollowCar	241.551

Figure 7: Graph showing the speed and lane position of the vehicle without boxes with zoom

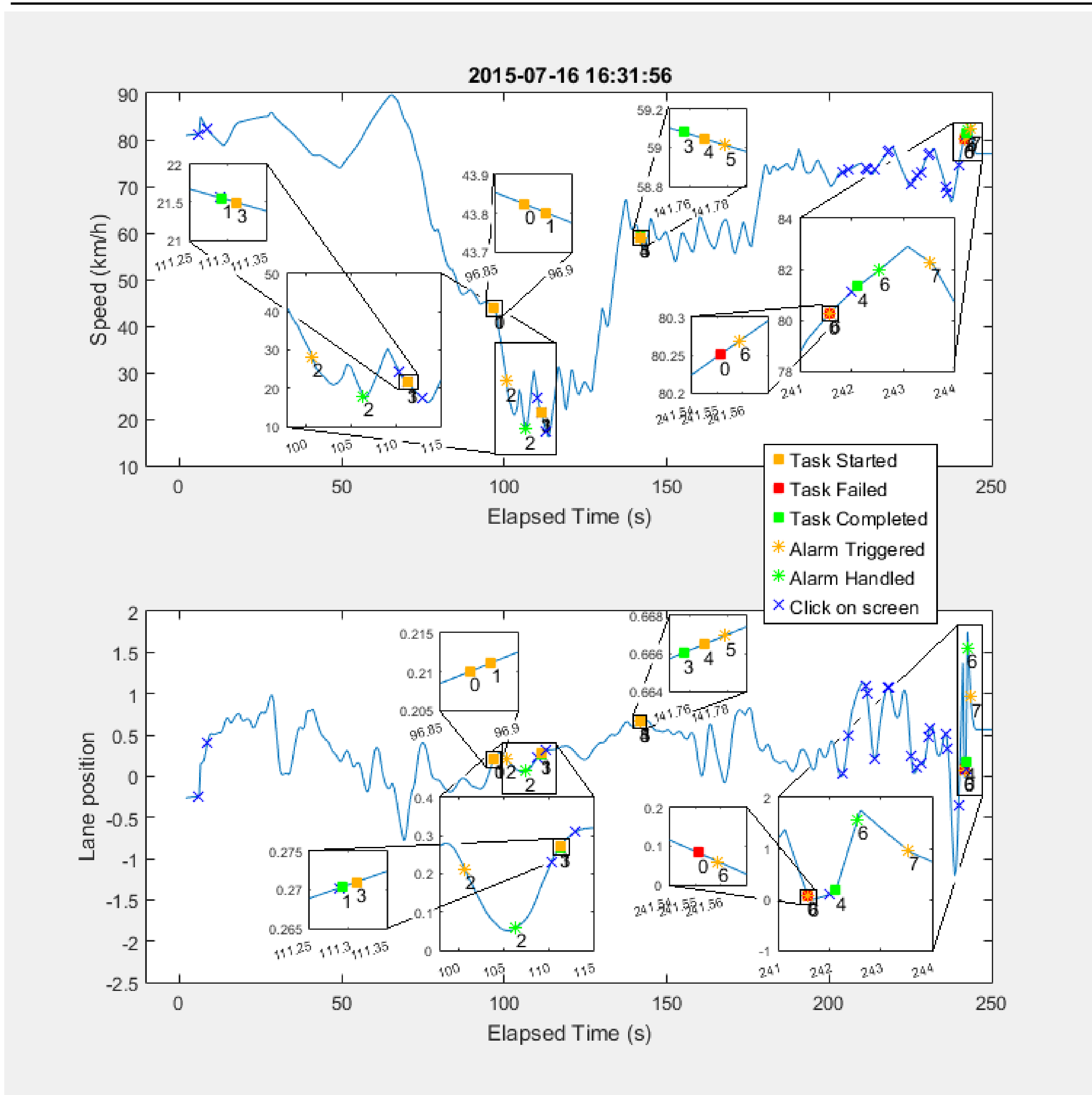

Figure 8: Graph showing the speed and lane position of the vehicle with boxes that zooms in on part of the graph in order to show more details.

Figure 7 and Figure 8 shows the speed and lane position of the vehicle that the test driver was driving in the simulator as well as the times events changed status. The figures also shows when the driver clicked on the touchpad. The first figure is without any zoom boxes in order to give a clean overview of the whole scenario while the second figure has boxes that zoom in on parts of the figure in order to show details of the plot where there was action.
The data shows the test driver first at high speed, trying to catch up to the car in front. As the driver came close, they decelerated the vehicle. An alarm, event 2, triggered as the test driver got within 10 meters of the car in front but was soon handled due to the slow speed. Once the alarm was cleared, a series of 3 touchpad clicks can be seen in the graph. Checking the log file

reveals that the series of 3 clicks are the driver navigating to the Music tab, then clicking play and then navigating back to the Car tab, where the distance to the next car can be seen as a number. When the driver was done, there's soon a sudden acceleration, presumably because they were too far away from the car in front and had to catch up. The FollowCar task, event 0, has not yet been failed however. After the acceleration, the fuel alarm sounds, but it takes about a minute before the test driver manages to get enough control over the situation to be able to navigate to the Refuel tab. While having been relatively steady before, the driver becomes rather unsteady as they start doing the math on the Refuel tab. The log reveals that they also navigated to the Music tab for a second and a half before they navigated to the Refuel tab, indicating a miss click. When the driver is just about to be done with the math, they drive outside the lane and fail the FollowCar task as well as triggering the lane alarm, event 6. Roughly a second afterwards, the Refuel task, event 4, is completed. There are a series of large adjustments to the lane position done by the driver in the final moment, indicating that they had lost required control over the vehicle while looking for too long on the touchpad.

The accuracy of the timing in the logs

The accuracy of the time stated in the logs is tested by checking different outputs for the time and comparing them in a specifically made test scenario.
The timing of the software is tested by utilizing the stopwatch class as well as getting timestamps from the DateTime.Now property, which is how the log entries are made. The stopwatch class is created in the Utilities modules and then restarted when the mouse click happens, which resets it to 0 and lets it start counting up.
3 different setups are tested.

- Only 1 task, which triggers when a certain control is clicked in the UI.
- Only 1 alarm, which triggers when a certain control is clicked in the UI.
- Both 1 task and 1 alarm, which both trigger when a certain control is clicked in the UI.

All 3 setups use the same test UI and with variable updating both timed, with a 20 millisecond interval and when a variable is changed. UDP constantly tries to connect to a simulator device on the network, but will fail as the test is done without one.
The timing is checked at the following times, depending on the test setup:

1. When the PreFilterMessage function is called, in the MainForm. The stopwatch is reset at this point and the first timestamp is made.
2. When the code has progressed in the PreFilterMessage function to after checking that the message is a mouse click message. A timestamp is made and the stopwatch is checked.
3. When the log message for the mouse click is created. A timestamp is made and the stopwatch is checked.
4. When the task is triggered. A timestamp is made and the stopwatch is checked.
5. When the log message for the task is created. A timestamp is made and the stopwatch is checked.
6. When the alarm is triggered. A timestamp is made and the stopwatch is checked.
7. When the log message for the alarm is created. A timestamp is made and the stopwatch is checked.

It is stated on the MSDN page for the DateTime.Now property that its accuracy is 15 milliseconds. (MSDN, p. DateTime.Now Property)
For the stopwatch class, it is stated on MSDN that it counts the ticks from the so called underlying timer mechanism. They explain this as that if the installed hardware and operating system support a high-resolution performance counter, the stopwatch class uses that to measure the elapsed time, otherwise the system timer is used. No explanation is presented on how to deduce if a system is using one or the other method. (MSDN, p. Stopwatch Class)
There are 10000 ticks for each millisecond according to the TimeSpan.TicksPerMillisecond constant (MSDN, p. TimeSpan.TicksPerMillisecond Field).

Below follows tables with test results.
"TS" means timestamp and refers to the timestamp in the code, specifically for testing.
"SW" means stopwatch.
The stopwatch delivers the elapsed time when checked. The value is presented as milliseconds, with 4 decimals.
The timestamps deliver the year, month, day, hour, minute and second, with 7 decimals. In the tables, only the elapsed time, based on the data from the timestamps, will be shown, counted in milliseconds, with 4 decimals, which is equal precision to seconds with 7 decimals.
The result is averaged over 5 tests for each setup, as can be seen in the tables below.

Alarm only (ms)					
	1	2	3	6	7
TS	0	0.0000	1.0000	3.0001	3.0001
SW	0	0.0045	0.2941	0.8329	0.8966
TS	0	0.0000	1.0000	2.0001	3.0001
SW	0	0.0276	0.3261	0.8940	0.9578
TS	0	0.0000	1.0001	2.0001	3.0002
SW	0	0.0087	0.2869	0.8313	0.8951
TS	0	0.0000	1.0000	3.0002	3.0002
SW	0	0.0265	0.3296	0.8739	0.9336
TS	0	0.0000	1.0001	3.0002	3.0002
SW	0	0.0290	0.3454	0.9039	0.9647
Average					
TS	0	0.0000	1.0000	2.6001	3.0002
SW	0	0.0193	0.3164	0.8672	0.9296

Task only (ms)					
	1	2	3	4	5
TS	0	0.0000	1.0001	3.0002	4.0003
SW	0	0.0376	0.3654	1.1643	1.3345
TS	0	0.0000	1.0000	3.0001	4.0002
SW	0	0.0160	0.4033	1.1466	1.3132
TS	0	0.0000	1.0000	3.0002	4.0002
SW	0	0.0091	0.3584	1.1313	1.3236
TS	0	0.0000	1.0001	4.0002	5.0003
SW	0	0.0223	0.5028	1.5605	1.7503
TS	0	0.0000	1.0000	4.0002	4.0002
SW	0	0.0103	0.3615	1.2209	1.3963
Average					
TS	0	0.0000	1.0000	3.4002	4.2002
SW	0	0.0191	0.3983	1.2447	1.4236

Alarm and Task (ms)							
	1	2	3	4	5	6	7
TS	0	0.0000	1.0001	3.0002	4.0003	6.0004	6.0004
SW	0	0.0047	0.3237	1.0617	1.2435	1.7981	1.8015
TS	0	0.0000	1.0001	4.0003	4.0003	6.0004	6.0004
SW	0	0.0061	0.3410	1.1127	1.2804	1.7338	1.7372
TS	0	0.0000	1.0001	4.0002	4.0002	6.0004	6.0004
SW	0	0.0287	0.3526	1.1540	1.3208	1.8091	1.8132
TS	0	0.0000	1.0000	4.0002	5.0003	6.0003	6.0003
SW	0	0.0055	0.3956	1.2623	1.4456	1.9267	1.9302
TS	0	0.0000	1.0001	3.0002	3.0002	5.0003	5.0003
SW	0	0.0332	0.3442	1.0517	1.2162	1.6718	1.6763
Average							
TS	0	0.0000	1.0001	3.6002	4.0003	5.8004	5.8004
SW	0	0.0156	0.3514	1.1285	1.3013	1.7879	1.7917

Discussion

The software and live tests in the Chalmers Simulator

The live testing shows that a fully custom user interface can be made and that the framework properly interacts with the user interface to generate test data and testing functionality. The project takes no credit for that any style of user interface can be created as long as it is within the powers of Visual Basic .Net, that is because of how Visual Basic works. However the framework is made in such a way that it allows the user interface to take advantage of the possibilities of Visual Basic.

The task and alarm system does have some inconvenient limitations when it comes to scripting. It's not possible to do more advanced logics within them, such as nestled AND or OR operators for triggering different events. It is unknown whether or not this would be useful in an actual live testing environment, but it would be something that could be improved in the software for the sake of functionality. Nestled scripts could potentially be used to create such a functionality as it is now, but that would increase clutter in the logs and it would have a non-zero effect on the performance of the software which would be unneeded if the functionality was there.

The standard event log always follow a certain format. However a custom log can be made with any format the designer desires and can log any information that is available to the software.

The software doesn't have any associated script reader or any tools for viewing and analyzing the test data and instead a third party software has to be used for this. It would be useful if there was a software that could load the user interface and graphically show when and where the test driver clicked on a certain location and that could replay the information that was logged in real time. The software could also have tools for automatically plotting the logs to graphs or filter them and search through them etc.

A start function and a reset function could be useful as well. A start function would be some way of triggering the test to start remotely, for example from the simulator or the computer that hosts the test and surveys it. A reset function would be a way to reset the software without exiting it for each time.

The timing of the software

Regardless of the accuracy of the timing and if it's good or bad, the timing itself cannot be taken as a hard fact in determining how long it will take to run a piece of the code. That will depend on what hardware the system uses, how optimized the version of the operating system is, what other applications or executables are running on the system at the same time, how many tasks and alarms there are, how advanced the UI and the other parts of the testing software are and other factors. This means that the times can only be used for comparisons and not predictions.

As can be seen in the test results, presented in the Testing section (p. 24) of the report, the two methods of measuring the time differ greatly. However, as stated, the timestamp method only has an accuracy of 15 milliseconds, which would justify the difference. The stopwatch counts the individual ticks from the timing mechanism in the operating system or the hardware of the system and as such should then be as accurate as possible.

Another interesting point to make is that it takes longer to trigger a task than it takes to trigger an alarm. This is despite the fact that the ReportClickToTask method is called before

the ReportClickToAlarm method in the code.
In the case with a single task, it takes, according to the average of the stopwatch, 1.2447 milliseconds to trigger the task while it takes only 0.8672 milliseconds to trigger the alarm in the case with a single alarm. The alarms also consistently trigger faster than the tasks in all tests made.

The structure of the TaskManager vs. the AlarmManager

The structure of the TaskManager and AlarmManager are different despite them doing similar things. This is because more knowledge had been acquired at the time the AlarmManager was made compared to the TaskManager. Initially, the TaskManager was using a struct to save the task data, which was ok since a task only has one level of variables to be saved. An alarm however needed a layered data structure in order to be less cluttered as it has more detailed settings than a task does. In order to change a variable that is deep in a struct, it was discovered during coding that the whole struct has to be overwritten with a new one that is the copy of the old one but with the variable changed. In order to be able to change all variables individually, a data structure made of nestled classes was used instead, where each class is used as a data type. Afterwards, this was changed for the TaskManager as well in an attempt to assimilate the two managers.
While TaskManager is instanced to create a new task, the AlarmManager contains an AlarmItem class that is instanced instead. The TaskManager list of tasks is stored in the MainForm while the AlarmManager is self contained.
Something that could be improved with the AlarmManager is its reportClick method. It could be divided into one method for each state instead of having it all in one as it is now, in order to make it more understandable and more easily modifiable. A negative impact this may have however is that it may make the software slightly slower. On the other hand, the opposite could be done to the TaskManager's reportClick, that is, to make the different methods that method calls into a single one in order to make it more optimized. But this lowers the modifiability and understandability of the code.
As can be seen in the The timing of the software subsection above (p. 33), an alarm takes shorter time to trigger than a task does. The exact reason for this is unknown, but one possibility is that this is because of how classes and alarms are stored differently requiring tasks to have more layers of method calls. The task uses 5 layers of method calls while alarms use 3 layers of method calls. One reason tasks use more layers of method calls is because wrapper methods are needed to forward the information first into the TaskManager class itself and then from that into each instance of it. Another reason for the difference in time may be that tasks have more methods to call from the reportClick method, which also may impact the timing. However if a task has no actionlist or fail state for example, then these methods should return almost instantly, so it's hard to say where the delay comes from.

Why a custom scripting language as opposed to a pre-existing one?

The reason that a new scripting language was created as opposed to using an existing one is to make it as easy as possible for the designer to make scripts. Regardless of which language was used, code would still have to be written for the software to understand the data which was contained in the scripts and to do error handling. Even if an automatic parser was used to read the data structure and parse it into a predefined data structure in the software, the software would still have to have code to logically deduce what information was there and

how to treat it and to make error handling for the scripts. The actual reading of the script file is the simple part of this task.
For that reason, it was deemed more valuable to have a custom scripting language that is made to be easy to understand and use even for someone who is not a programmer. The parsing is done simultaneously as the logics to deduce what information is there and what to do with it.
However, additional functionality, for example in order to be able to make more complex condition logic like AND or OR statements and nestled AND or OR statements, would be a useful improvement to the scripting language.

Conclusion

The software

The software has a complete set of functions and is in working condition. There are some issues, for example the timing and the lack of functionality in the scripting. Improvements can be made to make the software available for tests requiring more precise timing or more advanced scripting. However working within the boundaries of the software, it's still usable for live testing.
All goals in the specification for the software were met. The software is able to use scripts to make events and alarms. The position and time of clicks as well as what was clicked on the screen is tracked. The time when events trigger or change state is tracked. Anything available to the software can be logged and system information, events and mouse clicks are always logged. There is a networking module that can connect to external vehicle simulators via UDP. Error handling is available, both natively for Visual Basic by using Visual Studio, not credited by the project and built into the software for checking for errors in the scripts, which is made by the project.

Timing

There were no constraints or goals set for the accuracy of the timing in the software, however, improvements can be made. The DateTime.Now property should be switched out for the System.Diagnostics.Stopwatch class in order to improve accuracy of the timing. In addition to that, the time should be checked and carried from the moment the software notices the mouse clicks to the moment the log entries are made. Currently it's programmed so that the time is read the moment the log entries are made, which is less accurate.
The current accuracy is 15 milliseconds for the DateTime.Now property, according to the MSDN webpage (MSDN, p. DateTime.Now Property), plus the latency between when the software registers the mouse click and writes the log entry plus the input latency of the hardware, which is unknown to the software entirely and outside the scope of the project.

Possible improvements

This section lists some ideas and possible improvements a future project involving this software could work with.

- Software that has the functionality to open a user interface and replay the recorded events visually. The software could also have functionality for reading and filtering logs and plot them into graphs or create statistics like for example how many people managed to completed certain tasks, how many miss clicks were made and what the task was that the test driver was performing while doing them, etc.
- Replacing the current timing method with one that saves the time when an action happens and forwards it to the time when a log entry is made about it and in addition to that uses the StopWatch class or a similar, more accurate method than the current DateTime.Now property.
- Implement functionality to send dedicated start, stop and reset commands remotely to the software in order to start, stop or reset a test. This is possible to be done via scripts and programming the user interface accordingly, but a dedicated system for it would be an improvement.
- Implementing functionality to individually name sets of log files with a user name or a pattern that is customizable, which changes automatically with each test driver. The current naming convention that uses the current date and time is harder to keep track of.
- Expand the networking possibilities to use TCP connections and set up network profiles for different kinds of simulators as well as expanding to support not only float variables but any type of data.
- Expand the networking possibilities to allow for real time streaming of the user interface and whatever happens on it to a nearby surveillance pc that an operator is using. Alternatively this can be done via third party streaming software.
- Expanded script support to allow for logic operators such as AND or OR and the possibility to nestle them. New functionality such as creating variables inside of the scripts or creating network packages from inside of the scripts could be useful.
- Uploading the software to a version control system such as Git would possibly make it easier to work with the software and build upon it or use it from different projects at the same time.

Bibliography

developer.com. (2003, February 28). *Microsoft .Net Glossary*. Retrieved October 13, 2015, from developer.com: http://www.developer.com/net/asp/article.php/1756291/Microsoft-NET-Glossary.htm

foedan. (2009, December 27). *[VB.NET] Playing audio with the Media Command Interface from the windows API.* Retrieved May 27, 2016, from Codecall forum: http://forum.codecall.net/topic/52694-vbnet-playing-audio-with-the-media-command-interface-from-the-windows-api/

MSDN. (n.d.). *.Net framework class library*. Retrieved May 27, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/w0x726c2%28v=vs.110%29.aspx

MSDN. (n.d.). *MSDN Documentation Library*. Retrieved May 27, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/

MSDN. (n.d.). *MSDN Library, Access Levels in Visual Basic*. Retrieved June 03, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/76453kax.aspx

MSDN. (n.d.). *MSDN Library, Control Class*. Retrieved June 02, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/system.windows.forms.control(v=vs.110).aspx

MSDN. (n.d.). *MSDN Library, DateTime.Now Propery*. Retrieved May 27, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/system.datetime.now%28v=vs.110%29.aspx

MSDN. (n.d.). *MSDN Library, Declare Statement*. Retrieved June 03, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/4zey12w5.aspx

MSDN. (n.d.). *MSDN Library, Form Class*. Retrieved June 03, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/system.windows.forms.form(v=vs.110).aspx

MSDN. (n.d.). *MSDN Library, IMessageFilter Interface*. Retrieved June 02, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/system.windows.forms.imessagefilter(v=vs.110).aspx

MSDN. (n.d.). *MSDN Library, Implements Statement*. Retrieved June 03, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/7z6hzchx.aspx

MSDN. (n.d.). *MSDN Library, Interface Statement*. Retrieved June 03, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/h9xt0sdd.aspx

MSDN. (n.d.). *MSDN Library, MCI*. Retrieved May 27, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/dd757151(v=vs.85).aspx

MSDN. (n.d.). *MSDN Library, Objects and Classes*. Retrieved June 02, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/527aztek(v=vs.90).aspx

MSDN. (n.d.). *MSDN Library, Status command*. Retrieved june 03, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/windows/desktop/dd798683(v=vs.85).aspx

MSDN. (n.d.). *MSDN Library, Stopwatch Class*. Retrieved May 27, 2016, from Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch(v=vs.110).aspx

MSDN. (n.d.). *MSDN Library, TimeSpan.TicksPerMillisecond Field*. Retrieved May 27, 2016, from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/system.timespan.tickspermillisecond(v=vs.110).aspx

Sjöberg, J., Fredriksson, J., & Falcone, P. (2013, October 28). *Chalmers Projects*. Retrieved August 09, 2016, from Chalmers Vehicle Simulator: https://www.chalmers.se/en/Projects/Pages/Chalmers-vehicle-simulator.aspx

Appendix

Appendix Contents

Appendix - Word List **43**
Appendix - Test results **44**
Live tests in the Chalmers simulator 44
Test 3, log 1 44
Test 3, log 2 45
Core Matlab script 49
Script to extract event data 50
Script to extract speed and lane data 50
Script to plot data 50
Software member reference **53**
Members of MainForm 53
Members of UDPTools 56
Members of UDPFaker 57
Members of Utilities 59
Members of VariableManager 63
Members of TaskManager 64
Members of AlarmManager 70
Members of SoundManager 75
Members of ErrorHandler 79
Appendix - The software code **80**
MainForm 80
UDPTools 86
UDP Faker 92
Utilities 94
VariableManager 101
TaskManager 105
AlarmManager 119
SoundManager 138
ErrorHandler 142

Appendix - Word List

Access level, determines from which part of the software an element can be accessed. Example: Public, Private.
(MSDN, p. Access levels in Visual Basic)

Call, to call something means that a piece of code is being told to run from a remote place.
Example: a function call, which means that said function is commanded to be executed.

Control, an object with a visual representation, used for the user interface of the software. (MSDN, p. Control Class)

Declare, references a procedure that's implemented in another file. This lets the programmer use methods from other files as if they existed in the file the declare statement is in.
(MSDN, p. Declare Statement)

Form, a visual representation of a window or dialog box that can be used to construct the user interface for an application.
(MSDN, p. Form Class)

Implement, when used to describe the functionality of a method, means that the method is implementing the functionality of an Interface member. That is, the method that implements the interface member will be the receiving end for the call to that interface member.
(MSDN, p. Implements Statement)

Interface member, defines a placeholder for a method or property as part of an interface that is to be implemented in another module. This is to give the current module a method to call without knowing the target method.
(MSDN, p. Interface Statement)

Member, an element of a class that helps defining its behaviors and properties. Includes events, variables, methods, constructors and properties.
(developer.com, 2003)

Method, a sub or a function

Module, in this case, a grouping of code that does similar tasks or that contains methods that work together to achieve a goal.
Example: the Utility module has utility methods that can be called. The AlarmManager module has code that has to do with alarms.

Object, a combination of code and data, for example a control, a form or an entire application. All objects are defined by classes. (MSDN, p. Objects and Classes)

Wrapper, a method that is there only to call another method. The reason is often purely for a more easy to modify and to understand infrastructure.
Example: 6 places in a module needs to call a method in another module. If the call is made directly, then if it's later to be changed, all 6 instances of the call has to be changed. However if there's a wrapper that the 6 instances call, only the wrapper has to be changed.

Appendix - Test results

Live tests in the Chalmers simulator

Below follows a set of logs for the tests described in the Live tests in the Chalmers simulator subsection of the Testing section in the main report. The logs are from a live test with a test driver driving in the Chalmers vehicle simulator. After the logs are the Matlab scripts used to extract the data.
Because of poor naming convention of the controls on the test user interface, the use of the buttons with the following names have to be explained:

- LcarsMainButton6, this button is the button to change the UI to show the Car tab.
- LcarsMainButton7, this button is the button to change the UI to show the Music tab.
- LcarsMainButton8, this button is the button to change the UI to show the Refuel tab.
- LcarsActionButton2, this button is the button labeled "check" on the Refuel tab. This checks that the math problems are correct and trigger the variable change if they are.

Test 3, log 1

Filename: Touchpanel log 2015-07-16 163156.txt

```
07-16-2015 16:31:56.774;SY;Loaded sound: C:\Users\sim_ui\Dropbox\Touchpanel\Demo Test\Touchpanel
Platform\bin\Debug\sounds\12. Womble on a Street.mp3
07-16-2015 16:31:56.790;SY;Loaded sound: C:\Users\sim_ui\Dropbox\Touchpanel\Demo Test\Touchpanel
Platform\bin\Debug\sounds\Checkout Scanner Beep-SoundBible.com-593325210.wav
07-16-2015 16:31:56.790;SY;Loaded sound: C:\Users\sim_ui\Dropbox\Touchpanel\Demo Test\Touchpanel
Platform\bin\Debug\sounds\Industrial Alarm-SoundBible.com-1012301296.wav
07-16-2015 16:31:56.790;SY;Loaded sound: C:\Users\sim_ui\Dropbox\Touchpanel\Demo Test\Touchpanel
Platform\bin\Debug\sounds\Woop Woop-SoundBible.com-198943467.wav
07-16-2015 16:31:57.641;SY;Loaded alarm: C:\Users\sim_ui\Dropbox\Touchpanel\Demo Test\Touchpanel
Platform\bin\Debug\alarms\FrontAlarm.alm
07-16-2015 16:31:57.651;SY;Loaded alarm: C:\Users\sim_ui\Dropbox\Touchpanel\Demo Test\Touchpanel
Platform\bin\Debug\alarms\Fuel.alm
07-16-2015 16:31:57.665;SY;Loaded alarm: C:\Users\sim_ui\Dropbox\Touchpanel\Demo Test\Touchpanel
Platform\bin\Debug\alarms\LaneAlarm.alm
07-16-2015 16:31:57.777;SY;System started
07-16-2015 16:32:01.800;MC;0498:0268;MainForm/cbTraffic
07-16-2015 16:32:04.612;MC;0926:0378;MainUI/TabControlMain/TPCar/Label5
07-16-2015 16:33:32.869;EV;TaskStarted;FollowCar|0
07-16-2015 16:33:32.883;EV;TaskStarted;TurnOnStereo|1
07-16-2015 16:33:36.728;EV;AlarmTriggered;FrontAlarm|2
07-16-2015 16:33:42.415;EV;AlarmHandled;FrontAlarm|2
07-16-2015 16:33:46.370;MC;1365:0120;MainUI/LcarsMainButton7
07-16-2015 16:33:47.289;MC;0898:0736;MainUI/TabControlMain/TPMus/PlayButton
07-16-2015 16:33:47.292;EV;TaskCompleted;TurnOnStereo|1
07-16-2015 16:33:47.311;EV;TaskStarted;Delay|3
07-16-2015 16:33:49.050;MC;1052:0125;MainUI/LcarsMainButton6
07-16-2015 16:34:17.757;EV;TaskCompleted;Delay|3
07-16-2015 16:34:17.768;EV;TaskStarted;Refuel|4
07-16-2015 16:34:17.779;EV;AlarmTriggered;Fuel|5
07-16-2015 16:35:20.287;MC;1325:0181;MainUI/LcarsMainButton7
07-16-2015 16:35:21.966;MC;1599:0146;MainUI/LcarsMainButton8
07-16-2015 16:35:27.177;MC;1511:0570;MainUI/TabControlMain/TPMath/BtnMathButton1
07-16-2015 16:35:27.614;MC;1692:0499;MainUI/TabControlMain/TPMath/BtnMathButton6
07-16-2015 16:35:30.158;MC;1381:0561;MainUI/TabControlMain/TPMath/BtnMathButtonDOWN
07-16-2015 16:35:33.963;MC;1604:0421;MainUI/TabControlMain/TPMath/BtnMathButton8
07-16-2015 16:35:34.593;MC;1356:0561;MainUI/TabControlMain/TPMath/BtnMathButtonDOWN
07-16-2015 16:35:41.282;MC;1514:0562;MainUI/TabControlMain/TPMath/BtnMathButton1
07-16-2015 16:35:43.066;MC;1507:0570;MainUI/TabControlMain/TPMath/BtnMathButton1
07-16-2015 16:35:44.221;MC;1367:0561;MainUI/TabControlMain/TPMath/BtnMathButtonDOWN
07-16-2015 16:35:46.441;MC;1600:0493;MainUI/TabControlMain/TPMath/BtnMathButton5
07-16-2015 16:35:46.959;MC;1397:0561;MainUI/TabControlMain/TPMath/BtnMathButtonDOWN
07-16-2015 16:35:51.984;MC;1700:0422;MainUI/TabControlMain/TPMath/BtnMathButton9
```

07-16-2015 16:35:52.533;MC;1371:0561;MainUI/TabControlMain/TPMath/BtnMathButtonDOWN
07-16-2015 16:35:55.837;MC;1692:0426;MainUI/TabControlMain/TPMath/BtnMathButton9
07-16-2015 16:35:57.551;EV;TaskFailed;FollowCar|0|Condition vehicleRadar more than 20 (1000).
07-16-2015 16:35:57.559;EV;AlarmTriggered;LaneAlarm|6
07-16-2015 16:35:57.991;MC;1396:0654;MainUI/TabControlMain/TPMath/LcarsActionButton2
07-16-2015 16:35:58.113;EV;TaskCompleted;Refuel|4
07-16-2015 16:35:58.534;EV;AlarmHandled;LaneAlarm|6
07-16-2015 16:35:59.534;EV;AlarmTriggered;FrontAlarm|7

Test 3, log 2

Filename: Speedlog 2015-07-16 163156.txt

07-16-2015 16:31:58.071;80.92855;1;-0.2644759;3.764476
07-16-2015 16:31:58.575;80.98368;1;-0.2614328;3.761433
07-16-2015 16:31:59.071;81.02039;1;-0.2583932;3.758393
07-16-2015 16:31:59.575;81.07538;1;-0.2553526;3.755352
07-16-2015 16:32:00.100;81.11201;1;-0.2523121;3.752312
07-16-2015 16:32:00.575;81.11201;1;-0.2523121;3.752312
07-16-2015 16:32:01.070;81.16687;1;-0.2492706;3.749271
07-16-2015 16:32:01.572;81.20341;1;-0.2462312;3.746231
07-16-2015 16:32:02.081;81.25815;1;-0.2431936;3.743194
07-16-2015 16:32:02.570;84.91125;1;0.1568512;3.343149
07-16-2015 16:32:03.071;84.33017;1;0.1611497;3.33885
07-16-2015 16:32:03.606;83.58297;1;0.19143;3.30857
07-16-2015 16:32:04.071;82.98403;1;0.2755056;3.224494
07-16-2015 16:32:04.581;82.28591;1;0.3948255;3.105175
07-16-2015 16:32:05.071;81.81019;1;0.46776;3.03224
07-16-2015 16:32:05.571;81.39095;1;0.5136711;2.986329
07-16-2015 16:32:06.074;80.99529;1;0.5126902;2.98731
07-16-2015 16:32:06.570;80.6993;1;0.5073609;2.992639
07-16-2015 16:32:07.098;80.2991;1;0.5047356;2.995265
07-16-2015 16:32:07.572;80.01535;1;0.5133374;2.986663
07-16-2015 16:32:08.092;79.65126;1;0.5256194;2.97438
07-16-2015 16:32:08.573;79.37244;1;0.5256012;2.974399
07-16-2015 16:32:09.076;79.02351;1;0.536855;2.963145
07-16-2015 16:32:09.571;78.68211;1;0.5917674;2.908233
07-16-2015 16:32:10.070;78.98541;1;0.6560584;2.843942
07-16-2015 16:32:10.602;79.53481;1;0.7034272;2.796573
07-16-2015 16:32:11.075;79.9996;1;0.6963547;2.803645
07-16-2015 16:32:11.595;80.83651;1;0.6574647;2.842535
07-16-2015 16:32:12.074;81.53185;1;0.6435;2.8565
07-16-2015 16:32:12.575;82.26164;1;0.6739383;2.826062
07-16-2015 16:32:13.073;82.96429;1;0.715273;2.784727
07-16-2015 16:32:13.572;83.30013;1;0.7387131;2.761287
07-16-2015 16:32:14.074;83.674;1;0.7637249;2.736275
07-16-2015 16:32:14.570;83.74975;1;0.7512321;2.748768
07-16-2015 16:32:15.080;83.85477;1;0.6958987;2.804101
07-16-2015 16:32:15.573;83.93853;1;0.6680803;2.83192
07-16-2015 16:32:16.074;84.03133;1;0.6741208;2.825879
07-16-2015 16:32:16.570;84.1293;1;0.6916638;2.808336
07-16-2015 16:32:17.073;84.21918;1;0.6957278;2.804272
07-16-2015 16:32:17.598;84.31416;1;0.6818292;2.818171
07-16-2015 16:32:18.070;84.38177;1;0.6424345;2.857565
07-16-2015 16:32:18.581;84.46211;1;0.5862037;2.913796
07-16-2015 16:32:19.070;84.539;1;0.5878198;2.91218
07-16-2015 16:32:19.577;84.61632;1;0.6248153;2.875185
07-16-2015 16:32:20.075;84.69573;1;0.6828815;2.817119
07-16-2015 16:32:20.575;84.75952;1;0.7353314;2.764668
07-16-2015 16:32:21.103;84.83797;1;0.7708608;2.729139
07-16-2015 16:32:21.570;84.89661;1;0.7712073;2.728793
07-16-2015 16:32:22.072;84.96992;1;0.7607808;2.739219
07-16-2015 16:32:22.569;85.00195;1;0.7706377;2.729362
07-16-2015 16:32:23.072;84.92711;1;0.8189242;2.681076
07-16-2015 16:32:23.572;85.29279;1;0.9058505;2.594149
07-16-2015 16:32:24.072;85.82711;1;0.9710655;2.528934
07-16-2015 16:32:24.612;85.78526;1;0.981238;2.518762
07-16-2015 16:32:25.076;85.29832;1;0.8970309;2.602969
07-16-2015 16:32:25.581;84.75299;1;0.6733448;2.826655
07-16-2015 16:32:26.069;84.46617;1;0.4565867;3.043413
07-16-2015 16:32:26.575;84.09116;1;0.2193124;3.280688
07-16-2015 16:32:27.070;83.73824;1;0.06817506;3.431825
07-16-2015 16:32:27.579;83.44673;1;0.01801503;3.481985
07-16-2015 16:32:28.069;83.09641;1;0.0206503;3.47935
07-16-2015 16:32:28.573;82.82417;1;0.03053357;3.469466
07-16-2015 16:32:29.109;82.4873;1;0.03254301;3.467457
07-16-2015 16:32:29.570;82.23451;1;0.02044525;3.479555
07-16-2015 16:32:30.074;81.89509;1;0.04103341;3.458966
07-16-2015 16:32:30.573;81.59846;1;0.1365271;3.363473
07-16-2015 16:32:31.075;81.32362;1;0.242349;3.257651
07-16-2015 16:32:31.599;80.88615;1;0.3751785;3.124821
07-16-2015 16:32:32.076;80.51079;1;0.4672182;3.032782
07-16-2015 16:32:32.588;80.06702;1;0.5577819;2.942218
07-16-2015 16:32:33.073;79.77528;1;0.5957803;2.90422
07-16-2015 16:32:33.579;79.35843;1;0.5894699;2.91053
07-16-2015 16:32:34.071;78.91792;1;0.5680246;2.931975
07-16-2015 16:32:34.574;78.5415;1;0.5999807;2.900019
07-16-2015 16:32:35.093;78.10424;1;0.7485722;2.751428
07-16-2015 16:32:35.572;77.76938;1;0.7980671;2.701933
07-16-2015 16:32:36.079;77.34862;1;0.7828793;2.717121
07-16-2015 16:32:36.574;77.02127;1;0.7983541;2.701646
07-16-2015 16:32:37.081;76.73953;1;0.8131804;2.68682
07-16-2015 16:32:37.570;76.64185;1;0.7694025;2.730597
07-16-2015 16:32:38.073;76.63673;1;0.6976128;2.802387
07-16-2015 16:32:38.598;76.65706;1;0.6229804;2.87702
07-16-2015 16:32:39.071;76.70351;1;0.5695904;2.930409
07-16-2015 16:32:39.587;76.59686;1;0.5170364;2.982964
07-16-2015 16:32:40.070;76.37635;1;0.4904919;3.009508
07-16-2015 16:32:40.572;76.09206;1;0.4649737;3.035026
07-16-2015 16:32:41.071;75.84776;1;0.4403063;3.059694
07-16-2015 16:32:41.575;75.64235;1;0.4263457;3.073654
07-16-2015 16:32:42.103;75.38717;1;0.3972158;3.102784
07-16-2015 16:32:42.573;75.14171;1;0.3514828;3.148517
07-16-2015 16:32:43.086;74.88255;1;0.3074259;3.192574
07-16-2015 16:32:43.571;74.68404;1;0.2855375;3.214463
07-16-2015 16:32:44.070;74.434;1;0.2653601;3.23464
07-16-2015 16:32:44.570;74.18785;1;0.2348906;3.265109
07-16-2015 16:32:45.075;73.99052;1;0.1987113;3.301289
07-16-2015 16:32:45.594;73.91786;1;0.1751255;3.324874
07-16-2015 16:32:46.070;74.36086;1;0.1888207;3.311179
07-16-2015 16:32:46.580;74.90016;1;0.2410137;3.258986
07-16-2015 16:32:47.072;75.31337;1;0.2809551;3.219045
07-16-2015 16:32:47.569;75.83151;1;0.311683;3.188317
07-16-2015 16:32:48.070;76.33822;1;0.2868418;3.213158
07-16-2015 16:32:48.571;76.74462;1;0.228043;3.271957
07-16-2015 16:32:49.103;77.18775;1;0.1541822;3.345818
07-16-2015 16:32:49.571;77.60313;1;0.11277;3.38723
07-16-2015 16:32:50.083;78.05045;1;0.1227509;3.377249
07-16-2015 16:32:50.570;78.53098;1;0.1530264;3.346974
07-16-2015 16:32:51.070;79.13911;1;0.1661712;3.333829
07-16-2015 16:32:51.569;79.72997;1;0.1369011;3.363099
07-16-2015 16:32:52.073;80.17674;1;0.07209378;3.427906
07-16-2015 16:32:52.601;80.946;1;-0.03231678;3.532317

07-16-2015 16:32:53.070;81.5624;1;-0.107113;3.607113
07-16-2015 16:32:53.588;82.25599;1;-0.1795794;3.679579
07-16-2015 16:32:54.074;82.75911;1;-0.1910039;3.691004
07-16-2015 16:32:54.570;83.4017;1;-0.1099965;3.609997
07-16-2015 16:32:55.072;84.03079;1;0.04194427;3.458056
07-16-2015 16:32:55.572;84.48428;1;0.1736998;3.3263
07-16-2015 16:32:56.104;85.0628;1;0.2975831;3.202417
07-16-2015 16:32:56.574;85.5379;1;0.3478447;3.152155
07-16-2015 16:32:57.082;86.06249;1;0.3414982;3.158502
07-16-2015 16:32:57.571;86.51305;1;0.2796404;3.22036
07-16-2015 16:32:58.072;87.04788;1;0.1572588;3.342741
07-16-2015 16:32:58.570;87.56761;1;0.03150236;3.468498
07-16-2015 16:32:59.075;87.96394;1;-0.04691661;3.546917
07-16-2015 16:32:59.598;88.45843;1;-0.1097654;3.609765
07-16-2015 16:33:00.074;88.85368;1;-0.1394081;3.639408
07-16-2015 16:33:00.585;89.32967;1;-0.1540649;3.654065
07-16-2015 16:33:01.070;89.65314;1;-0.1276994;3.627699
07-16-2015 16:33:01.573;89.59568;1;-0.08307175;3.583072
07-16-2015 16:33:02.071;89.23524;1;-0.131531;3.631531
07-16-2015 16:33:02.571;88.91296;1;-0.2294041;3.729404
07-16-2015 16:33:03.126;88.41817;1;-0.3629826;3.862983
07-16-2015 16:33:03.574;87.70233;1;-0.4708928;3.970893
07-16-2015 16:33:04.089;86.57903;1;-0.5963127;4.096313
07-16-2015 16:33:04.572;85.68117;1;-0.6994684;4.199469
07-16-2015 16:33:05.080;84.64259;1;-0.7869924;4.286993
07-16-2015 16:33:05.570;83.94217;1;-0.7522799;4.25228
07-16-2015 16:33:06.072;83.41249;1;-0.6235604;4.12356
07-16-2015 16:33:06.597;82.69688;1;-0.4213218;3.921322
07-16-2015 16:33:07.071;82.14148;1;-0.291783;3.791783
07-16-2015 16:33:07.579;80.96223;1;-0.189821;3.689821
07-16-2015 16:33:08.075;79.61568;1;-0.1438204;3.64382
07-16-2015 16:33:08.570;78.06911;1;-0.05938814;3.559388
07-16-2015 16:33:09.074;76.7774;1;0.1037873;3.396213
07-16-2015 16:33:09.575;75.78404;1;0.2508414;3.249159
07-16-2015 16:33:10.120;74.4859;1;0.3584604;3.14154
07-16-2015 16:33:10.574;73.2392;1;0.3945155;3.105484
07-16-2015 16:33:11.113;71.12947;1;0.4035242;3.096476
07-16-2015 16:33:11.574;69.32743;1;0.3763583;3.123642
07-16-2015 16:33:12.096;67.13798;1;0.3133507;3.186649
07-16-2015 16:33:12.570;64.96278;1;0.2133465;3.286654
07-16-2015 16:33:13.072;63.40934;1;0.1237743;3.376226
07-16-2015 16:33:13.572;61.79726;1;0.02621243;3.473788
07-16-2015 16:33:14.075;61.04568;1;-0.01718873;3.517189
07-16-2015 16:33:14.585;60.33419;1;-0.0361018;3.536102
07-16-2015 16:33:15.087;60.60014;1;-0.04346519;3.543465
07-16-2015 16:33:15.585;60.79613;1;-0.04537579;3.545376
07-16-2015 16:33:16.126;59.6085;1;-0.05819348;3.558193
07-16-2015 16:33:16.591;58.1679;1;-0.08017324;3.580173
07-16-2015 16:33:17.119;56.43425;1;-0.1148611;3.614861
07-16-2015 16:33:17.589;55.13478;1;-0.1386535;3.638654
07-16-2015 16:33:18.097;53.96114;1;-0.1645263;3.664526
07-16-2015 16:33:18.590;52.97463;1;-0.1832514;3.683251
07-16-2015 16:33:19.091;52.47281;1;-0.1911776;3.691178
07-16-2015 16:33:19.614;52.10294;1;-0.1921524;3.692152
07-16-2015 16:33:20.087;52.17475;1;-0.1821536;3.682154
07-16-2015 16:33:20.588;51.28307;1;-0.1581438;3.658144
07-16-2015 16:33:21.085;50.36849;1;-0.130525;3.630525
07-16-2015 16:33:21.589;49.09759;1;-0.09556358;3.595564
07-16-2015 16:33:22.087;47.86444;1;-0.05991297;3.559913
07-16-2015 16:33:22.591;47.11605;1;-0.03850239;3.538502
07-16-2015 16:33:23.114;46.8447;1;-0.02670681;3.526707
07-16-2015 16:33:23.585;46.83155;1;-0.02832407;3.528324
07-16-2015 16:33:24.101;47.0096;1;-0.04014989;3.54015
07-16-2015 16:33:24.590;47.28576;1;-0.0542799;3.55428
07-16-2015 16:33:25.088;47.61823;1;-0.07896792;3.578968
07-16-2015 16:33:25.588;47.8863;1;-0.1040629;3.604063
07-16-2015 16:33:26.089;47.56127;1;-0.1193237;3.619324
07-16-2015 16:33:26.620;46.95418;1;-0.1299986;3.629999
07-16-2015 16:33:27.088;46.479;1;-0.1339978;3.633998
07-16-2015 16:33:27.591;45.81721;1;-0.1438746;3.643875
07-16-2015 16:33:28.089;45.13776;1;-0.1513886;3.651389
07-16-2015 16:33:28.599;44.61165;1;-0.1572455;3.657246
07-16-2015 16:33:29.090;44.80218;1;-0.1496395;3.64964
07-16-2015 16:33:29.585;44.96246;1;-0.1310168;3.631017
07-16-2015 16:33:30.110;45.15187;1;-0.09259045;3.59259
07-16-2015 16:33:30.588;45.15907;1;-0.04971216;3.549712
07-16-2015 16:33:31.094;44.96239;1;0.01366243;3.486338
07-16-2015 16:33:31.586;44.78764;1;0.06726016;3.43274
07-16-2015 16:33:32.093;44.66467;1;0.1281126;3.371887
07-16-2015 16:33:32.586;44.26445;1;0.1871191;3.312881
07-16-2015 16:33:33.101;43.46223;1;0.2287673;3.271233
07-16-2015 16:33:33.639;42.21969;1;0.2610343;3.238966
07-16-2015 16:33:34.091;40.36944;1;0.2756582;3.224342
07-16-2015 16:33:34.604;37.986;1;0.2810893;3.218911
07-16-2015 16:33:35.088;36.35519;1;0.2768066;3.223193
07-16-2015 16:33:35.590;33.76475;1;0.2634617;3.236538
07-16-2015 16:33:36.089;31.46023;1;0.2414942;3.258506
07-16-2015 16:33:36.588;29.19058;1;0.2204511;3.279549
07-16-2015 16:33:37.135;25.2462;1;0.1878248;3.312175
07-16-2015 16:33:37.589;24.04124;1;0.1691816;3.330818
07-16-2015 16:33:38.114;22.80127;1;0.1425332;3.357467
07-16-2015 16:33:38.591;21.44734;1;0.1213155;3.378685
07-16-2015 16:33:39.087;20.81394;1;0.09909964;3.4009
07-16-2015 16:33:39.588;21.45398;1;0.08248756;3.417512
07-16-2015 16:33:40.093;22.88257;1;0.07178429;3.428216
07-16-2015 16:33:40.603;26.3682;1;0.05852201;3.441478
07-16-2015 16:33:41.086;25.34481;1;0.05244981;3.44755
07-16-2015 16:33:41.623;22.0196;1;0.05110895;3.448891
07-16-2015 16:33:42.090;19.51354;1;0.05408525;3.445915
07-16-2015 16:33:42.592;17.11123;1;0.06202488;3.437975
07-16-2015 16:33:43.090;18.93798;1;0.07162508;3.428375
07-16-2015 16:33:43.595;21.52598;1;0.08357899;3.416421
07-16-2015 16:33:44.116;24.5;1;0.1034951;3.396505
07-16-2015 16:33:44.587;27.55981;1;0.1289552;3.371045
07-16-2015 16:33:45.102;30.42485;1;0.1616515;3.338349
07-16-2015 16:33:45.589;28.31878;1;0.188969;3.311031
07-16-2015 16:33:46.115;25.64505;1;0.2175496;3.28245
07-16-2015 16:33:46.588;23.33663;1;0.2450471;3.254953
07-16-2015 16:33:47.085;22.14388;1;0.263098;3.236902
07-16-2015 16:33:47.617;20.62224;1;0.2817837;3.218216
07-16-2015 16:33:48.089;19.47848;1;0.2929794;3.207021
07-16-2015 16:33:48.589;18.16511;1;0.3039047;3.196095
07-16-2015 16:33:49.140;17.18194;1;0.3103286;3.189671
07-16-2015 16:33:49.588;16.11415;1;0.3154034;3.184597
07-16-2015 16:33:50.087;17.14196;1;0.3184325;3.181567
07-16-2015 16:33:50.588;19.7709;1;0.3196341;3.180366
07-16-2015 16:33:51.134;22.78538;1;0.320777;3.179223
07-16-2015 16:33:51.586;24.81258;1;0.3223597;3.17764
07-16-2015 16:33:52.104;28.45768;1;0.3277902;3.17221
07-16-2015 16:33:52.588;31.26059;1;0.3350374;3.164963
07-16-2015 16:33:53.088;30.54982;1;0.3458221;3.154178
07-16-2015 16:33:53.591;28.75314;1;0.3550946;3.144905
07-16-2015 16:33:54.088;26.62501;1;0.3602138;3.139786
07-16-2015 16:33:54.598;22.9854;1;0.3647343;3.135266
07-16-2015 16:33:55.090;21.91422;1;0.3643566;3.135643
07-16-2015 16:33:55.600;24.43351;1;0.3588158;3.141184
07-16-2015 16:33:56.086;27.14288;1;0.352518;3.147482
07-16-2015 16:33:56.616;30.53263;1;0.3394656;3.160534
07-16-2015 16:33:57.090;30.38422;1;0.3171882;3.182812
07-16-2015 16:33:57.588;28.078;1;0.2990009;3.200999
07-16-2015 16:33:58.123;26.52636;1;0.2737294;3.226271
07-16-2015 16:33:58.587;25.47058;1;0.2556428;3.244357
07-16-2015 16:33:59.104;25.18921;1;0.2334171;3.266583
07-16-2015 16:33:59.586;26.2979;1;0.2188058;3.281194
07-16-2015 16:34:00.086;27.22713;1;0.2038232;3.296177
07-16-2015 16:34:00.588;28.0093;1;0.1972679;3.302732

07-16-2015 16:34:01.087;27.83235;1;0.201269;3.298731
07-16-2015 16:34:01.614;27.06916;1;0.2164153;3.283585
07-16-2015 16:34:02.086;26.48882;1;0.2362874;3.263713
07-16-2015 16:34:02.602;25.79462;1;0.27029;3.22971
07-16-2015 16:34:03.086;26.26909;1;0.2986358;3.201364
07-16-2015 16:34:03.610;29.30517;1;0.3314556;3.168544
07-16-2015 16:34:04.085;32.57767;1;0.3570033;3.142997
07-16-2015 16:34:04.596;34.63705;1;0.374731;3.125269
07-16-2015 16:34:05.115;38.24115;1;0.3971474;3.102853
07-16-2015 16:34:05.587;40.16274;1;0.403738;3.096262
07-16-2015 16:34:06.128;43.83435;1;0.4055971;3.094403
07-16-2015 16:34:06.586;43.04422;1;0.4087954;3.091205
07-16-2015 16:34:07.090;41.17619;1;0.4230893;3.076911
07-16-2015 16:34:07.586;39.8565;1;0.4516313;3.048369
07-16-2015 16:34:08.086;40.14079;1;0.4758494;3.024151
07-16-2015 16:34:08.649;43.52093;1;0.4993396;3.00066
07-16-2015 16:34:09.085;45.94327;1;0.5061842;2.993816
07-16-2015 16:34:09.594;49.49403;1;0.4989099;3.00109
07-16-2015 16:34:10.090;51.94427;1;0.4942219;3.005778
07-16-2015 16:34:10.591;55.38753;1;0.5024794;2.997521
07-16-2015 16:34:11.085;58.0398;1;0.5216289;2.978371
07-16-2015 16:34:11.591;59.78349;1;0.5429966;2.957003
07-16-2015 16:34:12.135;62.57433;1;0.5895983;2.910402
07-16-2015 16:34:12.586;64.18047;1;0.6142458;2.885754
07-16-2015 16:34:13.126;66.80211;1;0.6448351;2.855165
07-16-2015 16:34:13.587;67.10619;1;0.6578563;2.842144
07-16-2015 16:34:14.091;64.6776;1;0.6571329;2.842867
07-16-2015 16:34:14.586;62.51784;1;0.6307537;2.869246
07-16-2015 16:34:15.086;60.77446;1;0.6087402;2.89126
07-16-2015 16:34:15.618;60.80494;1;0.5999519;2.900048
07-16-2015 16:34:16.087;61.6637;1;0.6012861;2.898714
07-16-2015 16:34:16.603;61.85991;1;0.6147163;2.885284
07-16-2015 16:34:17.091;60.95549;1;0.6328683;2.867132
07-16-2015 16:34:17.595;59.57193;1;0.6592548;2.840745
07-16-2015 16:34:18.091;58.06644;1;0.6799808;2.820019
07-16-2015 16:34:18.591;57.19824;1;0.6906282;2.809372
07-16-2015 16:34:19.104;56.78229;1;0.6904074;2.809593
07-16-2015 16:34:19.587;57.34499;1;0.6818476;2.818152
07-16-2015 16:34:20.092;59.12773;1;0.6674467;2.832553
07-16-2015 16:34:20.585;60.87657;1;0.6577908;2.842209
07-16-2015 16:34:21.087;63.14455;1;0.6311163;2.868884
07-16-2015 16:34:21.590;62.47541;1;0.5948065;2.905194
07-16-2015 16:34:22.087;60.96169;1;0.5662873;2.933713
07-16-2015 16:34:22.634;59.31733;1;0.5505645;2.949435
07-16-2015 16:34:23.086;58.63817;1;0.5496601;2.95034
07-16-2015 16:34:23.588;58.41107;1;0.5478145;2.952185
07-16-2015 16:34:24.088;58.68586;1;0.5418185;2.958181
07-16-2015 16:34:24.587;59.22456;1;0.5361239;2.963876
07-16-2015 16:34:25.091;59.79954;1;0.538835;2.961165
07-16-2015 16:34:25.587;60.33157;1;0.5479819;2.952018
07-16-2015 16:34:26.114;59.68472;1;0.5593756;2.940624
07-16-2015 16:34:26.586;58.57782;1;0.5568569;2.943143
07-16-2015 16:34:27.104;57.17842;1;0.5352402;2.96476
07-16-2015 16:34:27.591;56.12354;1;0.5123283;2.987672
07-16-2015 16:34:28.090;54.9312;1;0.5013699;2.99863
07-16-2015 16:34:28.588;53.72437;1;0.5102755;2.989725
07-16-2015 16:34:29.091;54.3905;1;0.5292733;2.970727
07-16-2015 16:34:29.616;56.49551;1;0.5426921;2.957308
07-16-2015 16:34:30.088;58.31854;1;0.540328;2.959672
07-16-2015 16:34:30.610;59.97614;1;0.5196408;2.980359
07-16-2015 16:34:31.087;59.99904;1;0.4885414;3.011459
07-16-2015 16:34:31.586;59.16668;1;0.4560343;3.043966
07-16-2015 16:34:32.089;58.28569;1;0.4468064;3.053194
07-16-2015 16:34:32.587;57.67968;1;0.4608598;3.03914
07-16-2015 16:34:33.116;56.7397;1;0.509202;2.990798
07-16-2015 16:34:33.588;56.1852;1;0.524503;2.975497
07-16-2015 16:34:34.100;55.78877;1;0.5121612;2.987839
07-16-2015 16:34:34.589;57.02231;1;0.4612874;3.038713
07-16-2015 16:34:35.088;59.23135;1;0.3428552;3.157145
07-16-2015 16:34:35.589;61.67836;1;0.1915752;3.308425
07-16-2015 16:34:36.099;63.55732;1;0.09157959;3.40842
07-16-2015 16:34:36.614;63.21045;1;0.00513857;3.494861
07-16-2015 16:34:37.085;62.12442;1;0.008326846;3.491673
07-16-2015 16:34:37.590;60.93164;1;0.1251593;3.374841
07-16-2015 16:34:38.087;60.05704;1;0.2829404;3.21706
07-16-2015 16:34:38.590;59.03958;1;0.4975697;3.00243
07-16-2015 16:34:39.087;58.10959;1;0.6273878;2.872612
07-16-2015 16:34:39.588;57.54987;1;0.6290699;2.87093
07-16-2015 16:34:40.120;56.88835;1;0.5440933;2.955907
07-16-2015 16:34:40.590;56.48803;1;0.4401686;3.059831
07-16-2015 16:34:41.085;57.11711;1;0.3189293;3.181071
07-16-2015 16:34:41.588;58.14911;1;0.2314079;3.268592
07-16-2015 16:34:42.104;59.57523;1;0.1454342;3.354566
07-16-2015 16:34:42.586;61.00628;1;0.07864671;3.421353
07-16-2015 16:34:43.086;61.98709;1;-0.003080693;3.503081
07-16-2015 16:34:43.613;62.62342;1;-0.08101603;3.581016
07-16-2015 16:34:44.086;62.8665;1;-0.06806541;3.568065
07-16-2015 16:34:44.601;62.474;1;0.005602438;3.494398
07-16-2015 16:34:45.087;61.20953;1;0.1176297;3.38237
07-16-2015 16:34:45.620;59.31758;1;0.2884713;3.211529
07-16-2015 16:34:46.088;57.44983;1;0.4771353;3.022865
07-16-2015 16:34:46.589;56.00046;1;0.6119329;2.888067
07-16-2015 16:34:47.114;55.49838;1;0.7276226;2.772377
07-16-2015 16:34:47.590;56.79045;1;0.7658181;2.734182
07-16-2015 16:34:48.102;58.22067;1;0.7982737;2.701726
07-16-2015 16:34:48.591;59.44664;1;0.8074051;2.692595
07-16-2015 16:34:49.095;61.12964;1;0.7891412;2.710859
07-16-2015 16:34:49.588;62.42305;1;0.7593441;2.740656
07-16-2015 16:34:50.100;61.3429;1;0.7404211;2.759579
07-16-2015 16:34:50.619;59.5848;1;0.7799569;2.720043
07-16-2015 16:34:51.089;58.41875;1;0.8155919;2.684408
07-16-2015 16:34:51.586;57.38232;1;0.8314947;2.668505
07-16-2015 16:34:52.086;56.94075;1;0.7868909;2.713109
07-16-2015 16:34:52.585;59.27358;1;0.659955;2.840045
07-16-2015 16:34:53.091;61.77791;1;0.5333329;2.966667
07-16-2015 16:34:53.591;63.71256;1;0.447786;3.052214
07-16-2015 16:34:54.098;66.01864;1;0.3699711;3.130029
07-16-2015 16:34:54.587;67.9748;1;0.3308863;3.169114
07-16-2015 16:34:55.093;70.32716;1;0.308245;3.191755
07-16-2015 16:34:55.589;72.22113;1;0.3051506;3.194849
07-16-2015 16:34:56.088;74.48196;1;0.3314335;3.168566
07-16-2015 16:34:56.588;74.56351;1;0.3706118;3.129388
07-16-2015 16:34:57.086;73.52792;1;0.3889179;3.111082
07-16-2015 16:34:57.617;72.5349;1;0.3601949;3.139805
07-16-2015 16:34:58.089;72.9726;1;0.3018721;3.198128
07-16-2015 16:34:58.604;74.20248;1;0.2256513;3.274349
07-16-2015 16:34:59.089;74.3143;1;0.1921599;3.30784
07-16-2015 16:34:59.589;73.71992;1;0.1804669;3.319533
07-16-2015 16:35:00.091;73.40479;1;0.1736316;3.326368
07-16-2015 16:35:00.585;73.69277;1;0.1610104;3.338989
07-16-2015 16:35:01.123;74.01545;1;0.1587826;3.341217
07-16-2015 16:35:01.589;74.2103;1;0.1714313;3.328569
07-16-2015 16:35:02.096;74.40642;1;0.1824904;3.31751
07-16-2015 16:35:02.588;74.66607;1;0.1808253;3.319175
07-16-2015 16:35:03.094;74.92714;1;0.1536678;3.346332
07-16-2015 16:35:03.586;74.30122;1;0.1101428;3.389857
07-16-2015 16:35:04.089;73.91901;1;0.08086719;3.419133
07-16-2015 16:35:04.614;73.43941;1;0.05535573;3.444644
07-16-2015 16:35:05.085;73.89291;1;0.06704824;3.432952
07-16-2015 16:35:05.596;75.17346;1;0.1069052;3.393095
07-16-2015 16:35:06.091;76.42522;1;0.1496074;3.350393
07-16-2015 16:35:06.600;78.04;1;0.237626;3.262374
07-16-2015 16:35:07.091;76.89118;1;0.3004307;3.199569
07-16-2015 16:35:07.588;76.04844;1;0.3243027;3.175697
07-16-2015 16:35:08.142;74.27379;1;0.3213327;3.178667
07-16-2015 16:35:08.588;73.28867;1;0.2935183;3.206482

07-16-2015 16:35:09.104;72.85073;1;0.2501877;3.249812
07-16-2015 16:35:09.591;72.80711;1;0.2099039;3.290096
07-16-2015 16:35:10.089;72.90811;1;0.1498909;3.350109
07-16-2015 16:35:10.585;73.10148;1;0.09973498;3.400265
07-16-2015 16:35:11.086;73.75078;1;0.07215935;3.427841
07-16-2015 16:35:11.620;74.68555;1;0.09363262;3.406367
07-16-2015 16:35:12.088;75.35654;1;0.1866912;3.313309
07-16-2015 16:35:12.603;75.63979;1;0.3140869;3.185913
07-16-2015 16:35:13.089;74.84744;1;0.3372166;3.162783
07-16-2015 16:35:13.589;73.95691;1;0.2884434;3.211557
07-16-2015 16:35:14.087;73.38081;1;0.2441747;3.255825
07-16-2015 16:35:14.589;72.77707;1;0.2274717;3.272528
07-16-2015 16:35:15.112;71.90915;1;0.2385887;3.261411
07-16-2015 16:35:15.587;71.18968;1;0.2820367;3.217963
07-16-2015 16:35:16.129;70.35324;1;0.372149;3.127851
07-16-2015 16:35:16.589;69.97704;1;0.428924;3.071076
07-16-2015 16:35:17.087;70.33234;1;0.4179615;3.082038
07-16-2015 16:35:17.586;70.82662;1;0.3495479;3.150452
07-16-2015 16:35:18.095;71.51678;1;0.2296039;3.270396
07-16-2015 16:35:18.615;72.33038;1;0.09881201;3.401188
07-16-2015 16:35:19.088;72.85785;1;0.03241687;3.467583
07-16-2015 16:35:19.600;72.90878;1;-0.006944264;3.506944
07-16-2015 16:35:20.088;72.89656;1;0.006225114;3.493775
07-16-2015 16:35:20.595;73.04202;1;0.08001725;3.419983
07-16-2015 16:35:21.091;73.21689;1;0.2098665;3.290133
07-16-2015 16:35:21.587;73.34799;1;0.3464751;3.153525
07-16-2015 16:35:22.102;73.53501;1;0.5512572;2.948743
07-16-2015 16:35:22.587;73.68701;1;0.6914872;2.808513
07-16-2015 16:35:23.102;73.89626;1;0.8036758;2.696324
07-16-2015 16:35:23.588;74.09716;1;0.8662076;2.633792
07-16-2015 16:35:24.087;74.35213;1;0.9378805;2.562119
07-16-2015 16:35:24.588;74.59906;1;1.002051;2.497949
07-16-2015 16:35:25.086;74.77525;1;1.048454;2.451546
07-16-2015 16:35:25.615;74.83323;1;1.098051;2.401949
07-16-2015 16:35:26.085;74.90498;1;1.126625;2.373375
07-16-2015 16:35:26.604;74.39879;1;1.136781;2.363219
07-16-2015 16:35:27.088;73.94752;1;1.107143;2.392857
07-16-2015 16:35:27.620;73.465;1;1.007587;2.492413
07-16-2015 16:35:28.090;73.09799;1;0.8334543;2.666546
07-16-2015 16:35:28.586;72.85667;1;0.6499369;2.850063
07-16-2015 16:35:29.090;72.57633;1;0.3997876;3.100212
07-16-2015 16:35:29.585;72.76718;1;0.2589491;3.241051
07-16-2015 16:35:30.087;73.43745;1;0.2061054;3.293895
07-16-2015 16:35:30.585;73.95206;1;0.2597178;3.240282
07-16-2015 16:35:31.087;74.51722;1;0.4267466;3.073253
07-16-2015 16:35:31.588;75.07123;1;0.6236141;2.876386
07-16-2015 16:35:32.086;75.47222;1;0.7590779;2.740922
07-16-2015 16:35:32.608;75.98718;1;0.901145;2.598855
07-16-2015 16:35:33.085;76.4001;1;0.9895849;2.510415
07-16-2015 16:35:33.589;76.90533;1;1.061747;2.438253
07-16-2015 16:35:34.090;77.3129;1;1.089279;2.410721
07-16-2015 16:35:34.618;77.79985;1;1.066932;2.433068
07-16-2015 16:35:35.090;78.28208;1;0.9843815;2.515619
07-16-2015 16:35:35.588;78.49376;1;0.8906152;2.609385
07-16-2015 16:35:36.118;77.55988;1;0.7768198;2.72318
07-16-2015 16:35:36.588;76.75771;1;0.7185606;2.781439
07-16-2015 16:35:37.099;75.82007;1;0.7058133;2.794187
07-16-2015 16:35:37.586;75.09049;1;0.7444166;2.755583
07-16-2015 16:35:38.090;74.13287;1;0.8595737;2.640426
07-16-2015 16:35:38.587;73.16715;1;0.995021;2.504979
07-16-2015 16:35:39.096;72.47746;1;1.047336;2.452664
07-16-2015 16:35:39.620;71.56693;1;0.9962156;2.503784
07-16-2015 16:35:40.086;70.85923;1;0.8547469;2.645253
07-16-2015 16:35:40.603;69.9885;1;0.5738209;2.926179
07-16-2015 16:35:41.088;70.13328;1;0.3207796;3.17922
07-16-2015 16:35:41.587;70.79903;1;0.1041127;3.395887
07-16-2015 16:35:42.088;71.36922;1;0.03174896;3.468251
07-16-2015 16:35:42.585;71.96874;1;0.04836861;3.451631
07-16-2015 16:35:43.090;72.40858;1;0.1122471;3.387753
07-16-2015 16:35:43.590;72.55616;1;0.1453333;3.354667
07-16-2015 16:35:44.095;72.85925;1;0.1497879;3.350212
07-16-2015 16:35:44.591;73.69722;1;0.1442847;3.355715
07-16-2015 16:35:45.088;74.71445;1;0.1522575;3.347743
07-16-2015 16:35:45.585;75.64623;1;0.2287528;3.271247
07-16-2015 16:35:46.086;76.25302;1;0.3591689;3.140831
07-16-2015 16:35:46.616;76.80901;1;0.5234286;2.976571
07-16-2015 16:35:47.089;77.22196;1;0.5966023;2.903398
07-16-2015 16:35:47.593;77.72482;1;0.6068545;2.893146
07-16-2015 16:35:48.089;78.12866;1;0.5529117;2.947088
07-16-2015 16:35:48.594;77.70328;1;0.4205764;3.079424
07-16-2015 16:35:49.085;76.37543;1;0.292565;3.207435
07-16-2015 16:35:49.585;75.37298;1;0.244536;3.255464
07-16-2015 16:35:50.113;74.09407;1;0.2527615;3.247238
07-16-2015 16:35:50.587;73.09367;1;0.3181192;3.181881
07-16-2015 16:35:51.110;71.84496;1;0.447521;3.052479
07-16-2015 16:35:51.590;70.84702;1;0.5260829;2.973917
07-16-2015 16:35:52.096;69.57306;1;0.5028873;2.997113
07-16-2015 16:35:52.588;68.44124;1;0.3204305;3.179569
07-16-2015 16:35:53.102;67.48526;1;0.0283901;3.47161
07-16-2015 16:35:53.604;66.58222;1;-0.4844732;3.984473
07-16-2015 16:35:54.089;67.60925;1;-0.9062303;4.40623
07-16-2015 16:35:54.594;69.47368;1;-1.20725;4.70725
07-16-2015 16:35:55.090;71.1729;1;-1.13856;4.63856
07-16-2015 16:35:55.590;73.36613;1;-0.6964446;4.196445
07-16-2015 16:35:56.086;75.51199;1;-0.03005823;3.530058
07-16-2015 16:35:56.585;77.26296;1;0.6219153;2.878085
07-16-2015 16:35:57.108;79.20771;1;1.372356;2.127644
07-16-2015 16:35:57.589;80.33975;2;-0.0272132;0.1272132
07-16-2015 16:35:58.098;81.3016;2;0.1262765;-0.02627651
07-16-2015 16:35:58.592;82.05919;1;1.748608;1.751392
07-16-2015 16:35:59.090;82.8945;1;1.318422;2.181578
07-16-2015 16:35:59.588;82.19721;1;0.9167193;2.583281
07-16-2015 16:36:00.095;80.39634;1;0.6999387;2.800061
07-16-2015 16:36:00.641;78.22722;1;0.5895478;2.910452
07-16-2015 16:36:01.088;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:01.595;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:02.089;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:02.586;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:03.088;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:03.589;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:04.086;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:04.590;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:05.085;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:05.589;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:06.086;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:06.589;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:07.085;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:07.590;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:08.095;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:08.586;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:09.088;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:09.589;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:10.086;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:10.588;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:11.087;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:11.588;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:12.090;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:12.589;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:13.090;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:13.585;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:14.090;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:14.586;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:15.085;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:15.587;76.9761;1;0.5635968;2.936403
07-16-2015 16:36:16.086;76.9761;1;0.5635968;2.936403

Core Matlab script

Filename: readLogs.m

```
speedFname = 'Speedlog';
simFname = 'Touchpanel log';

% timeFname = '2015-07-16 160134';
% xLimits = [150,500];

% timeFname = '2015-07-16 161806';
% xLimits = [0,0];

timeFname = '2015-07-16 163156';
xLimits = [0,0];

suffixFname = '.txt';

initTime = datenum(timeFname,'yyyy-mm-dd HHMMSS');
speedName = [speedFname, ' ', timeFname, suffixFname];
simName = [simFname, ' ', timeFname, suffixFname];
speedTable = speedDataToTable(speedName,initTime);
speedHeader = {'elapsedTime','kmh','laneID','leftLanePos','rightLanePos'};
simTable = simDataToTable(simName,initTime);
simHeader = {'elapsedTime','type','arguments'};

figure()
subplot(2,1,1)
prop = struct;
prop.plotTitle = datestr(initTime,'yyyy-mm-dd HH:MM:SS');
prop.xLabel = 'Elapsed Time (s)';
prop.yLabel = 'Speed (km/h)';

listList = plotInCurrent(speedTable,simTable,2,xLimits,prop);
eventList = listList{1};
completedList = listList{2};
failedList = listList{3};

subplot(2,1,2)
prop = struct;
prop.plotTitle = '';
prop.xLabel = 'Elapsed Time (s)';
prop.yLabel = 'Lane position';

plotInCurrent(speedTable,simTable,4,xLimits,prop);
display(eventList)
display(completedList)
display(failedList)

clear prop
clear xLimits
clear listList
clear speedFname
clear simFname
clear timeFname
clear suffixFname
clear simName
clear speedName
clear speedX2
clear speedX1
clear speedY2
clear speedY1
clear simX
clear simY
clear speedX
clear speedIndex
clear i
clear simCount
clear listArgs
```

clear listDetails

Script to extract event data

Filename: simDataToTable.m

```
function inTable = simDataToTable(filename,initTime)

    permission = 'rt';
    machinefmt = 'n';
    encodingIn = 'UTF-8';

    fid = fopen(filename,permission,machinefmt,encodingIn);

    formatSpec = '%d-%d-%d %d:%d:%f;%2s;%s';
    %formatSpec = '%{MM-dd-yyyy HH:mm:ss}D;%2s;%s';

    inTable = textscan(fid,formatSpec,'Delimiter','\r\n');

    fclose(fid);

    inTable =
[num2cell(inTable{1,1}),num2cell(inTable{1,2}),num2cell(inTable{1,3}),num2cell(inTable{1,4}),num2cell(inTable{1,5}),num2cell(inTable{1
,6}),inTable{1,7},inTable{1,8}];

    inTable =
[num2cell(etime([double(cell2mat(inTable(:,3))),double(cell2mat(inTable(:,1))),double(cell2mat(inTable(:,2))),double(cell2mat(inTable(:,4
))),double(cell2mat(inTable(:,5))),double(cell2mat(inTable(:,6)))],datevec(ones(size(inTable,1),1)*initTime))),inTable(:,7),inTable(:,8)];

end
```

Script to extract speed and lane data

Filename: speedDataToTable.m

```
function inTable = speedDataToTable(filename,initTime)

    permission = 'rt';
    machinefmt = 'n';
    encodingIn = 'UTF-8';

    fid = fopen(filename,permission,machinefmt,encodingIn);

    formatSpec = '%d-%d-%d %d:%d:%f;%f;%d;%f;%f';
    sizeA = [10,inf];
    inTable = fscanf(fid,formatSpec,sizeA);
    fclose(fid);

    inTable = inTable';
    inTable =
[num2cell(etime([double(inTable(:,3)),double(inTable(:,1)),double(inTable(:,2)),double(inTable(:,4)),double(inTable(:,5)),double(inTable(:,
6))],datevec(ones(size(inTable,1),1)*initTime))),num2cell(inTable(:,7)),num2cell(inTable(:,8)),num2cell(inTable(:,9)),num2cell(inTable(:,1
0))];

end
```

Script to plot data

Filename: plotInCurrent.m

```
function [listList] = plotInCurrent(speedTable, simTable, speedYMatIndex, xLimits, properties)

    plot(cell2mat(speedTable(:,1)),cell2mat(speedTable(:,speedYMatIndex)))

    if length(properties.plotTitle) > 0
        title(properties.plotTitle)
    end
```

```
if length(properties.xLabel) > 0
   xlabel(properties.xLabel)
end

if length(properties.yLabel) > 0
   ylabel(properties.yLabel)
end

if (xLimits(2) > 0) && (xLimits(1) < xLimits(2))
   xlim(xLimits)
end

speedIndex = 1;
simCount = 0;
eventList = {};
completedList = {};
failedList = {};

hold on
for i = 1:size(simTable,1)

   speedX = speedTable{speedIndex,1};
   simX = simTable{i,1};

   while speedX <= simX
      speedIndex = speedIndex + 1;
      speedX = speedTable{speedIndex,1};
   end

   if strcmp('EV',simTable(i,2)) || strcmp('MC',simTable(i,2))
      if speedIndex == 1
         simY = 0;
      else
         speedX2 = speedX;
         speedX1 = speedTable{speedIndex-1,1};

         speedY2 = speedTable{speedIndex,speedYMatIndex};
         speedY1 = speedTable{speedIndex-1,speedYMatIndex};

         simY = speedY2 + (speedY1 - speedY2)*(speedX2-simX)/(speedX2-speedX1);
      end

      if strcmp('EV',simTable(i,2))
         simCount = simCount + 1;
         listArgs = strsplit(simTable{i,3},';');
         listDetails = strsplit(listArgs{1,2},'|');

         if strcmp('TaskStarted',listArgs(1,1))
            eventList = [eventList;[listDetails{1,2},': ',listDetails{1,1},' (task)', ' (',num2str(simX),'s)']];
            if (simX > xLimits(1) && simX < xLimits(2)) || (xLimits(2) == 0)
               plot(simX,simY,'s', 'MarkerFaceColor', [1.0,0.687,0], 'Color', [1.0,0.687,0])
               text(simX,simY,['  ',listDetails(1,2)])
            end
         elseif strcmp('TaskFailed',listArgs(1,1))
            failedList = [failedList;[listDetails{1,2},': ',listDetails{1,1},' (task)', ' (',num2str(simX),'s)']];
            if (simX > xLimits(1) && simX < xLimits(2)) || (xLimits(2) == 0)
               plot(simX,simY,'sr', 'MarkerFaceColor', 'r')
               text(simX,simY,['  ',listDetails(1,2)])
            end
         elseif strcmp('TaskCompleted',listArgs(1,1))
            completedList = [completedList;[listDetails{1,2},': ',listDetails{1,1},' (task)', ' (',num2str(simX),'s)']];
            if (simX > xLimits(1) && simX < xLimits(2)) || (xLimits(2) == 0)
               plot(simX,simY,'sg', 'MarkerFaceColor', 'g')
               text(simX,simY,['  ',listDetails(1,2)])
            end
         elseif strcmp('AlarmTriggered',listArgs(1,1))
            eventList = [eventList;[listDetails{1,2},': ',listDetails{1,1},' (alarm)', ' (',num2str(simX),'s)']];
            if (simX > xLimits(1) && simX < xLimits(2)) || (xLimits(2) == 0)
```

```
                plot(simX,simY,'*', 'MarkerFaceColor', [1.0,0.687,0], 'Color', [1.0,0.687,0])
                text(simX,simY,['   ',listDetails(1,2)])
            end
        elseif strcmp('AlarmHandled',listArgs(1,1))
            if (simX > xLimits(1) && simX < xLimits(2)) || (xLimits(2) == 0)
                plot(simX,simY,'*g', 'MarkerFaceColor', 'g')
                text(simX,simY,['   ',listDetails(1,2)])
            end
        end

      elseif strcmp('MC',simTable(i,2))
        if (simX > xLimits(1) && simX < xLimits(2)) || (xLimits(2) == 0)
            plot(simX,simY,'xb')
        end
      end

    end

  end
  hold off

  listList = {eventList,completedList,failedList};

end
```

Software member reference

This section goes into detail of each of the members of each of the modules to explain the inner functionality of the software. In the reference lists, the members are stated in the order they appear in the code. In most cases, this means that the most relevant methods are at the top.

Members of MainForm

This section describes the different members contained in the module and what their code does.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

MainForm
Public Class MainForm
This is the base class of the module and it hosts all other members of the module.
In addition to that, it also implements the IMessageFilter interface. It also implements the PreFilterMessage interface function. The base class hosts the publisher and the subscriber for the UDP transmissions, which is where the local port is set. The target IP and port are set in the textboxes on the form. It also hosts a number of variables:

- the names and directories of the folders
- the start time of the software,
- the list of tasks
- the ID counter for the events
- the file endings for the different types of supported script and sound files

New
Public Sub New()
The constructor of the MainForm. It initializes the global mouse clicking detection by adding a so called Message Filter. The actual filtering is handled in a separate method "PreFilterMessage" described later in this section.

OnFormClose
Protected Overrides Sub OnFormClosed(ByVal e As System.Windows.Forms.FormClosedEventArgs)
This method triggers when the form closes. It removes the message filter that was added by the constructor.

PreFilterMessage
Public Function PreFilterMessage(ByRef m As System.Windows.Forms.Message) As Boolean Implements System.Windows.Forms.IMessageFilter.PreFilterMessage
This method then implements the PreFilterMessage interface function.
The message filter, that this method provides, lets the software interrupt so called Windows messages and read them or modify them before they are being used normally in the software.
Among the windows messages are the messages the mouse sends to the software when it's clicked, the coordinates of it and what was clicked. This is read and logged with the method WriteMouseLog and a call is made to the methods ReportClickToTask and ReportClickToAlarm to give the information to the TaskManager and AlarmManager.
The method returns the message, which in this case is unmodified, so that the software can continue using it.

MainForm_Load
Private Sub MainForm_Load(sender As Object, e As EventArgs) Handles Me.Load
This method is called when the form is loaded.
The method first creates the folder structure in the root folder of the software unless the folders already exist. A timestamp is recorded as a starting time for the software to be used for the name of the logs. The standard log is named here and a variable, logNameAndPath, containing its name and path is created so that other modules can access it if needed.
The SoundManager and VariableManager are then initiated and started.
The UDP connection and its timer is initiated as well as the UDP faker, which exists for debugging purposes for the designer.
Then the MainUI form, the main user interface that the test driver sees, is initiated and shown.
The Utilities module and the AlarmManager and TaskManager are also initiated.
Lastly a log entry is made, via the method WriteSystemLog, that the system is started.

TimUDP_Tick
Private Sub TimUDP_Tick(sender As Object, e As EventArgs) Handles TimUDP.Tick
This method is called each time the UDP timer triggers. It tells the UDPTools module to send a number of variables specified in and handled by the VariableManager via UDP to the simulator. The method also triggers a reading of a number of variables that has been received from the simulator via UDP to the UDPTools module and forwards them to the VariableManager.

WriteToLog
Shared Sub WriteToLog(ByVal LogString As String)
This method is used to write text to the standard log file.
It writes text on the following syntax, using the input parameter LogString:
[Current timestamp];LogString

WriteToCustomLog
Shared Sub WriteToCustomLog(ByVal logNameAndPathCustom As String, LogString As String)
This method is used to write text to a custom log file, defined by the input parameter logNameAndPathCustom.
It writes text on the following syntax, using the input parameter LogString:
[Current timestamp];LogString

WriteMouseLog
Shared Sub WriteMouseLog(ByVal ControlPathAndName As String, ByVal PosX As Integer, ByVal PosY As Integer)
This method works as a wrapper for the WriteToLog method, specifically writing mouse events to the log. It receives the clicked control's name and path and the location of the mouse as parameters and writes them to the log on the following syntax:
[Current timestamp];MC;PosX:PosY;ControlPathAndName
Where "MC" is a tag that signifies that it's a mouse click.

WriteEventLog
Shared Sub WriteEventLog(ByVal strEvent As String, ByVal strDetails As String)
This method works as a wrapper for the WriteToLog method, specifically writing events to the log mainly from the TaskManager or AlarmManager. It receives the name of the event and the details regarding the event as parameters and writes them to the log on the following syntax:
[Current timestamp];EV;strEvent;strDetails
Where "EV" is a tag that signifies that it's an event.

WriteSystemLog
Shared Sub WriteSystemLog(ByVal strDetails As String)
This method works as a wrapper for the WriteToLog method, specifically writing system information to the log. It receives the details about the entry as parameters and writes it to the log on the following syntax:
[Current timestamp];SY;strDetails
Where "SY" is a tag that signifies that it's a system message.

ReportClickToTask
Private Sub ReportClickToTask(ByVal ControlName As String, ByVal PosX As Integer, ByVal PosY As Integer)
This method works as a wrapper and calls the method reportClick in the TaskManager module.

ReportClickToAlarm
Private Sub ReportClickToAlarm(ByVal ControlName As String, ByVal PosX As Integer, ByVal PosY As Integer)
This method works as a wrapper and calls the method reportClick in the AlarmManager module.

InitiateTasks
Private Sub InitiateTasks()
This method finds and lists the files in the Tasks folder. Then, for each of them that has the correct file ending, creates a new instance of the TaskManager with the file as an input parameter for the constructor. The method then lists them in the list taskList in the MainForm module. This process initiates all the tasks from the scripts in the Tasks folder.

GetActionID
Shared Function GetActionID() As Integer
This method is called from the alarms and tasks for them to receive a unique ID. The IDs are sequential and incremented for each call. The ID is visible in the log entries created by the alarms and tasks so that they can be uniquely identified.
This method returns the generated ID.

objectExistInUI
Public Function objectExistInUI(ByVal objectName As String) As Boolean
This method takes a name as an input parameter and recursively searches through the objects in the MainUI, the user interface, to check if an object with the given name exists. This method calls the method objectRecursiveExist for all found objects in order to create the recursive search. The reason it searches recursively is because some objects may be containers that contain other objects. This method is useful in order to check that the scripts made by the designer are made correctly so that all objects stated in the scripts actually exist.
This method returns True if the object exists, otherwise False.

objectRecursiveExist
Private Function objectRecursiveExist(ByVal objectName As String, ByRef controlContainer As Object) As Boolean
This method is a helper method to objectExistInUI in order to make the process recursive. In addition to a name, it also takes a container object as an input parameter. objectExistInUI calls this method for all found objects in order to search recursively for objects with the given name on the MainUI form and then this method calls itself for all found objects in order to search recursively.
This method returns True if the object exists, otherwise False.

cbTraffic_CheckedChanged
Private Sub cbTraffic_CheckedChanged(sender As Object, e As EventArgs) Handles cbTraffic.CheckedChanged
This method is called when the checked status of the check box, on the MainForm, for starting traffic is changed (from clicking on it for example). It reads whether the check box is checked or not and sets the variable StartTraffic, in the VariableManager, to 1 or 0 accordingly. The variable is later sent to the simulator to start or stop the traffic in the simulation.

Members of UDPTools

This section describes the different members contained in the module and what their code does.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

UDPTools
Public Class UDPTools
This is the base class of the module and it hosts all other members of the module.

sendUDP
Shared Sub sendUDP(ByVal IP As String, ByVal Port As Integer, ByVal Packet() As Byte, ByRef UDPpublisher As Sockets.UdpClient)
This method takes the IP address and port of the target as input parameters and uses them to address and send a UDP packet. The packet as well as the UDPClient that is set up to be the sender, the publisher, are also input parameters. The method createSngPacket can be used to create the packet of information out of variables of the single type to be sent, however any packet of data could be sent.

receiveUDP

Shared Function receiveUDP(ByRef UDPsubscriber As Sockets.UdpClient) As Byte()

This method reads the received packet buffer, using the specified UDPClient in the input parameter to do so.
The method returns the read series of bytes that is the packet that has been received via UDP, if any. If no packet has been received, the method returns Nothing, a kind of null value.

createSngPacket

Shared Function createSngPacket(ByVal Vars() As Object) As Byte()

This method takes an array of numeric variables as the input parameter, turns them into single type variables and serializes them as an array of bytes, which is a packet that can be sent via UDP.
The method returns the created packet.

readSngPacket

Shared Function readSngPacket(ByVal bytes() As Byte) As Single()

This method takes a byte array, a packet, as an input parameter and deserializes it into as many single type variables as it can find in the array. The byte array must only contain serialized single type variables or the output will be faulty.
The method returns an array of the deserialized single type variables.

arrMod

Shared Function arrMod(ByVal source() As String, ByVal target() As String, ByVal startIndex As Integer) As String()
Shared Function arrMod(ByVal source() As Integer, ByVal target() As Integer, ByVal startIndex As Integer) As Integer()
Shared Function arrMod(ByVal source() As Double, ByVal target() As Double, ByVal startIndex As Integer) As Double()
Shared Function arrMod(ByVal source() As Long, ByVal target() As Long, ByVal startIndex As Integer) As Long()
Shared Function arrMod(ByVal source() As Single, ByVal target() As Single, ByVal startIndex As Integer) As Single()
Shared Function arrMod(ByVal source() As Byte, ByVal target() As Byte, ByVal startIndex As Integer) As Byte()
Shared Function arrMod(ByVal source() As Char, ByVal target() As Char, ByVal startIndex As Integer) As Char()
Shared Function arrMod(ByVal source() As Boolean, ByVal target() As Boolean, ByVal startIndex As Integer) As Boolean()
Shared Function arrMod(ByVal source() As Object, ByVal target() As Object, ByVal startIndex As Integer) As Object()

This method can be called in one of several ways depending on which type of variable is used as an input. If for example an integer is used as an input parameter, the integer version of the method is called. The method takes two arrays as input parameters, source and target. It takes the source array and then, step by step, replaces each entry, starting at startIndex, with the entries in the target array. This may result in a larger output array than the source.
The method returns the source array with the entries replaced as described.

Members of UDPFaker

This section describes the different members contained in the module and what their code does.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

UDPFaker
Public Class UDPFaker
This is the base class of the module and it hosts all other members of the module.
In addition to this, the base class also hosts a variable for each of the textboxes on the form in order to be able to save the value while the textbox is being modified, so that the last legitimate value can be used. Whereas a textbox with a non-numeric string or an empty textbox would cause errors if it was to be used to set VariableManager variables directly.

UDP_Faker_Load
Private Sub UDP_Faker_Load(sender As Object, e As EventArgs) Handles MyBase.Load
This method triggers when the module loads and starts the timer UpdateState that controls when the variables are set.

UpdateState_Tick
Private Sub UpdateState_Tick(sender As Object, e As EventArgs) Handles UpdateState.Tick
This method is called when the timer UpdateState triggers. First it updates the textboxes on the form with the values of the variables the designer wants shown. Then, if the checkbox CBFakeOn is checked, it updates, in the VariableManager, the set of variables that are chosen to be simulated with values from the textboxes on the form of the module. The variables aren't updated from the textboxes directly, but instead from an internal list of variables in the module. When changing the text in the textboxes, the variables in the list change accordingly with the [Control]_TextChanged methods.

[Control]_TextChanged
Private Sub TBPosIne_TextChanged(sender As Object, e As EventArgs) Handles TBPosIne.TextChanged
Private Sub TBVel_TextChanged(sender As Object, e As EventArgs) Handles TBVel.TextChanged
Private Sub TBRPM_TextChanged(sender As Object, e As EventArgs) Handles TBRPM.TextChanged
Private Sub TBSteerAngle_TextChanged(sender As Object, e As EventArgs) Handles TBSteerAngle.TextChanged
Private Sub TBleftLaneDist_TextChanged(sender As Object, e As EventArgs) Handles TBleftLaneDist.TextChanged
Private Sub TBrightLaneDist_TextChanged(sender As Object, e As EventArgs) Handles TBrightLaneDist.TextChanged
Private Sub TBlaneID_TextChanged(sender As Object, e As EventArgs) Handles TBlaneID.TextChanged
Private Sub TBvehicleRadar_TextChanged(sender As Object, e As EventArgs) Handles TBvehicleRadar.TextChanged
Private Sub TBsimTimer_TextChanged(sender As Object, e As EventArgs) Handles TBsimTimer.TextChanged
This type of method trigger when the text in the textbox is changed. The method sets the variables from the text boxes to the internal list of variables in the module, if the new value is a valid numeric value. These variables are then used in the UpdateTimer_Tick in order to set the variables in the VariableManager. This ensures that the variables aren't used directly from the textboxes, which in turn allows the last valid input to be saved and used in case the new inputs are invalid.

[Control]_CheckedChanged
Private Sub CBSpeLim_CheckedChanged(sender As Object, e As EventArgs) Handles CBSpeLim.CheckedChanged
This type of method trigger when the checked status of a checkbox is changed. The method sets the variables from the checkboxes to the internal list of variables in the module. These

variables are then used in the UpdateTimer_Tick in order to set the variables in the VariableManager. This is done purely for consistency with the [Control]_TextChanged methods as a boolean input cannot be invalid when using a checkbox to set it.

Members of Utilities

This section describes the different members contained in the module and what their code does.
To shorten the description on some of the methods and to make them more easily readable, a table over requirements and the error message that is the consequence if the requirements aren't met will replace a descriptive text. Any error will, in addition to showing an error message, also close the software.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

Utilities
Public Class Utilities
This is the base class of the module and it hosts all other members of the module.
In addition to this, the base class also hosts a variable for the UpdateTimer updating interval.

condition
```
Public Class condition
    Public type As String
    Public info1 As String
    Public info2 As String
End Class
```
This class is used to store information about conditions for tasks and alarms.

New
Public Sub New()
This method is the constructor for this module. It starts the UpdateTimer that controls the update rate for the VariableManager and AlarmManager

taskUpdateTimer_Tick
Private Sub taskUpdateTimer_Tick(sender As Object, e As EventArgs) Handles UpdateTimer.Tick
This method is called when the timer UpdateTimer triggers. It calls the methods called updateTimerTick in both TaskManager and AlarmManager. Those methods handle the updating of the tasks and alarms.

readTextFromTaglist

Public Function readTextFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As String

Requirement	Error
Taglist size 2	ShowParamNumError

If called without errors, the method will return a string that is the value described by the second entry of the taglist.
tagList structure:

0. Identifier
1. Value of the string

readBoolFromTaglist

Public Function readBoolFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As Boolean

Requirement	Error
Taglist size 2	ShowParamNumError
Parameter 2 is a boolean	ShowBoolError

If called without errors, the method will return a boolean that is the value described by the second entry of the taglist.
tagList structure:

0. Identifier
1. Value of the boolean

readVarnameAndNumberFromTaglist

Public Function readVarnameAndNumberFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As condition

Requirement	Error
Taglist size 3	ShowParamNumError
Parameter 2 is the name of a variable in the VariableManager	ShowUnknownVarError
Parameter 3 is a number	ShowNotNumericVarError

If called without errors, this method will return a condition type variable. It contains the type of condition, which is the identifier of the taglist, the name of the variable as info1 and value of the variable as info2.
tagList structure:

0. Identifier
1. Name of the variable
2. Value of the variable

readVarnameFromTaglist

Public Function readVarnameFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As condition

Requirement	Error
Taglist size 2	ShowParamNumError
Parameter 2 is the name of a variable in the VariableManager	ShowUnknownVarError

If called without errors, this method will return a condition type variable. It contains the type of condition, which is the identifier of the taglist and the name of the variable as info1 and a Nothing value as info2.
tagList structure:

0. Identifier
1. Name of the variable

readClickConditionFromTaglist

Public Function readClickConditionFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As condition

Requirement	Error
Taglist size 2	ShowParamNumError
Parameter 2 is the name of an object in mainUI	ShowUnknownObjError

If called without errors, this method will return a condition type variable. It contains the type of condition, which is the identifier of the taglist and the name of the object as info1 and a Nothing value as info2.
tagList structure:

0. Identifier
1. Name of the object

readTimerConditionFromTaglist

Public Function readTimerConditionFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As condition

Requirement	Error
Taglist size 2	ShowParamNumError
Parameter 2 is a number	ShowTimerNotNumError
Parameter 2 is positive	ShowTimerNotPosError

If called without errors, this method will return a condition type variable. It contains the type of condition, which is the identifier of the taglist and the number as info1 and a Nothing value as info2. This condition describes a timer condition and its value.
tagList structure:

0. Identifier
1. Value of the timer

readControlFromTaglist
Public Function readControlFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As String

Requirement	Error
Taglist size 2	ShowParamNumError
Parameter 2 is the name of an object in mainUI	ShowUnknownObjError

If called without errors, this method will return the name of a control as a string.
tagList structure:
0. Identifier
1. Name of the control

readSoundFromTaglist
Public Function readSoundFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal fileNameAndDir As String) As String

Requirement	Error
Taglist size 2	ShowParamNumError
Parameter 2 is the name of a sound in the SoundManager	ShowSoundExistError

If called without errors, this method will return the name of a sound as a string.
tagList structure:
0. Identifier
1. Name of the sound

checkOneConditionHolds
Public Function checkOneConditionHolds(ByVal conditionList As List(Of condition)) As Boolean
This method takes a list of conditions as an input and checks if at least one condition holds true based on the current variables in the VariableManager. A condition in this case may for example be varLess, Speed, 100 which would mean that the condition holds true if the variable Speed is less than 100.
The method returns true if at least one condition holds and false otherwise.

checkAllConditionsHolds
Public Function checkAllConditionsHolds(ByVal conditionList As List(Of condition)) As Boolean
This method takes a list of conditions as an input and checks if all conditions hold true based on the current variables in the VariableManager. A condition in this case may for example be varLess, Speed, 100 which would mean that the condition holds true if the variable Speed is less than 100.
The method returns true if all conditions hold and false otherwise.

eventSetVars
Public Shared Sub eventSetVars(ByVal varList As List(Of Utilities.condition))
This method takes a list of conditions as an input and sets all variables in the VariableManager as described by the commands in the list. A command in this case may for example be addVar, Counter, 3 which would add 3 to the variable Counter. Commands use the same structure as a

condition and are placed in the condition lists in the AlarmManager and TaskManager. This is in order to be able to check the whole list at once without the need for filtering.

Members of VariableManager

This section describes the different members contained in the module and what their code does.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

VariableManager
Public Class VariableManager
This is the base class of the module and it hosts all other members of the module.

simVariable
Class simVariable
Public name As String
Public value As Single
End Class
A name and value pair that makes up a variable in the VariableManager.

initVariables
Shared Sub initVariables()
This method initiates all the variables with a name and a value. This is where the designer creates the variables to be used by using the method createVariable. The variables can be created at other places in the software, but doing it in this method ensures that the variables are created before the scripts are read because of the start order of the modules. This ensures that the variable is ready for use when the validation and error handling is done for the scripts.

createVariable
Shared Sub createVariable(ByVal name As String, ByVal value As Single)
This method creates a new variable by assigning a name and a value to a simVariable type variable and adding it to the variableList list which contains all the variables in the VariableManager.

setUDPVar
Shared Sub SetUDPVar(ByVal VarArray() As Single)
This method takes an array of single type variables as an input. This method is called from the UDP timer in the MainForm or the UDP faker and sets the variables in the VariableManager each time new variable values are received via UDP. The variables in the array are identified by their order.
The designer adds a setVar method call for each variable to be set in this method and connects them to the correct indices of the array.

readUDPVar
Shared Function readUDPVar() As Object()
This method is called by the UDP timer in the MainForm each time variable values are to be sent via UDP. The variables in the array are identified by their order.
The designer adds an entry in the array for each variable to be sent in this method and connects them with a readVar method for that variable.
The method returns an array of Object type variables to be sent via UDP.

setVar
Shared Sub setVar(ByVal name As String, ByVal value As Single)
Shared Sub setVar(ByVal varlist As List(Of simVariable))
There are two versions of this method, one takes a name and a value as an input and the other takes a list of simVariable type variables as an input. A simVariable is a name and value pair. The first method sets a single variable and the second method sets many variables at once. They both work the same way only that the second one loops through the list of simVariables and extracts the name and value from each entry.
The methods searches for a variable in the VariableManager with the given name and sets the variable with that name to the given value if it can find the variable in the VariableManager. If it doesn't find the variable, it generates an error, ShowVarNotExist and the software closes. If the method succeeds, it also calls the updateState methods in the TaskManager and AlarmManager when it's done, which triggers an update in the TaskManager and AlarmManager based on the new variables.

readVar
Shared Function readVar(ByVal name As String) As Object
This method takes a name as an input parameter and reads the value of the variable with that name if it finds it. If it doesn't find the variable, it generates an error, ShowVarNotExist and the software closes.
The method returns an Object type variable with the value of the named variable.

varExists
Shared Function varExists(ByVal name As String) As Boolean
This method takes a name as an input parameter and searches the VariableManager for a variable with that name.
The method returns True if the variable is found or False otherwise.

Members of TaskManager

This section describes the different members contained in the module and what their code does.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

TaskManager

Public Class TaskManager

This is the base class of the module and it hosts all other members of the module.
In addition to this, the base class also hosts variables to run the module and the variables that store the tasks in each of the instances of the module:

- taskName
- taskID
- taskRunning
- Triggers
- Fails
- Ends
- actionList
- TriggerSetVar
- FailsSetVar
- EndsSetVar
- TriggerHasClick
- FailHasClick
- EndHasClick
- TriggerTimerDone
- FailTimerDone
- EndTimerDone
- HasActionList
- actionListDone
- actionListCounter
- TriggerOnce
- HasTriggered
- triggerTimerCounter
- failTimerCounter
- endTimerCounter
- TriggerTimerTarget
- FailTimerTarget
- EndTimerTarget
- timerInterval

New

Public Sub New(ByVal fileNameAndDir As String)

The constructor for the module. The constructor takes a file name with included directory as an input and loads the task script file with that name into the memory of that instance of the module. See The script reading process section (p. 14).

resetTask
Private Sub resetTask()
This method resets the task to its initial state before it was started, with counters and flags reset.
If TriggerOnce is not true for the task, then it is ready to restart again if the trigger conditions are met once more.

StartTask
Private Sub StartTask()
This method starts the task if it is the first time it's supposed to trigger or if it's allowed to trigger more than once.
The method requests a new ID from MainForm via the method GetActionID. Then the method modifies the variables that are supposed to be modified when the task starts, as written in the script for the task. A log entry is made that the task has started and the flag variable taskRunning is set to true.

EndTask
Private Sub EndTask()
This method ends the task, completing it.
It sets the flag variable taskRunning to false and then writes a log entry that the task has been completed. Then the method modifies the variables that are supposed to be modified when the task ends, as written in the script for the task. The method ends by calling the resetTask method, resetting the task.

FailTask
Private Sub FailTask(ByVal reason As String)
This method ends the task by failing it.
It sets the flag variable taskRunning to false and then writes a log entry that the task has been failed, along with the reason stated in the input parameter. Then the method modifies the variables that are supposed to be modified when the task fails, as written in the script for the task. The method ends by calling the resetTask method, resetting the task.

updateTimerTick
Public Shared Sub updateTimerTick()
Wrapper method that is called by the updateTimer in the Utilities module in. This method calls the updateTimerLocal method in all instances of this module.

updateTimerLocal
Private Sub updateTimerLocal()
Wrapper method that calls the method UpdateTimerCounters, which updates the counters for the timers. If the flag variable updateWithTimer is true, that is, if the module is set to update states based on a timer, then this method also calls the methods checkTriggerVars, checkFailsVars, checkEndsVars and checkActionListVars, which handle the updating of the state based on the variables in VariableManager.

updateState
Public Shared Sub updateState()
Wrapper method that calls updateStateLocal in all instances of this module if the flag variable updateOnDemand is set to true, that is, if the module is set to update states when any variable is changed in the VariableManager.

updateStateLocal
Private Sub updateStateLocal()
Wrapper method that calls the methods checkTriggerVars, checkFailsVars, checkEndsVars and checkActionListVars, which handle the updating of the state based on the variables in VariableManager.

UpdateTimerCounters
Private Sub UpdateTimerCounters()
This method updates the counters for the timers. The actual timing is done by the updateTimer in the Utilities module and for each trigger of that timer, this method is called. This method adds the timer interval to a counter for each call and that stores the time that has passed. If a timer has reached its target, a flag variable is set to true to show that the timer is done for when the next state update is. The trigger timer starts when the software starts and the fail and end timers start when the task starts. This method also calls the updateStateLocal method to allow for updating the state.

checkTriggerVars
Private Sub checkTriggerVars()
This method checks the variable conditions for triggering the task.
It checks that

- the task is not running
- that it does not have a trigger click condition and
- that it either does not have a trigger timer or the timer is completed

If this holds, then it uses the method checkAllConditionsHolds in the Utilities module, with the list of trigger conditions as an input, to check if all conditions for triggering the task has been met. If so, the task is triggered by calling the StartTask method. Otherwise nothing happens.

checkFailsVars
Private Sub checkFailsVars()
This method checks the variable conditions for failing the task.
It checks that

- the task is running.

If this holds, it then checks if any of the conditions to fail the task has been met and if so, fails the task by calling the method FailTask with the reason for failing as an input parameter. This method does not use any Utility module method to check if the fail conditions hold as it needs to use the condition info to give the reason for failing. However the method works similar to how the Utility module method checkOneConditionHolds works in that regard. It checks only that one condition holds as opposed to that all conditions hold.

checkEndsVars
Private Sub checkEndsVars()
This method checks the variable conditions for ending the task, completing it.
It checks that
- the task is running
- that it does not have an end click condition
- that it either does not have an end timer or the timer is completed and
- that it either does not have an actionlist or that the actionlist has been completed

If this holds, then it uses the method checkAllConditionsHolds in the Utilities module, with the list of end conditions as an input, to check if all conditions for triggering the task has been met. If so, the task is ended, and completed, by calling the EndTask method. Otherwise nothing happens.

checkActionListVars
Private Sub checkActionListVars()
This method checks the variable conditions, stepwise, for the actionlist.
It checks that
- the task is running
- that it does have an actionlist and
- that the actionlist has not been completed

If this holds, then the method checks if the next condition in the actionlist holds or if the next entry in the actionlist is a command rather than condition, it manipulates a variable according to the command, for example addVar.
If the next entry is a condition, and it holds, then the counter for keeping track of the actionlist position is advanced and a log entry is made, stating that the entry in the actionlist is completed. If the above doesn't hold, nothing happens.
If the actionlist in itself is completed after this, a new log entry is made, stating that and a flag variable is set to mark it for completion.
At the end, the updateState method is then called, in case the completion of the actionlist or a variable manipulated by the actionlist means that the end conditions all hold or another condition's status has changed.

reportClick
Public Shared Sub reportClick(ByVal controlName As String)
Wrapper method that calls the method LocalReportClick for all instances of the module.
This method is called by the ReportClickToTask method in MainForm when a mouse click on a control is detected.

LocalReportClick
Private Sub LocalReportClick(ByVal controlname As String)
Wrapper method that calls the methods clickActionlist, clickEnds, clickFails and clickTrigger.
These methods handle the click conditions for the different states of the task.

clickTrigger
Private Sub clickTrigger(ByVal controlname As String)
This method checks the click condition for triggering the task.
It checks that

- the task is not running
- that it either does not have a trigger timer or the timer is completed and
- that it has a trigger click condition

If this holds, it loops through the conditions to find the click condition. When the click condition is found, the method checks if the clicked control, in the input parameter, matches that of the condition. If it does, the method then proceeds with checking if all other conditions hold with the Utility module method checkAllConditionsHold and if they do, the task is started by calling the StartTask method.

clickFails

Private Sub clickFails(ByVal controlname As String)

This method checks the click condition for failing the task.
It checks that

- the task is running and
- it has a fail click condition.

If this holds, it loops through the conditions to find a click condition. When a click condition is found, the method checks if the clicked control, in the input parameter, matches that of the condition. If it does, the method then immediately fails the task by calling the FailTask method with a string as the input parameter, stating it was failed because the control was clicked.

clickEnds

Private Sub clickEnds(ByVal controlname As String)

This method checks the click condition for ending the task, completing it.
It checks that

- the task is running
- that it has an end click condition
- that it either does not have an end timer or the timer is completed and
- that it either does not have an actionlist or the actionlist is completed

If this holds, it loops through the conditions to find the click condition. When the click condition is found, the method checks if the clicked control, in the input parameter, matches that of the condition. If it does, the method then proceeds with checking if all other conditions hold with the Utility module method checkAllConditionsHold and if they do, the task is ended by calling the EndTask method.

clickActionlist
Private Sub clickActionlist(ByVal controlname As String)
This method checks if the actionlist should be progressed or not.
It checks that

- the task is running
- that it has an actionlist and
- that the actionlist is not done

If this holds, the method reads the current condition from the actionlist and checks if it's a click condition and if it is, it checks if the clicked control, in the input parameter, matches that of the current condition. If it does, the actionlist is then advanced and a log entry is made. Then, if the actionlist is completed, a new log entry is made and a flag variable is set to mark it for completion and the method updateState is called in case the completion of the actionlist makes the conditions to end the task hold.

Members of AlarmManager

This section describes the different members contained in the module and what their code does.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

AlarmManager
Public Class AlarmManager
This is the base class of the module and it hosts all other members of the module.
In addition to this, the base class also hosts variables to run the module as well as the list of the alarms.

AlarmItem
Public Class AlarmItem
This is a storage class, utilizing other classes to build a tree in which an entire alarm is stored. The class has a constructor that populates a new instance of the class with standard values. The structure of the class can be viewed in the bullet list below

- name
- triggerOnce
- triggerVariables
- triggerClicks
- triggerTimerTarget
- triggerSetVar
- confirmationRules
    - confirmationNeeded
    - confirmWithMessagebox
    - confirmWithControl
    - confirmWithCondition
    - confirmationVariableList

    - confirmationClickList
    - confirmationSetVar
- handlingRules
    - handlingNeeded
    - handlingWithControl
    - handlingWithCondition
    - handlingVariableList
    - handlingClickList
    - handlingSetVar
- showRules
    - soundRules
        - useSound
        - sound
        - doLoop
    - messageBoxRules
        - useMessageBox
        - text
        - caption
- status
    - triggered
    - confirmed
    - alarmID
    - soundID
    - triggerTimer
    - hasTriggered

confirmRule
Public Class confirmRule
This is a storage class used by the AlarmItem class to generate the storage tree.

handlingRule
Public Class handlingRule
This is a storage class used by the AlarmItem class to generate the storage tree.

showRule
Public Class showRule
This is a storage class used by the AlarmItem class to generate the storage tree.

alarmStatus
Public Class alarmStatus
This is a storage class used by the AlarmItem class to generate the storage tree.

messageBoxRule
Public Class messageBoxRule
This is a storage class used by the AlarmItem class to generate the storage tree.

soundRule
Public Class soundRule
This is a storage class used by the AlarmItem class to generate the storage tree.

initializeAlarmSystem
Shared Sub initializeAlarmSystem()
Wrapper method handling the initialization of the alarms.
It's called by the MainForm module when it loads.
The method calls the method findAndAddAlarms.

findAndAddAlarms
Private Shared Sub findAndAddAlarms()
This method searches the alarm folder for alarm scripts and utilizes other methods to load them into the memory. See The script reading process (p. 18) for more information.

readAlarm
Private Shared Function readAlarm(ByVal fileNameAndDir As String) As AlarmItem
This method reads the script with the name from the input parameter and returns it as an AlarmItem. See The script reading process (p. 18) for more information.

cleanupAlarm
Private Shared Sub cleanupAlarm(ByRef alarm As AlarmItem)
This method takes a reference to an AlarmItem as an input parameter and cleans that AlarmItem up, removing minor erroneous entries to it and warning the designer of it by writing the errors to the log.
An example would be that the designer had scripted the alarm to not show a message box, but even so had made a caption and a text for the message box. The text and caption are then removed and a log entry is made, describing the issue. The software doesn't close. See The script reading process (p. 18) for more information.

reportClick
Public Shared Sub reportClick(ByVal controlName As String)
This method is called by the ReportClickToTask method in MainForm when a mouse click on a control is detected. This method handles the cases for when an alarm should be triggered, handled or confirmed when clicked.
The method loops through all of the alarms and for each of them checks the status and settings of the alarm to determine if it should be updated.
To trigger an alarm, the alarm must:

- not be triggered and
- have a click condition for triggering, where the clicked control matches the condition

To confirm an alarm, the alarm must:

- be triggered
- not be confirmed
- be confirmable with a control and
- have a click condition for confirming, where the clicked control matches the condition

To handle an alarm, the alarm must:

- be triggered

- be confirmed (or not needing to be confirmed, thus automatically confirming it)
- be handle-able with a control and
- have a click condition for handling, where the clicked control matches the condition

The triggering, confirming and handling is done by calling the methods triggerAlarm, confirmAlarm and handleAlarm respectively.

updateTimerTick
Public Shared Sub updateTimerTick()
Wrapper method called by the UpdateTimer in the Utilities module.
This method calls the updateTimerCounters method, which updates the counters for the timers.
If the flag variable updateWithTimer is true, that is, if the module is set to update states based on a timer, then this method also calls the methods checkTriggerVars, checkConfirmVars and checkHandledVars. These methods handle the updating of the state based on the variables in the VariableManager.

updateState
Public Shared Sub updateState()
Wrapper method that calls the methods checkTriggerVars, checkConfirmVars and checkHandledVars, if the flag variable updateOnDemand is set to true, that is, if the module is set to update states when any variable is changed in the VariableManager.

checkTriggerVars
Private Shared Sub checkTriggerVars()
This method loops through all alarms and individually checks if they:

- are not triggered

If this holds, the method checks the conditions for triggering and triggers the alarm using the triggerAlarm method if any one of the conditions hold. This is done by calling the checkOneConditionHolds method in the Utility module with the list of the trigger conditions as the input parameter.

checkConfirmVars
Private Shared Sub checkConfirmVars()
This method loops through all alarms and individually checks if they:

- are triggered
- are not confirmed
- need confirmation and
- can be confirmed with a condition

If this holds, the method checks the conditions for confirming and confirms the alarm using the confirmAlarm method if any one of the conditions hold. This is done by calling the checkOneConditionHolds method in the Utility module with the list of the confirm conditions as the input parameter.

checkHandledVars
Private Shared Sub checkHandledVars()
This method loops through all alarms and individually checks if they:

- are triggered

- are confirmed (or not needing to be confirmed, thus automatically confirming it)
- need handling
- can be handled with a condition and
- don't have a click condition

If this holds, the method checks the conditions for handling and handles the alarm using the handleAlarm method if all of the conditions hold. This is done by calling the checkAllConditionsHolds method in the Utility module with the list of the confirm conditions as the input parameter.

updateTimerCounters
Private Shared Sub updateTimerCounters()
This method updates the counters for the timers. The actual timing is done by the updateTimer in the Utilities module and for each trigger of that timer, this method is called. This method adds the timer interval to a counter for each call and that stores the time that has passed. The trigger timer starts when the software starts. If the trigger timer reaches its target, the alarm triggers instantly, using the triggerAlarm method. There are no timers for confirming or handling an alarm.

triggerAlarm
Public Shared Sub triggerAlarm(ByVal index As Integer)
This method handles triggering an alarm with the index from the input parameter.
It first checks that the alarm hasn't triggered before or that it is allowed to trigger more than once. If not, nothing happens.
If this holds, the method changes the alarm status to triggered and requests a new ID for the alarm, using the GetActionID method in the MainForm module.
A log entry is made that the alarm has triggered.
If a sound is to be played according to the script, it is now played and the ID of the sound is saved, so that the sound can later be stopped when the alarm is confirmed or handled.
The Utilities module method eventSetVars is called with the list of variables to be modified when the alarm is triggered. The method modifies these variables accordingly, if any.
If confirmation is not needed for the alarm, its status is set to consider it confirmed.
If handling is not needed for the alarm, its status is reset to its initial state again as no further user input is needed for the alarm. The sound keeps playing for its duration however.
If a message box is scripted to be shown, it is now shown. If the alarm can be confirmed by the message box, that also happens as the user presses a button on the message box. The alarm is confirmed by calling the confirmAlarm method.

confirmAlarm
Public Shared Sub confirmAlarm(ByVal index As Integer)
This method handles confirming an alarm with the index from the input parameter.
It changes the status of the alarm to confirmed and a log entry is made that the alarm has been confirmed.
The Utilities module method eventSetVars is called with the list of variables to be modified when the alarm is confirmed. The method modifies these variables accordingly, if any.
If there exist an alarm sound, it is turned off.

handleAlarm
Public Shared Sub handleAlarm(ByVal index As Integer)
This method handles handling an alarm with the index from the input parameter.
The Utilities module method eventSetVars is called with the list of variables to be modified when the alarm is handled. The method modifies these variables accordingly, if any.
A log entry is made that the alarm has been handled.
If there exist an alarm sound, it is turned off.
The alarm is then reset to its initial state without an ID and with the flags confirmed and triggered set to False.

findAlarmIndexByName
Public Shared Function findAlarmIndexByName(ByVal name As String) As Integer
This method takes an alarm name as the input parameter and finds the alarm with that name in the list of alarms and then returns the index of that alarm.
If no alarm was found with that name, a -1 is returned instead.

getAlarmStatus
Public Shared Function getAlarmStatus(ByVal index As Integer) As Boolean()
This method takes an alarm index as the input parameter and returns an array of two booleans where the first value is whether the alarm is triggered and the second value is whether the alarm is confirmed.
If the index is -1, both entries are returned as false.
If the index is otherwise less than 0, a log entry is made that an alarm with an index lower than 0 was requested to be read and both entries are returned as false.
if the index is larger than the highest index of an alarm, a log entry is made that an alarm with an index higher than the available alarms was requested to be read and both entries are returned as false.

Members of SoundManager

This section describes the different members contained in the module and what their code does.
The format is on the following syntax:

Name of the member.
The full syntax for the member.
Description of the member's body of code.

SoundManager
Public Class SoundManager
This is the base class of the module and it hosts all other members of the module.
In addition to this, the base class also hosts a variable to keep track of the last sound ID as well as the list of the sounds.

mciSendString

Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal lpstrCommand As String, ByVal lpstrReturnString As String, ByVal uReturnLength As Integer, ByVal hwndCallback As Integer) As Integer

This is a function declared from the file winmm.dll and works as the Media Control Interface for the Windows Multimedia API. It is not original work, but provided as part of the Windows operating system.

The software uses this function to interface with the WinMM in order to play sounds, see Using the Windows Multimedia API (p. 23) for more information.

LoadSounds

Shared Sub LoadSounds()

This method is called by the MainForm when it's loaded and it lists all the names of the sound files from the sound folder into a the list SoundList. For each sound, a log entry is made that the sound has been loaded.

SoundExist

Shared Function SoundExist(ByVal soundName As String) As Boolean

This method takes a sound name as the input parameter and returns true if the sound exists in the SoundList or false if it does not.

OpenSound

Shared Function OpenSound(ByVal soundName As String) As Integer

This method opens a sound file and loads it into memory to prepare it for playback. It starts by checking if the sound name from the input parameter exists, calling the SoundExist method. If it doesn't, an error is generated to notify the user and the software closes. Otherwise, a new unique ID is requested and the mciSendString function is called.

The string sent via the function is "Open [soundNameAndDir] type mpegvideo alias [ID]". This tells the WinMM to open the sound file and treat it as an mpegvideo type file and name the sound [ID].

The mpegvideo type is chosen so that a multitude of different file formats can be used for the sound, among which are mp3 and wav. Having a unique ID as the name of the sound means that it can later be targeted by other command strings to be played back, stopped, unloaded etc.

The method returns the ID of the sound.

PlaySound

Shared Sub PlaySound(ByVal ID As Integer, ByVal doLoop As Boolean)

This method takes a sound ID as an input parameter and plays that sound using the mciSendString function. It also takes an input parameter determining if the sound should loop or not.

The string sent via the function is "play [ID]" or "play [ID] repeat" depending on if the sound should loop or not.

StopSound

Shared Sub StopSound(ByVal ID As Integer)

This method takes a sound ID as the input parameter and stops that sound, but does not set the playback timer to 0. This is done using the mciSendString function.

The string sent via the function is "stop [ID]"

CloseSound
Shared Sub CloseSound(ByVal ID As Integer)
This method takes a sound ID as the input parameter and closes that sound file, unloading it from memory. It cannot be played again until reopened. This is done using the mciSendString function.
The string sent via the function is "close [ID]"

SoundSetPosition
Shared Sub SoundSetPosition(ByVal ID As Integer, ByVal position As Integer, ByVal playing As Boolean, ByVal doLoop As Boolean)
This method takes a sound ID as an input parameter and sets the playback timer position of that sound to the value, in milliseconds, in the input parameter called position and then automatically stops the playback. This is done using the mciSendString function.
The string sent via the function is "seek [ID] to [position]"
If the input parameter called playing is set to true, the PlaySound method is called to start the playback again and if the input parameter doLoop is true, then the sound will also repeat when played again.

SoundStatusMode
Shared Function SoundStatusMode(ByVal ID As Integer) As String
This method takes a sound ID as the input parameter and returns the playback status of the sound as a string. This is done using the mciSendString function.
The string sent via the function is "status [ID] mode".
This call tells the WinMM to return the playback status to a temporary string variable.
The possible return messages could not be deterministically identified from the MSDN webpage on the status command (MSDN, p. Status command). It is stated that all devices will return the following values:

- not ready
- paused
- playing and
- stopped

While some devices can return the following additional values:

- open
- parked
- recording and
- seeking

The main use of this method is however to determine whether a sound is playing or not. The other values are unused by the software at this point.

SoundStatusPosition
Shared Function SoundStatusPosition(ByVal ID As Integer) As Integer
This method takes a sound ID as the input parameter and returns the playback timer position of the sound in milliseconds. This is done using the mciSendString function.
The string sent via the function is "status [ID] position".
This call tells the WinMM to return the playback timer position to a temporary string variable.
The string is type casted to an integer and returned by this method. A 0 is returned if the return string wasn't numeric, that is, if an unexpected error occurred.

SoundStatusLength
Shared Function SoundStatusLength(ByVal ID As Integer) As Integer
This method takes a sound ID as the input parameter and returns the length of the sound in milliseconds. This is done using the mciSendString function.
The string sent via the function is "status [ID] length".
This call tells the WinMM to return the length of the sound to a temporary string variable.
The string is type casted to an integer and returned by this method. A 0 is returned if the return string wasn't numeric, that is, if an unexpected error occurred.

SoundStatusVolume
Shared Function SoundStatusVolume(ByVal ID As Integer) As Integer
This method takes a sound ID as the input parameter and returns the volume of the sound as a value between 0 and 1000. This is done using the mciSendString function.
The string sent via the function is "status [ID] volume".
This call tells the WinMM to return the volume of the sound to a temporary string variable.
The string is type casted to an integer and returned by this method. A 0 is returned if the return string wasn't numeric, that is, if an unexpected error occurred.

SoundSetVolume
Shared Sub SoundSetVolume(ByVal ID As Integer, ByVal volume As Integer)
This method takes a sound ID as the input parameter and sets the volume of the sound as a value between 0 and 1000 determined by the input parameter called volume. This is done using the mciSendString function.
The string sent via the function is "setaudio [ID] volume to [volume]".
If the volume input parameter is lower than 0 or higher than 1000, the volume is set to the limit that's closest to its value and a log entry is made with a warning about the issue.

PlayNewSound
Shared Function PlayNewSound(ByVal soundName As String, ByVal doLoop As Boolean, ByVal volume As Integer) As Integer
Wrapper method that calls the methods OpenSound, PlaySound and SoundSetVolume in order to load and play the sound from the input parameter called soundName with the volume from the input parameter volume. It loops the sound if the input parameter doLoop is true.
The method returns the ID of the sound.

StopAndCloseSound
Shared Sub StopAndCloseSound(ByVal ID As Integer)
Wrapper method that takes a sound ID as the input parameter and calls the methods StopSound and CloseSound in order to stop and unload the sound with the given ID.

SoundSetPercentPosition
Shared Sub SoundSetPercentPosition(ByVal ID As Integer, ByVal percentPosition As Single, ByVal playing As Boolean, ByVal doloop As Boolean)
Wrapper method that sets the playback timer position for the sound with the sound ID from the input parameter ID to a percental position as determined by the input parameter percentPosition. The method does this by calling SoundStatusLength to get the length of the sound and then calculates the position in milliseconds from that and the percental position.
The method then calls the method SoundSetPosition to set the position accordingly.

Members of ErrorHandler

This section describes the different members contained in the module and what their code does.
The errors are shown in the table below:

Error method	Message
ShowAlarmMultipleClickError	Error on row [row] in the file: [file] There are more than a single click condition, only a single click condition is supported in the Handling section.
ShowBoolError	Error on row [row] in the file: [file] The value is not true or false.
ShowInitialSignError	Error on row [row] in the file: [file] Expected ' or [at start of row.
ShowMultipleTimerError	Error on row [row] in the file: [file] There are more than a single timer. Only a single timer is supported.
ShowNoEndSignError	Error on row [row] in the file: [file] Expected] at the end of the tag.
ShowNotNumericVarError	Error on row [row] in the file: [file] Value [varValue] is not numeric.
ShowParamNumError	Error on row [row] in the file: [file] Wrong number of parameters.
ShowSoundExistError	Error on row [row] in the file: [file] Sound [soundName] does not exist.
ShowTagError	Error on row [row] in the file: [file] Invalid tag [tagName].
ShowTaskMultipleClickError	Error on row [row] in the file: [file] There are more than a single click condition, only a single click condition is supported in the Trigger and End sections.
ShowTimerNotNumError	Error on row [row] in the file: [file] Timer value is not numeric.
ShowTimerNotPosError	Error on row [row] in the file: [file] Timer value is not positive.
ShowUnknownObjError	Error on row [row] in the file: [file] Object [objName] does not exist.
ShowUnknownVarError	Error on row [row] in the file: [file] Variable [varName] does not exist.
ShowVarNotExist	Error in variable search: Variable with name [varName] could not be found.

Appendix - The software code

This appendix hosts the code for the software. In addition to the code here, which is the code for all the modules, Visual basic also makes auto-generated code that is required to run the software, for example the code for the graphics. No auto-generated code is present in the report.

MainForm

Filename: MainForm.vb

```
Imports System.Net
Imports System.Text.Encoding
Imports System.BitConverter
Imports System.Collections.Generic



Public Class MainForm
  Implements IMessageFilter

  Dim publisher As New Sockets.UdpClient(0)
  Dim subscriber As New Sockets.UdpClient(49160)

  Public Shared logNameAndPath As String
  Public Shared logFolder As String = "logs"
  Public Shared taskFolder As String = "tasks"
  Public Shared soundFolder As String = "sounds"
  Public Shared alarmFolder As String = "alarms"
  Public Shared logFolderDir As String
  Public Shared taskFolderDir As String
  Public Shared soundFolderDir As String
  Public Shared alarmFolderDir As String
  Public Shared startTime As Date
  Public Shared taskList As New List(Of TaskManager)()
  Public Shared IDCounter As Integer = -1
  Public Shared taskFileEnding As String = ".tsk"
  Public Shared soundFileEndings As New List(Of String) From {".wav", ".mp3"}
  Public Shared alarmFileEnding As String = ".alm"

  'Constructor
  Public Sub New()
    'Global mouseclick event stuff start'
    InitializeComponent()
    Application.AddMessageFilter(Me)
    'Global mouseclick event stuff end'
  End Sub


  'When form is closed
  Protected Overrides Sub OnFormClosed(ByVal e As System.Windows.Forms.FormClosedEventArgs)
    'Global mouseclick event stuff start'
    Application.RemoveMessageFilter(Me)
    'Global mouseclick event stuff end'
  End Sub
```

```
   Public Function PreFilterMessage(ByRef m As System.Windows.Forms.Message) As Boolean Implements
System.Windows.Forms.IMessageFilter.PreFilterMessage
        Dim tempTimeNow As Date = Now
        'Utilities.stopWatchItem.Restart()

        Dim fullName As String
        Dim mp As Point = MousePosition

        'catch WM_LBUTTONDOWN
        If m.Msg = &H201 Then

            'Console.WriteLine(tempTimeNow.ToString("MM/dd/yyyy HH:mm:ss.fffffff") + " " +
Now.ToString("MM/dd/yyyy HH:mm:ss.fffffff") + " " + CStr(Utilities.stopWatchItem.ElapsedTicks /
TimeSpan.TicksPerMillisecond))

            Dim pos As New Point(m.LParam.ToInt32() And &HFFFF, m.LParam.ToInt32() >> 16)
            Dim ctl As Control = Control.FromHandle(m.HWnd)
            Dim controlName As String
            If ctl IsNot Nothing Then
                'If you hit a control, use this


                'Build the full path of the control
                Dim testCtrl As Object
                testCtrl = ctl
                fullName = ctl.Name
                While (testCtrl.Parent IsNot Nothing)
                    fullName = testCtrl.Parent.Name + "/" + fullName
                    testCtrl = testCtrl.Parent
                End While

                controlName = ctl.Name

            Else
                'If you don't hit a control, use this

                'Say that no control was pressed
                fullName = "Nothing"
                controlName = "Nothing"

            End If
            'Use this to do stuff regardless if you hit or don't hit a control

            'Write the mouseclick to the log
            WriteMouseLog(fullName, mp.X, mp.Y)
            ReportClickToTask(controlName, mp.X, mp.Y)
            ReportClickToAlarm(controlName, mp.X, mp.Y)

            'Write to debug boxes
            DebugBox.Text = fullName 'ctl.Name
            DebugCoordX.Text = CInt(mp.X)
            DebugCoordY.Text = CInt(mp.Y)
```

```
    End If
    Return False
End Function




Private Sub MainForm_Load(sender As Object, e As EventArgs) Handles Me.Load

    'Check if logfolder exists otherwise create it
    logFolderDir = CurDir() + "\" + logFolder
    If Not My.Computer.FileSystem.DirectoryExists(logFolderDir) Then
        My.Computer.FileSystem.CreateDirectory(logFolderDir)
    End If

    'Check if taskfolder exists otherwise create it
    taskFolderDir = CurDir() + "\" + taskFolder
    If Not My.Computer.FileSystem.DirectoryExists(taskFolderDir) Then
        My.Computer.FileSystem.CreateDirectory(taskFolderDir)
    End If

    'Check if soundfolder exists otherwise create it
    soundFolderDir = CurDir() + "\" + soundFolder
    If Not My.Computer.FileSystem.DirectoryExists(soundFolderDir) Then
        My.Computer.FileSystem.CreateDirectory(soundFolderDir)
    End If

    'Check if alarmfolder exists otherwise create it
    alarmFolderDir = CurDir() + "\" + alarmFolder
    If Not My.Computer.FileSystem.DirectoryExists(alarmFolderDir) Then
        My.Computer.FileSystem.CreateDirectory(alarmFolderDir)
    End If

    'Initiate the text file with the name formatted as Touchpanel log YYYY-MM-DD HHMMSS in the folder \logs
    startTime = Now
    logNameAndPath = logFolderDir + "\Touchpanel log " + Format(startTime, "yyyy-MM-dd HHmmss") + ".txt"

    SoundManager.LoadSounds()
    VariableManager.initVariables()

    'Initiate timers
    TimUDP.Interval = 100
    TimUDP.Enabled = True


    'Initiate UDP
    subscriber.Client.ReceiveTimeout = 100
    subscriber.Client.Blocking = False
    subscriber.Client.ReceiveBufferSize = 16

    'Initiate the UDP faker
    Dim frmUDP_Faker As New UDP_Faker()
```

```
    frmUDP_Faker.Show()

    'Initiate the Main UI
    Dim frmMainUI As New MainUI()
    frmMainUI.Show()

    'Initiate the Utility form
    Dim frmClassUtilities As New Utilities()

    'Initiate Alarms
    AlarmManager.initializeAlarmSystem()

    'Initiate Tasks
    InitiateTasks()

    WriteSystemLog("System started")
    VariableManager.setVar("SystemStarted", 1)

  End Sub



  Private Sub TimUDP_Tick(sender As Object, e As EventArgs) Handles TimUDP.Tick

    'Send
    UDPTools.sendUDP(TBIP.Text, CInt(TBPort.Text),
UDPTools.createSngPacket(VariableManager.readUDPVar()), publisher)

    'Receive
    VariableManager.setUDPVar(UDPTools.readSngPacket(UDPTools.receiveUDP(subscriber)))
  End Sub

  ''' <summary>
  ''' Write a log entry. "HH:MM:SS.fff;LogString"
  ''' </summary>
  ''' <param name="LogString">String to write in log</param>
  ''' <remarks></remarks>
  Shared Sub WriteToLog(ByVal LogString As String)

    Dim writeString As String = ""
    Dim writeTime As String = DateAndTime.Now.ToString("MM/dd/yyyy HH:mm:ss.fff")


    writeString = writeTime + ";" + LogString + vbCrLf
    My.Computer.FileSystem.WriteAllText(logNameAndPath, writeString, True)

  End Sub

  ''' <summary>
  ''' Write a log entry. "HH:MM:SS.fff;LogString" in a custom logfile
  ''' </summary>
  ''' <param name="LogString">String to write in log</param>
  ''' <remarks></remarks>
  Shared Sub WriteToCustomLog(ByVal logNameAndPathCustom As String, LogString As String)
```

```
        Dim writeString As String = ""
        Dim writeTime As String = DateAndTime.Now.ToString("MM/dd/yyyy HH:mm:ss.fff")


        writeString = writeTime + ";" + LogString + vbCrLf
        My.Computer.FileSystem.WriteAllText(logNameAndPathCustom, writeString, True)

    End Sub

    ''' <summary>
    ''' Write a mouse click log entry. "HH:MM:SS.fff;MC;PosX;PosY;ControlPathAndName"
    ''' </summary>
    ''' <param name="ControlPathAndName">Name of the clicked control</param>
    ''' <param name="PosX">Clicked position x</param>
    ''' <param name="PosY">Clicked position y</param>
    ''' <remarks></remarks>
    Shared Sub WriteMouseLog(ByVal ControlPathAndName As String, ByVal PosX As Integer, ByVal PosY As
Integer)

        'Console.WriteLine("MC: " + Now.ToString("MM/dd/yyyy HH:mm:ss.fffffff") + " " +
CStr(Utilities.stopWatchItem.ElapsedTicks / TimeSpan.TicksPerMillisecond))

        Dim writeString As String = ""
        writeString = "MC" + ";" + PosX.ToString().PadLeft(4, "0") + ":" + PosY.ToString().PadLeft(4, "0") + ";" +
ControlPathAndName

        WriteToLog(writeString)

    End Sub

    ''' <summary>
    ''' Write an event log entry. "HH:MM:SS.fff;EV;strEvent;strDetails"
    ''' </summary>
    ''' <param name="strEvent">Event tag</param>
    ''' <param name="strDetails">Event details</param>
    ''' <remarks></remarks>
    Shared Sub WriteEventLog(ByVal strEvent As String, ByVal strDetails As String)

        'Console.WriteLine("EV: " + Now.ToString("MM/dd/yyyy HH:mm:ss.fffffff") + " " +
CStr(Utilities.stopWatchItem.ElapsedTicks / TimeSpan.TicksPerMillisecond))

        Dim writeString As String = ""
        writeString = "EV" + ";" + strEvent + ";" + strDetails

        WriteToLog(writeString)
    End Sub

    ''' <summary>
    ''' Write a system log entry. "HH:MM:SS.fff;SY;strDetails"
    ''' </summary>
    ''' <param name="strDetails">Details</param>
    ''' <remarks></remarks>
    Shared Sub WriteSystemLog(ByVal strDetails As String)
        Dim writeString As String = ""
```

```
    writeString = "SY" + ";" + strDetails

    WriteToLog(writeString)
End Sub

''' <summary>
''' Tell the TaskManager that a control has been clicked.
''' </summary>
''' <param name="ControlName">Name of the control</param>
''' <param name="PosX">Postion X</param>
''' <param name="PosY">Position Y</param>
''' <remarks></remarks>
Private Sub ReportClickToTask(ByVal ControlName As String, ByVal PosX As Integer, ByVal PosY As Integer)
    TaskManager.reportClick(ControlName)
End Sub

''' <summary>
''' Tell the AlarmManager that a control has been clicked.
''' </summary>
''' <param name="ControlName">Name of the control</param>
''' <param name="PosX">Postion X</param>
''' <param name="PosY">Position Y</param>
''' <remarks></remarks>
Private Sub ReportClickToAlarm(ByVal ControlName As String, ByVal PosX As Integer, ByVal PosY As Integer)
    AlarmManager.reportClick(ControlName)
End Sub

''' <summary>
''' Initiate the tasks from the taskfolder.
''' </summary>
''' <remarks></remarks>
Private Sub InitiateTasks()
    Dim TaskFiles() As String = My.Computer.FileSystem.GetFiles(taskFolderDir).ToArray()

    For Each taskName In TaskFiles
        If taskName.Substring(taskName.Length - 4) = taskFileEnding Then
            taskList.Add(New TaskManager(taskName))
        End If
    Next

End Sub

''' <summary>
''' Get a new unique action identifying number as an integer
''' </summary>
''' <returns></returns>
''' <remarks></remarks>
Shared Function GetActionID() As Integer
    IDCounter += 1
    Return IDCounter
End Function

''' <summary>
''' Check recursively if a control exists in any layer of the UI
''' </summary>
```

```
''' <param name="objectName">Name of the control</param>
''' <returns></returns>
''' <remarks></remarks>
Public Function objectExistInUI(ByVal objectName As String) As Boolean

  For Each currentObject In MainUI.Controls
    If currentObject.Name = objectName Then
      Return True
    ElseIf objectRecursiveExist(objectName, currentObject) Then
      Return True
    End If
  Next

  Return False

End Function

''' <summary>
''' Check recursively if a control exists in the specified container or any of it's contained containers.
''' </summary>
''' <param name="objectName">Name of the control</param>
''' <param name="controlContainer">The container to search</param>
''' <returns></returns>
''' <remarks></remarks>
Private Function objectRecursiveExist(ByVal objectName As String, ByRef controlContainer As Object) As
Boolean
  For Each currentObject In controlContainer.Controls
    If currentObject.Name = objectName Then
      Return True
    ElseIf objectRecursiveExist(objectName, currentObject) Then
      Return True
    End If
  Next

  Return False
End Function


Private Sub cbTraffic_CheckedChanged(sender As Object, e As EventArgs) Handles cbTraffic.CheckedChanged
  If cbTraffic.Checked = True Then
    VariableManager.setVar("StartTraffic", 1)
    cbTraffic.ForeColor = Color.Green
  Else
    VariableManager.setVar("StartTraffic", 0)
    cbTraffic.ForeColor = Color.Red

  End If

End Sub
End Class
```

UDPTools

Filename: UDPTools.vb

```
Imports System.Net
Imports System.Text.Encoding
Imports System.BitConverter



''' <summary>
''' UDP-related tools for the program
''' </summary>
''' <remarks></remarks>
Public Class UDPTools

    ''' <summary>
    ''' Connect and send a packet over UDP to the IP on the Port.
    ''' </summary>
    ''' <param name="IP">Receiving IP adress</param>
    ''' <param name="Port">Receiving Port number</param>
    ''' <param name="Packet">Packet to send</param>
    ''' <param name="UDPpublisher">UDP sender to use</param>
    ''' <remarks></remarks>
    Shared Sub sendUDP(ByVal IP As String, ByVal Port As Integer, ByVal Packet() As Byte, ByRef UDPpublisher As
Sockets.UdpClient)

        UDPpublisher.Connect(IP, Port)
        UDPpublisher.Send(Packet, Packet.Length)

    End Sub

    ''' <summary>
    ''' Receive a packet over UDP
    ''' </summary>
    ''' <param name="UDPsubscriber">UDP receiver to use</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function receiveUDP(ByRef UDPsubscriber As Sockets.UdpClient) As Byte()

        Try
            Dim endPoint As IPEndPoint = New IPEndPoint(IPAddress.Any, 0)
            Return UDPsubscriber.Receive(endPoint)
        Catch
            Dim tmp(0) As Byte
            tmp(0) = Nothing
            Return tmp
        End Try

    End Function

    ''' <summary>
    ''' Create a UDP packet of single variables and return them as a byte array
    ''' </summary>
    ''' <param name="Vars">Array of single variables to include in the packet</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function createSngPacket(ByVal Vars() As Object) As Byte()
```

```
    Dim bytes(0 To (Vars.Length * 4) - 1) As Byte

    For x = 0 To Vars.Length - 1
      bytes = arrMod(bytes, System.BitConverter.GetBytes(CSng(Vars(x))), x * 4)
    Next
    Return bytes

  End Function


  ''' <summary>
  ''' Read single variables from a UDP packet and return them as an array of singles.
  ''' </summary>
  ''' <param name="bytes">The packet in the form of a byte array</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Shared Function readSngPacket(ByVal bytes() As Byte) As Single()

    If bytes(0) = Nothing Then
      Dim tmpVars(0) As Single
      tmpVars(0) = Nothing
      Return tmpVars
    End If

    Dim numItems As Integer
    numItems = bytes.Length / 4
    Dim vars(0 To numItems - 1) As Single

    For x = 0 To numItems - 1
      vars(x) = ToSingle(bytes, x * 4)
    Next

    Return vars

  End Function


  ''' <summary>
  ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
  ''' The resulting array may be larger than the source array.
  ''' </summary>
  ''' <param name="source">Array to use as source.</param>
  ''' <param name="target">Array to replace parts of source with.</param>
  ''' <param name="startIndex">Index to replace at, going forward.</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Shared Function arrMod(ByVal source() As String, ByVal target() As String, ByVal startIndex As Integer) As
String()
    If source.Length < startIndex + target.Length Then
      ReDim Preserve source(startIndex + target.Length)
    End If

    For x = 0 To target.Length - 1
      source(x + startIndex) = target(x)
```

```
    Next

    Return source
  End Function

  ''' <summary>
  ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
  ''' The resulting array may be larger than the source array.
  ''' </summary>
  ''' <param name="source">Array to use as source.</param>
  ''' <param name="target">Array to replace parts of source with.</param>
  ''' <param name="startIndex">Index to replace at, going forward.</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Shared Function arrMod(ByVal source() As Integer, ByVal target() As Integer, ByVal startIndex As Integer) As
Integer()
    If source.Length < startIndex + target.Length Then
      ReDim Preserve source(startIndex + target.Length)
    End If

    For x = 0 To target.Length - 1
      source(x + startIndex) = target(x)
    Next

    Return source
  End Function

  ''' <summary>
  ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
  ''' The resulting array may be larger than the source array.
  ''' </summary>
  ''' <param name="source">Array to use as source.</param>
  ''' <param name="target">Array to replace parts of source with.</param>
  ''' <param name="startIndex">Index to replace at, going forward.</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Shared Function arrMod(ByVal source() As Double, ByVal target() As Double, ByVal startIndex As Integer) As
Double()
    If source.Length < startIndex + target.Length Then
      ReDim Preserve source(startIndex + target.Length)
    End If

    For x = 0 To target.Length - 1
      source(x + startIndex) = target(x)
    Next

    Return source
  End Function

  ''' <summary>
  ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
  ''' The resulting array may be larger than the source array.
```

```
    ''' </summary>
    ''' <param name="source">Array to use as source.</param>
    ''' <param name="target">Array to replace parts of source with.</param>
    ''' <param name="startIndex">Index to replace at, going forward.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function arrMod(ByVal source() As Long, ByVal target() As Long, ByVal startIndex As Integer) As
Long()
        If source.Length < startIndex + target.Length Then
            ReDim Preserve source(startIndex + target.Length)
        End If

        For x = 0 To target.Length - 1
            source(x + startIndex) = target(x)
        Next

        Return source
    End Function

    ''' <summary>
    ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
    ''' The resulting array may be larger than the source array.
    ''' </summary>
    ''' <param name="source">Array to use as source.</param>
    ''' <param name="target">Array to replace parts of source with.</param>
    ''' <param name="startIndex">Index to replace at, going forward.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function arrMod(ByVal source() As Single, ByVal target() As Single, ByVal startIndex As Integer) As
Single()
        If source.Length < startIndex + target.Length Then
            ReDim Preserve source(startIndex + target.Length)
        End If

        For x = 0 To target.Length - 1
            source(x + startIndex) = target(x)
        Next

        Return source
    End Function

    ''' <summary>
    ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
    ''' The resulting array may be larger than the source array.
    ''' </summary>
    ''' <param name="source">Array to use as source.</param>
    ''' <param name="target">Array to replace parts of source with.</param>
    ''' <param name="startIndex">Index to replace at, going forward.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function arrMod(ByVal source() As Byte, ByVal target() As Byte, ByVal startIndex As Integer) As Byte()
        If source.Length < startIndex + target.Length Then
            ReDim Preserve source(startIndex + target.Length)
```

```
    End If

    For x = 0 To target.Length - 1
      source(x + startIndex) = target(x)
    Next

    Return source
  End Function

  ''' <summary>
  ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
  ''' The resulting array may be larger than the source array.
  ''' </summary>
  ''' <param name="source">Array to use as source.</param>
  ''' <param name="target">Array to replace parts of source with.</param>
  ''' <param name="startIndex">Index to replace at, going forward.</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Shared Function arrMod(ByVal source() As Char, ByVal target() As Char, ByVal startIndex As Integer) As
Char()
    If source.Length < startIndex + target.Length Then
      ReDim Preserve source(startIndex + target.Length)
    End If

    For x = 0 To target.Length - 1
      source(x + startIndex) = target(x)
    Next

    Return source
  End Function

  ''' <summary>
  ''' Takes the source array and replaces it with the target array from position startIndex and forward and
returns it.
  ''' The resulting array may be larger than the source array.
  ''' </summary>
  ''' <param name="source">Array to use as source.</param>
  ''' <param name="target">Array to replace parts of source with.</param>
  ''' <param name="startIndex">Index to replace at, going forward.</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Shared Function arrMod(ByVal source() As Boolean, ByVal target() As Boolean, ByVal startIndex As Integer)
As Boolean()
    If source.Length < startIndex + target.Length Then
      ReDim Preserve source(startIndex + target.Length)
    End If

    For x = 0 To target.Length - 1
      source(x + startIndex) = target(x)
    Next

    Return source
  End Function
```

```
    ''' <summary>
    ''' Takes the source array and replaces it with the target array from position startIndex and forward and returns it.
    ''' The resulting array may be larger than the source array.
    ''' </summary>
    ''' <param name="source">Array to use as source.</param>
    ''' <param name="target">Array to replace parts of source with.</param>
    ''' <param name="startIndex">Index to replace at, going forward.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function arrMod(ByVal source() As Object, ByVal target() As Object, ByVal startIndex As Integer) As Object()
        If source.Length < startIndex + target.Length Then
            ReDim Preserve source(startIndex + target.Length)
        End If

        For x = 0 To target.Length - 1
            source(x + startIndex) = target(x)
        Next

        Return source
    End Function
End Class
```

UDP Faker

Filename: UDP Faker.vb

```
''' <summary>
''' Fakes a UDP connection by setting those variables from this window
''' </summary>
''' <remarks></remarks>
Public Class UDP_Faker

    Dim sngPosIne As Single = 0
    Dim sngVelocity As Single = 0
    Dim sngRPM As Single = 0
    Dim sngSpeLim As Single = 0
    Dim sngSteerAngle As Single = 0
    Dim sngleftLaneDist As Single = 0
    Dim sngrightLaneDist As Single = 0
    Dim snglaneID As Single = 0
    Dim sngvehicleRadar As Single = 0
    Dim sngsimTimer As Single = 0


    Private Sub UDP_Faker_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        UpdateState.Enabled = True
        UpdateState.Interval = 100
    End Sub


    Private Sub UpdateState_Tick(sender As Object, e As EventArgs) Handles UpdateState.Tick

        TBBoolAlaCom.Text = CStr(VariableManager.readVar("BoolAlaCom"))
        TBFriCoe.Text = CStr(VariableManager.readVar("FriCoe"))
```

```
    TBSteerFeedGain.Text = CStr(VariableManager.readVar("SteerFeedGain"))
    TBStartEvent.Text = CStr(VariableManager.readVar("StartEvent"))
    TBStartTraffic.Text = CStr(VariableManager.readVar("StartTraffic"))
    TBStopSim.Text = CStr(VariableManager.readVar("StopSim"))
    TBscriptState.Text = CStr(VariableManager.readVar("ScriptState"))

    If Not CBFakeOn.Checked Then
      Return
    End If

    Dim VarsArr(0 To 9) As Single
    VarsArr(0) = sngPosIne
    VarsArr(1) = sngVelocity
    VarsArr(2) = sngRPM
    VarsArr(3) = sngSpeLim
    VarsArr(4) = sngSteerAngle
    VarsArr(5) = sngleftLaneDist
    VarsArr(6) = sngrightLaneDist
    VarsArr(7) = snglaneID
    VarsArr(8) = sngvehicleRadar
    VarsArr(9) = sngsimTimer

    VariableManager.setUDPVar(VarsArr)

  End Sub

  Private Sub TBPosIne_TextChanged(sender As Object, e As EventArgs) Handles TBPosIne.TextChanged
    If IsNumeric(TBPosIne.Text) Then
      sngPosIne = CSng(TBPosIne.Text)
    End If
  End Sub

  Private Sub TBVel_TextChanged(sender As Object, e As EventArgs) Handles TBVel.TextChanged
    If IsNumeric(TBVel.Text) Then
      sngVelocity = CSng(TBVel.Text)
    End If
  End Sub

  Private Sub TBRPM_TextChanged(sender As Object, e As EventArgs) Handles TBRPM.TextChanged
    If IsNumeric(TBRPM.Text) Then
      sngRPM = CSng(TBRPM.Text)
    End If
  End Sub

  Private Sub CBSpeLim_CheckedChanged(sender As Object, e As EventArgs) Handles
CBSpeLim.CheckedChanged
    sngSpeLim = CSng(CBSpeLim.Checked)

  End Sub


  Private Sub TBSteerAngle_TextChanged(sender As Object, e As EventArgs) Handles TBSteerAngle.TextChanged
    If IsNumeric(TBSteerAngle.Text) Then
      sngSteerAngle = CSng(TBSteerAngle.Text)
    End If
```

```
  End Sub

  Private Sub TBleftLaneDist_TextChanged(sender As Object, e As EventArgs) Handles
TBleftLaneDist.TextChanged
    If IsNumeric(TBleftLaneDist.Text) Then
      sngleftLaneDist = CSng(TBleftLaneDist.Text)
    End If
  End Sub

  Private Sub TBrightLaneDist_TextChanged(sender As Object, e As EventArgs) Handles
TBrightLaneDist.TextChanged
    If IsNumeric(TBrightLaneDist.Text) Then
      sngrightLaneDist = CSng(TBrightLaneDist.Text)
    End If
  End Sub

  Private Sub TBlaneID_TextChanged(sender As Object, e As EventArgs) Handles TBlaneID.TextChanged
    If IsNumeric(TBlaneID.Text) Then
      snglaneID = CSng(TBlaneID.Text)
    End If
  End Sub

  Private Sub TBvehicleRadar_TextChanged(sender As Object, e As EventArgs) Handles
TBvehicleRadar.TextChanged
    If IsNumeric(TBvehicleRadar.Text) Then
      sngvehicleRadar = CSng(TBvehicleRadar.Text)
    End If
  End Sub

  Private Sub TBsimTimer_TextChanged(sender As Object, e As EventArgs) Handles TBsimTimer.TextChanged
    If IsNumeric(TBsimTimer.Text) Then
      sngsimTimer = CSng(TBsimTimer.Text)
    End If
  End Sub
End Class
```

Utilities

Filename: Utilities.vb

```
''' <summary>
''' Class with utility functions, timers and classes
''' </summary>
''' <remarks></remarks>
Public Class Utilities

  Public Shared stopWatchItem As New StopWatch

  Public Shared UpdateTimerInterval = 20

  ''' <summary>
  ''' Class that creates the structure for a condition used in alarms and tasks.
  ''' </summary>
  ''' <remarks></remarks>
  Public Class condition
    Public type As String
```

```
    Public info1 As String
    Public info2 As String
  End Class

  Public Sub New()

    ' This call is required by the designer.
    InitializeComponent()

    ' Add any initialization after the InitializeComponent() call.
    UpdateTimer.Interval = UpdateTimerInterval
    UpdateTimer.Enabled = True

  End Sub


  Private Sub UpdateTimer_Tick(sender As Object, e As EventArgs) Handles UpdateTimer.Tick
    TaskManager.updateTimerTick()
    AlarmManager.updateTimerTick()
  End Sub

  ''' <summary>
  ''' Reads a script tag and extracts a text from the contents with error handling
  ''' </summary>
  ''' <param name="tagList">The tag to extract from</param>
  ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
  ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Public Function readTextFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal
fileNameAndDir As String) As String
    If tagList.Length <> 2 Then
      ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
      MainForm.Close()
    End If
    Return tagList(1)
  End Function

  ''' <summary>
  ''' Reads a script tag and extracts a boolean from the contents with error handling
  ''' </summary>
  ''' <param name="tagList">The tag to extract from</param>
  ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
  ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Public Function readBoolFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal
fileNameAndDir As String) As Boolean
    If tagList.Length <> 2 Then
      ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
      MainForm.Close()
    End If
    If CStr(tagList(1)).ToLower() = "true" Then
      Return True
    ElseIf CStr(tagList(1)).ToLower() = "false" Then
```

```
            Return False
        Else
            ErrorHandler.ShowBoolError(rowCounter, fileNameAndDir)
            MainForm.Close()
        End If
        Return Nothing
    End Function

    ''' <summary>
    ''' Reads a script tag and extracts a variable name and value pair from the contents with error handling
    ''' </summary>
    ''' <param name="tagList">The tag to extract from</param>
    ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
    ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function readVarnameAndNumberFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer,
ByVal fileNameAndDir As String) As condition
        If tagList.Length <> 3 Then
            ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
            MainForm.Close()
        End If
        If Not VariableManager.varExists(tagList(1)) Then
            ErrorHandler.ShowUnknownVarError(rowCounter, fileNameAndDir, tagList(1))
            MainForm.Close()
        End If
        If Not IsNumeric(tagList(2)) Then
            ErrorHandler.ShowNotNumericVarError(rowCounter, fileNameAndDir, tagList(2))
            MainForm.Close()
        End If

        Dim curItem As New condition
        curItem.type = CStr(tagList(0))
        curItem.info1 = CStr(tagList(1))
        curItem.info2 = CSng(tagList(2))

        Return curItem
    End Function

    ''' <summary>
    ''' Reads a script tag and extracts a variable name from the contents with error handling
    ''' </summary>
    ''' <param name="tagList">The tag to extract from</param>
    ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
    ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function readVarnameFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal
fileNameAndDir As String) As condition
        If tagList.Length <> 2 Then
            ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
            MainForm.Close()
        End If
        If Not VariableManager.varExists(tagList(1)) Then
            ErrorHandler.ShowUnknownVarError(rowCounter, fileNameAndDir, tagList(1))
```

```
            MainForm.Close()
        End If

        Dim curItem As New condition
        curItem.type = CStr(tagList(0))
        curItem.info1 = CStr(tagList(1))
        curItem.info2 = Nothing

        Return curItem
    End Function

    ''' <summary>
    ''' Reads a script tag and extracts a boolean from the contents with error handling
    ''' </summary>
    ''' <param name="tagList">The tag to extract from</param>
    ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
    ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function readClickConditionFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal
fileNameAndDir As String) As condition
        If tagList.Length <> 2 Then
            ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
            MainForm.Close()
        End If
        If Not MainForm.objectExistInUI(tagList(1)) Then
            ErrorHandler.ShowUnknownObjError(rowCounter, fileNameAndDir, tagList(1))
            MainForm.Close()
        End If

        Dim curItem As New Utilities.condition
        curItem.type = CStr(tagList(0))
        curItem.info1 = CStr(tagList(1))
        curItem.info2 = Nothing

        Return curItem
    End Function

    ''' <summary>
    ''' Reads a script tag and extracts a timer condition from the contents with error handling
    ''' </summary>
    ''' <param name="tagList">The tag to extract from</param>
    ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
    ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function readTimerConditionFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer,
ByVal fileNameAndDir As String) As condition
        If tagList.Length <> 2 Then
            ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
            MainForm.Close()
        End If
        If Not IsNumeric(tagList(1)) Then
            ErrorHandler.ShowTimerNotNumError(rowCounter, fileNameAndDir)
            MainForm.Close()
```

```
    End If
    If CSng(tagList(1)) < 0 Then
      ErrorHandler.ShowTimerNotPosError(rowCounter, fileNameAndDir)
      MainForm.Close()
    End If
    Dim curItem As New Utilities.condition
    curItem.type = CStr(tagList(0))
    curItem.info1 = CStr(tagList(1))
    curItem.info2 = Nothing

    Return curItem
  End Function

  ''' <summary>
  ''' Reads a script tag and extracts a control name from the contents with error handling
  ''' </summary>
  ''' <param name="tagList">The tag to extract from</param>
  ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
  ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Public Function readControlFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal
fileNameAndDir As String) As String
    If tagList.Length <> 2 Then
      ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
      MainForm.Close()
    End If
    If Not MainForm.objectExistInUI(tagList(1)) Then
      ErrorHandler.ShowUnknownObjError(rowCounter, fileNameAndDir, tagList(1))
      MainForm.Close()
    End If
    Return tagList(1)
  End Function

  ''' <summary>
  ''' Reads a script tag and extracts a sound name from the contents with error handling
  ''' </summary>
  ''' <param name="tagList">The tag to extract from</param>
  ''' <param name="rowCounter">The row in the script that the tag is originating from</param>
  ''' <param name="fileNameAndDir">The filename and it's directory that the tag is originating from</param>
  ''' <returns></returns>
  ''' <remarks></remarks>
  Public Function readSoundFromTaglist(ByVal tagList() As String, ByVal rowCounter As Integer, ByVal
fileNameAndDir As String) As String
    If tagList.Length <> 2 Then
      ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
      MainForm.Close()
    End If
    If Not SoundManager.SoundExist(tagList(1)) Then
      ErrorHandler.ShowSoundExistError(rowCounter, fileNameAndDir, tagList(1))
      MainForm.Close()
    End If

    Return tagList(1)
  End Function
```

```
    ''' <summary>
    ''' Check if the simulator variables match the conditions in the conditionList and return true if at least one does
or false if at none matches.
    ''' </summary>
    ''' <param name="conditionList"></param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function checkOneConditionHolds(ByVal conditionList As List(Of condition)) As Boolean
        For Each condition As condition In conditionList
            Select Case condition.type
                Case "varLess"
                    If VariableManager.readVar(CStr(condition.info1)) < CSng(condition.info2) Then
                        Return True
                    End If

                Case "varMore"
                    If VariableManager.readVar(CStr(condition.info1)) > CSng(condition.info2) Then
                        Return True
                    End If

                Case "varEqual"
                    If VariableManager.readVar(CStr(condition.info1)) = CSng(condition.info2) Then
                        Return True
                    End If

                Case "varMoreEqual"
                    If VariableManager.readVar(CStr(condition.info1)) >= CSng(condition.info2) Then
                        Return True
                    End If

                Case "varLessEqual"
                    If VariableManager.readVar(CStr(condition.info1)) <= CSng(condition.info2) Then
                        Return True
                    End If

                Case "varNotEqual"
                    If VariableManager.readVar(CStr(condition.info1)) <> CSng(condition.info2) Then
                        Return True
                    End If
                Case Else

            End Select
        Next

        Return False

    End Function

    ''' <summary>
    ''' Check if the simulator variables match the conditions in the conditionList and return true if they all do or
false if at least one doesn't.
    ''' </summary>
    ''' <param name="conditionList">A list of conditions to check</param>
    ''' <returns></returns>
```

```
''' <remarks></remarks>
Public Function checkAllConditionsHolds(ByVal conditionList As List(Of condition)) As Boolean
    For Each condition As condition In conditionList
        Select Case condition.type
            Case "varLess"
                If Not VariableManager.readVar(CStr(condition.info1)) < CSng(condition.info2) Then
                    Return False
                End If

            Case "varMore"
                If Not VariableManager.readVar(CStr(condition.info1)) > CSng(condition.info2) Then
                    Return False
                End If

            Case "varEqual"
                If Not VariableManager.readVar(CStr(condition.info1)) = CSng(condition.info2) Then
                    Return False
                End If

            Case "varMoreEqual"
                If Not VariableManager.readVar(CStr(condition.info1)) >= CSng(condition.info2) Then
                    Return False
                End If

            Case "varLessEqual"
                If Not VariableManager.readVar(CStr(condition.info1)) <= CSng(condition.info2) Then
                    Return False
                End If

            Case "varNotEqual"
                If Not VariableManager.readVar(CStr(condition.info1)) <> CSng(condition.info2) Then
                    Return False
                End If
            Case Else

        End Select
    Next

    Return True

End Function

''' <summary>
''' A wrapper to let the events set variables with the setVar conditions in a list of conditions.
''' </summary>
''' <param name="varList">The list of conditions</param>
''' <remarks></remarks>
Public Shared Sub eventSetVars(ByVal varList As List(Of Utilities.condition))
    For Each varSet As Utilities.condition In varList

        Select Case varSet.type
            Case "setVar"
                VariableManager.setVar(varSet.info1, CSng(varSet.info2))

            Case "addVar"
```

```
                Dim curVal As Single = CSng(VariableManager.readVar(varSet.info1))
                curVal = curVal + CSng(varSet.info2)
                VariableManager.setVar(varSet.info1, curVal)

            Case "subVar"
                Dim curVal As Single = CSng(VariableManager.readVar(varSet.info1))
                curVal = curVal - CSng(varSet.info2)
                VariableManager.setVar(varSet.info1, curVal)

            Case "incVar"
                Dim curVal As Single = CSng(VariableManager.readVar(varSet.info1))
                curVal = curVal + 1
                VariableManager.setVar(varSet.info1, curVal)

            Case "decVar"
                Dim curVal As Single = CSng(VariableManager.readVar(varSet.info1))
                curVal = curVal - 1
                VariableManager.setVar(varSet.info1, curVal)
        End Select
    Next

End Sub

Private Sub Label1_Click(sender As Object, e As EventArgs) Handles Label1.Click

End Sub
End Class
```

VariableManager

Filename: VariableManager.vb

```
''' <summary>
''' Handles the global variables and variables used in scripts. Both custom ones used locally and those that are transferred to and from the vehicle simulator.
''' Allows searching through variables by name.
''' </summary>
''' <remarks></remarks>
Public Class VariableManager

    ''' <summary>
    ''' Pair of name and value that makes up a simulator variable.
    ''' </summary>
    ''' <remarks></remarks>
    Class simVariable
        Public Sub New(ByVal strname As String, ByVal sngvalue As Single)
            name = strname
            value = sngvalue
        End Sub

        Public name As String
        Public value As Single
    End Class

    Private Shared variableList As New List(Of simVariable)
```

```
''' <summary>
''' Initiate the variables.
''' </summary>
''' <remarks></remarks>
Shared Sub initVariables()
    'System variables
    createVariable("SystemStarted", 0)

    'UDP Sent Variables
    createVariable("BoolAlaCom", 0)
    createVariable("FriCoe", 0)
    createVariable("SteerFeedGain", 0)
    createVariable("StartEvent", 0)
    createVariable("StartTraffic", 0)
    createVariable("StopSim", 0)

    'UDP Received Variables
    createVariable("PosIne", 0)
    createVariable("Velocity", 0)
    createVariable("EngRPM", 0)
    createVariable("SpeLim", 0)
    createVariable("SteerAngle", 0)
    createVariable("leftLaneDist", 0)
    createVariable("rightLaneDist", 0)
    createVariable("laneID", 0)
    createVariable("vehicleRadar", 0)
    createVariable("simTimer", 0)


    'Custom variables
    createVariable("MathTest", 0)
    createVariable("ScriptState", 0)
    createVariable("StereoOn", 0)

End Sub

''' <summary>
''' Create a variable and assign a value to it.
''' </summary>
''' <param name="name">Name of variable.</param>
''' <param name="value">Value to assign the variable on creation.</param>
''' <remarks></remarks>
Shared Sub createVariable(ByVal name As String, ByVal value As Single)
    Dim tempVar As New simVariable(name, value)

    variableList.Add(tempVar)

End Sub

''' <summary>
''' Takes a variable array from the UDP receiver or UDP faker and sets the variables accordingly.
''' </summary>
''' <param name="VarArray">Array of variables from the UDP receiver.</param>
''' <remarks></remarks>
Shared Sub setUDPVar(ByVal VarArray() As Single)
```

```
    If VarArray.Length <> 10 Then
        Return
    End If
    Dim varlist As New List(Of simVariable)

    varlist.Add(New simVariable("PosIne", CSng(VarArray(0))))
    varlist.Add(New simVariable("Velocity", CSng(VarArray(1))))
    varlist.Add(New simVariable("EngRPM", CSng(VarArray(2))))
    varlist.Add(New simVariable("SpeLim", CSng(VarArray(3))))
    varlist.Add(New simVariable("SteerAngle", CSng(VarArray(4))))
    varlist.Add(New simVariable("leftLaneDist", CSng(VarArray(5))))
    varlist.Add(New simVariable("rightLaneDist", CSng(VarArray(6))))
    varlist.Add(New simVariable("laneID", CSng(VarArray(7))))
    varlist.Add(New simVariable("vehicleRadar", CSng(VarArray(8))))
    varlist.Add(New simVariable("simTimer", CSng(VarArray(9))))

    setVar(varlist)


    'setVar("PosIne", CSng(VarArray(0)))
    'setVar("Velocity", CSng(VarArray(1)))
    'setVar("EngRPM", CSng(VarArray(2)))
    'setVar("SpeLim", CSng(VarArray(3)))
    'setVar("SteerAngle", CSng(VarArray(4)))
    'setVar("leftLaneDist", CSng(VarArray(5)))
    'setVar("rightLaneDist", CSng(VarArray(6)))
    'setVar("laneID", CSng(VarArray(7)))
    'setVar("vehicleRadar", CSng(VarArray(8)))
    'setVar("simTimer", CSng(VarArray(9)))

End Sub

''' <summary>
''' Read the variables and put them in a variable array for the UDP sender to be sent to the simulator.
''' Returns the array.
''' </summary>
''' <returns></returns>
''' <remarks></remarks>
Shared Function readUDPVar() As Object()
    Dim tmpReadVar(0 To 5)
    tmpReadVar(0) = 5 'ReadVar("FriCoe")
    tmpReadVar(1) = 1 'ReadVar("BoolAlaCom")
    tmpReadVar(2) = readVar("SteerFeedGain")
    tmpReadVar(3) = readVar("StartEvent")
    tmpReadVar(4) = readVar("StartTraffic")
    tmpReadVar(5) = readVar("StopSim")

    Return tmpReadVar
End Function

''' <summary>
''' Set the value of a variable.
''' </summary>
''' <param name="name">Name of the variable.</param>
```

```
''' <param name="value">Value to be set.</param>
''' <remarks></remarks>
Shared Sub setVar(ByVal name As String, ByVal value As Single)
   For indexCounter As Integer = 0 To variableList.Count - 1
      If variableList(indexCounter).name = name Then
         Dim tempVar As New simVariable(name, value)
         variableList(indexCounter) = tempVar

         TaskManager.updateState()
         AlarmManager.updateState()
         Return
      End If
   Next
   ErrorHandler.ShowVarNotExist(name)
   MainForm.Close()
End Sub

Shared Sub setVar(ByVal varlist As List(Of simVariable))
   For Each var As simVariable In varlist
      Dim found As Boolean = False
      For indexCounter As Integer = 0 To variableList.Count - 1
         If variableList(indexCounter).name = var.name Then
            Dim tempVar As New simVariable(var.name, var.value)
            variableList(indexCounter) = tempVar

            found = True
            Exit For
         End If
      Next
      If found = False Then
         ErrorHandler.ShowVarNotExist(var.name)
         MainForm.Close()
      End If
   Next
   TaskManager.updateState()
   AlarmManager.updateState()
End Sub

''' <summary>
''' Read value from a variable. Returns the value.
''' </summary>
''' <param name="name">Name of the variable.</param>
''' <returns></returns>
''' <remarks></remarks>
Shared Function readVar(ByVal name As String) As Object
   For indexCounter As Integer = 0 To variableList.Count - 1
      If variableList(indexCounter).name = name Then
         Return variableList(indexCounter).value
      End If
   Next
   ErrorHandler.ShowVarNotExist(name)
   MainForm.Close()
   Return False
End Function
```

```
    ''' <summary>
    ''' Check if a variable with a certain name exists. Returns true or false.
    ''' </summary>
    ''' <param name="name">Name of the variable.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function varExists(ByVal name As String) As Boolean
        For indexCounter As Integer = 0 To variableList.Count - 1
            If variableList(indexCounter).name = name Then
                Return True
            End If
        Next
        Return False
    End Function

End Class
```

TaskManager

Filename: TaskManager.vb

```
''' <summary>
''' Manages the scripted tasks in the program. One instance per task file.
''' </summary>
''' <remarks></remarks>
Public Class TaskManager

    Public Shared updateWithTimer As Boolean = True
    Public Shared updateOnDemand As Boolean = True



    Private Const CATNONE = 0
    Private Const CATINFO = 1
    Private Const CATTRIGGER = 2
    Private Const CATFAIL = 3
    Private Const CATEND = 4
    Private Const CATACTION = 5

    Dim taskName As String
    Dim taskID As Integer = -1
    Dim taskRunning As Boolean

    Private Triggers As New List(Of Utilities.condition)   ' (type, info1, info2)
    Private Fails As New List(Of Utilities.condition)       ' example ("varLow", "Velocity", 100)
    Private Ends As New List(Of Utilities.condition)        ' example ("click", "ControlName", "")
    Private actionList As New List(Of Utilities.condition)

    Private TriggerSetVar As New List(Of Utilities.condition)
    Private FailsSetVar As New List(Of Utilities.condition)
    Private EndsSetVar As New List(Of Utilities.condition)

    Private TriggerHasClick As Boolean  ' If the list has a trigger condition
    Private FailHasClick As Boolean     ' Probably has no use since it's click or any variable at a certain value.
    Private EndHasClick As Boolean
```

```
Private TriggerTimerDone As Boolean
Private FailTimerDone As Boolean    ' Probably has no use since when it's done, it's failed.
Private EndTimerDone As Boolean

Private HasActionList As Boolean
Private actionListDone As Boolean
Private actionListCounter As Integer

Private TriggerOnce As Boolean = True
Private HasTriggered As Boolean

Private triggerTimerCounter As Single
Private failTimerCounter As Single
Private endTimerCounter As Single

Private TriggerTimerTarget As Single = -1
Private FailTimerTarget As Single = -1
Private EndTimerTarget As Single = -1


Private timerInterval As Single

''' <summary>
''' Constructor -- Create a new task from file.
''' </summary>
''' <param name="fileNameAndDir">Taskfile name and directory.</param>
''' <remarks></remarks>
Public Sub New(ByVal fileNameAndDir As String)

  Dim textRow() As String = My.Computer.FileSystem.ReadAllText(fileNameAndDir).Split(vbCrLf)
  Dim rowCounter As Integer
  Dim category As Integer = CATNONE
  timerInterval = Utilities.UpdateTimerInterval / 1000

  For rowCounter = 0 To textRow.GetLength(0) - 1
    textRow(rowCounter) = textRow(rowCounter).Trim()

    If textRow(rowCounter).Length = 0 Then
      Continue For
    ElseIf textRow(rowCounter).Substring(0, 1) = "'" Then
      Continue For
    ElseIf textRow(rowCounter).Substring(0, 1) <> "[" Then
      ErrorHandler.ShowInitialSignError(rowCounter, fileNameAndDir)
      MainForm.Close()
    End If

    Dim tagEnd As Integer = textRow(rowCounter).IndexOf("]")
    If tagEnd = -1 Then
      ErrorHandler.ShowNoEndSignError(rowCounter, fileNameAndDir)
      MainForm.Close()
    End If

    Dim tagContents As String = textRow(rowCounter).Substring(1, tagEnd - 1)
    Dim tagList() As String = tagContents.Split(":")
```

```
If tagList.Length < 1 Then
  ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
  MainForm.Close()
End If
If tagList.Length > 3 Then
  ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
  MainForm.Close()
End If

If category = CATNONE Then
  Select Case tagContents
    Case "info"
      category = CATINFO
      Continue For
    Case "trigger"
      category = CATTRIGGER
      Continue For
    Case "fail"
      category = CATFAIL
      Continue For
    Case "end"
      category = CATEND
      Continue For
    Case "actionlist"
      category = CATACTION
      Continue For
    Case Else
      ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
      MainForm.Close()
  End Select



ElseIf category = CATINFO Then
  Select Case tagList(0)
    Case "/info"
      category = CATNONE
      Continue For
    Case "name"
      taskName = Utilities.readTextFromTaglist(tagList, rowCounter, fileNameAndDir)
      Continue For
    Case Else
      ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
      MainForm.Close()
  End Select



ElseIf category = CATTRIGGER Then
  Select Case tagList(0)
    Case "/trigger"
      category = CATNONE
      Continue For

    Case "varLess", "varMore", "varEqual", "varMoreEqual", "varLessEqual", "varNotEqual"
```

```
                Triggers.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter, fileNameAndDir))
                Continue For

            Case "setVar", "addVar", "subVar"
                TriggerSetVar.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
                Continue For

            Case "incVar", "decVar"
                TriggerSetVar.Add(Utilities.readVarnameFromTaglist(tagList, rowCounter, fileNameAndDir))
                Continue For

            Case "click"
                If TriggerHasClick Then
                    ErrorHandler.ShowTaskMultipleClickError(rowCounter, fileNameAndDir)
                    MainForm.Close()
                End If
                Triggers.Add(Utilities.readClickConditionFromTaglist(tagList, rowCounter, fileNameAndDir))
                TriggerHasClick = True
                Continue For

            Case "timer"
                If TriggerTimerTarget <> -1 Then
                    ErrorHandler.ShowMultipleTimerError(rowCounter, fileNameAndDir)
                    MainForm.Close()
                End If
                Dim timerCondition As Utilities.condition = Utilities.readTimerConditionFromTaglist(tagList,
rowCounter, fileNameAndDir)
                TriggerTimerTarget = CSng(timerCondition.info1)
                Continue For

            Case "triggerOnce"
                TriggerOnce = Utilities.readBoolFromTaglist(tagList, rowCounter, fileNameAndDir)
                Continue For

            Case Else
                ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
                MainForm.Close()
        End Select



    ElseIf category = CATFAIL Then
        Select Case tagList(0)
            Case "/fail"
                category = CATNONE
                Continue For

            Case "varLess", "varMore", "varEqual", "varMoreEqual", "varLessEqual", "varNotEqual"
                Fails.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter, fileNameAndDir))
                Continue For

            Case "setVar", "addVar", "subVar"
                FailsSetVar.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
```

```
            Continue For

          Case "incVar", "decVar"
            FailsSetVar.Add(Utilities.readVarnameFromTaglist(tagList, rowCounter, fileNameAndDir))
            Continue For

          Case "click"
            Fails.Add(Utilities.readClickConditionFromTaglist(tagList, rowCounter, fileNameAndDir))
            FailHasClick = True
            Continue For

          Case "timer"
            If FailTimerTarget <> -1 Then
              ErrorHandler.ShowMultipleTimerError(rowCounter, fileNameAndDir)
              MainForm.Close()
            End If

            Dim timerCondition As Utilities.condition = Utilities.readTimerConditionFromTaglist(tagList,
rowCounter, fileNameAndDir)
            FailTimerTarget = CSng(timerCondition.info1)
            Continue For

          Case Else
            ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
            MainForm.Close()
        End Select



      ElseIf category = CATEND Then
        Select Case tagList(0)
          Case "/end"
            category = CATNONE
            Continue For

          Case "actionlist"
            If tagList.Length <> 1 Then
              ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
              MainForm.Close()
            End If

            HasActionList = True
            Continue For

          Case "varLess", "varMore", "varEqual", "varMoreEqual", "varLessEqual", "varNotEqual"
            Ends.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter, fileNameAndDir))
            Continue For

          Case "setVar", "addVar", "subVar"
            EndsSetVar.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
            Continue For

          Case "incVar", "decVar"
            EndsSetVar.Add(Utilities.readVarnameFromTaglist(tagList, rowCounter, fileNameAndDir))
```

```
            Continue For

        Case "click"
            If EndHasClick Then
                ErrorHandler.ShowTaskMultipleClickError(rowCounter, fileNameAndDir)
                MainForm.Close()
            End If
            Ends.Add(Utilities.readClickConditionFromTaglist(tagList, rowCounter, fileNameAndDir))
            EndHasClick = True
            Continue For

        Case "timer"
            If EndTimerTarget <> -1 Then
                ErrorHandler.ShowMultipleTimerError(rowCounter, fileNameAndDir)
                MainForm.Close()
            End If

            Dim timerCondition As Utilities.condition = Utilities.readTimerConditionFromTaglist(tagList,
rowCounter, fileNameAndDir)
            EndTimerTarget = CSng(timerCondition.info1)
            Continue For

        Case Else
            ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
            MainForm.Close()
    End Select



ElseIf category = CATACTION Then
    If Not HasActionList Then
        Continue For
    End If

    Select Case tagList(0)
        Case "/actionlist"
            category = CATNONE
            Continue For

        Case "click"
            actionList.Add(Utilities.readClickConditionFromTaglist(tagList, rowCounter, fileNameAndDir))
            Continue For

        Case "varLess", "varMore", "varEqual", "varMoreEqual", "varLessEqual", "varNotEqual"
            actionList.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
            Continue For

        Case "setVar", "addVar", "subVar"
            actionList.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
            Continue For

        Case "incVar", "decVar"
            actionList.Add(Utilities.readVarnameFromTaglist(tagList, rowCounter, fileNameAndDir))
```

```
                Continue For

             Case Else
                ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
                MainForm.Close()
          End Select

       Else

       End If

    Next


 End Sub

 ''' <summary>
 ''' Reset the task, making it possible to restart.
 ''' </summary>
 ''' <remarks></remarks>
 Private Sub resetTask()
    triggerTimerCounter = 0
    failTimerCounter = 0
    endTimerCounter = 0
    actionListCounter = 0

    TriggerTimerDone = False
    FailTimerDone = False
    EndTimerDone = False
    actionListDone = False

    taskID = -1
    taskRunning = False

 End Sub

 ''' <summary>
 ''' Start the task.
 ''' </summary>
 ''' <remarks></remarks>
 Private Sub StartTask()

    'Console.WriteLine("Task: " + Now.ToString("MM/dd/yyyy HH:mm:ss.fffffff") + " " +
CStr(Utilities.stopWatchItem.ElapsedTicks / TimeSpan.TicksPerMillisecond))

    If TriggerOnce And HasTriggered Then
       Return
    End If

    HasTriggered = True

    taskID = MainForm.GetActionID()
    Utilities.eventSetVars(TriggerSetVar)
    MainForm.WriteEventLog("TaskStarted", taskName + "|" + CStr(taskID))
    taskRunning = True
```

```
End Sub

''' <summary>
''' End the task.
''' </summary>
''' <remarks></remarks>
Private Sub EndTask()

    taskRunning = False
    MainForm.WriteEventLog("TaskCompleted", taskName + "|" + CStr(taskID))
    Utilities.eventSetVars(EndsSetVar)
    resetTask()
End Sub

''' <summary>
''' Fail the task.
''' </summary>
''' <param name="reason">The reason the task failed, printed to log.</param>
''' <remarks></remarks>
Private Sub FailTask(ByVal reason As String)
    taskRunning = False
    MainForm.WriteEventLog("TaskFailed", taskName + "|" + CStr(taskID) + "|" + reason)
    Utilities.eventSetVars(FailsSetVar)
    resetTask()
End Sub

''' <summary>
''' Calls all tasks to update their timed events.
''' </summary>
''' <remarks></remarks>
Public Shared Sub updateTimerTick()
    For Each taskObject In MainForm.taskList
        taskObject.updateTimerLocal()
    Next

End Sub

''' <summary>
''' Update the timed events for the current task.
''' </summary>
''' <remarks></remarks>
Private Sub updateTimerLocal()
    If updateWithTimer Then
        checkTriggerVars()
        checkFailsVars()
        checkEndsVars()
        checkActionListVars()
    End If

    UpdateTimerCounters()

End Sub

Public Shared Sub updateState()
```

```
    If updateOnDemand Then
      For Each taskObject In MainForm.taskList
        taskObject.updateStateLocal()
      Next
    End If
  End Sub

  Private Sub updateStateLocal()
    checkTriggerVars()
    checkFailsVars()
    checkEndsVars()
    checkActionListVars()
  End Sub

  ''' <summary>
  ''' Update the timers and their trigged events
  ''' </summary>
  ''' <remarks></remarks>
  Private Sub UpdateTimerCounters()
    If Not taskRunning And TriggerTimerTarget <> -1 Then
      triggerTimerCounter += timerInterval
      If triggerTimerCounter >= TriggerTimerTarget Then
        TriggerTimerDone = True
      End If
    ElseIf taskRunning And (FailTimerTarget <> -1 Or EndTimerTarget <> -1) Then
      failTimerCounter += timerInterval
      endTimerCounter += timerInterval
      If failTimerCounter >= FailTimerTarget And FailTimerTarget <> -1 Then
        FailTimerDone = True
        FailTask("Time out")
        Return
      End If
      If endTimerCounter >= EndTimerTarget And EndTimerTarget <> -1 Then
        EndTimerDone = True
      End If
    End If
    updateStateLocal()
  End Sub

  ''' <summary>
  ''' Check the conditions if the task should trigger
  ''' </summary>
  ''' <remarks></remarks>
  Private Sub checkTriggerVars()

    If taskRunning Then
      Return
    End If
    If TriggerHasClick Then
      Return
    End If
    If (Not TriggerTimerDone) And TriggerTimerTarget <> -1 Then
      Return
    End If
```

```
    If Utilities.checkAllConditionsHolds(Triggers) Then
      StartTask()
    End If

  End Sub

  ''' <summary>
  ''' Check the conditions if the task should fail
  ''' </summary>
  ''' <remarks></remarks>
  Private Sub checkFailsVars()
    If Not taskRunning Then
      Return
    End If

    For Each condition In Fails
      Dim type As String = condition.type
      If type = "varLess" Then
        If VariableManager.readVar(CStr(condition.info1)) < CSng(condition.info2) Then
          FailTask("Condition " + CStr(condition.info1) + " less than " + CStr(condition.info2) + " (" +
CStr(VariableManager.readVar(CStr(condition.info1))) + ").")
          Return
        End If

      ElseIf type = "varMore" Then
        If VariableManager.readVar(CStr(condition.info1)) > CSng(condition.info2) Then
          FailTask("Condition " + CStr(condition.info1) + " more than " + CStr(condition.info2) + " (" +
CStr(VariableManager.readVar(CStr(condition.info1))) + ").")
          Return
        End If

      ElseIf type = "varEqual" Then
        If VariableManager.readVar(CStr(condition.info1)) = CSng(condition.info2) Then
          FailTask("Condition " + CStr(condition.info1) + " equal to " + CStr(condition.info2) + " (" +
CStr(VariableManager.readVar(CStr(condition.info1))) + ").")
          Return
        End If

      ElseIf type = "varMoreEqual" Then
        If VariableManager.readVar(CStr(condition.info1)) >= CSng(condition.info2) Then
          FailTask("Condition " + CStr(condition.info1) + " more than or equal to " + CStr(condition.info2) + " ("
+ CStr(VariableManager.readVar(CStr(condition.info1))) + ").")
          Return
        End If

      ElseIf type = "varLessEqual" Then
        If VariableManager.readVar(CStr(condition.info1)) <= CSng(condition.info2) Then
          FailTask("Condition " + CStr(condition.info1) + " less than or equal to " + CStr(condition.info2) + " (" +
CStr(VariableManager.readVar(CStr(condition.info1))) + ").")
          Return
        End If

      ElseIf type = "varNotEqual" Then
        If VariableManager.readVar(CStr(condition.info1)) <> CSng(condition.info2) Then
```

```
          FailTask("Condition " + CStr(condition.info1) + " not equal to " + CStr(condition.info2) + " (" +
CStr(VariableManager.readVar(CStr(condition.info1))) + ").")
          Return
        End If

      End If
    Next

  End Sub

  ''' <summary>
  ''' Check the conditions if the task should end
  ''' </summary>
  ''' <remarks></remarks>
  Private Sub checkEndsVars()
    If Not taskRunning Then
      Return
    End If
    If EndHasClick Then
      Return
    End If
    If (Not EndTimerDone) And EndTimerTarget <> -1 Then
      Return
    End If
    If (HasActionList) And (Not actionListDone) Then
      Return
    End If

    If Utilities.checkAllConditionsHolds(Ends) Then
      EndTask()
    End If

  End Sub

  ''' <summary>
  ''' Update the ActionList
  ''' </summary>
  ''' <remarks></remarks>
  Private Sub checkActionListVars()

    If Not taskRunning Then
      Return
    End If

    If Not HasActionList Then
      Return
    End If

    If actionListDone Then
      Return
    End If

    Dim currentTaskItem As Utilities.condition = actionList(actionListCounter)
```

```
    Select Case currentTaskItem.type
      Case "varLess"
        If Not VariableManager.readVar(CStr(currentTaskItem.info1)) < CSng(currentTaskItem.info2) Then
          Return
        End If
      Case "varMore"
        If Not VariableManager.readVar(CStr(currentTaskItem.info1)) > CSng(currentTaskItem.info2) Then
          Return
        End If
      Case "varEqual"
        If Not VariableManager.readVar(CStr(currentTaskItem.info1)) = CSng(currentTaskItem.info2) Then
          Return
        End If
      Case "varLessEqual"
        If Not VariableManager.readVar(CStr(currentTaskItem.info1)) <= CSng(currentTaskItem.info2) Then
          Return
        End If
      Case "varMoreEqual"
        If Not VariableManager.readVar(CStr(currentTaskItem.info1)) >= CSng(currentTaskItem.info2) Then
          Return
        End If
      Case "varNotEqual"
        If Not VariableManager.readVar(CStr(currentTaskItem.info1)) <> CSng(currentTaskItem.info2) Then
          Return
        End If
      Case "setVar", "addVar", "subVar", "incVar", "decVar"
        Dim tempList As New List(Of Utilities.condition)
        tempList.Add(currentTaskItem)
        Utilities.eventSetVars(tempList)
      Case Else
        Return
    End Select

    actionListCounter += 1
    MainForm.WriteEventLog("ActionListVar", taskName + "|" + taskID.ToString + "|" + currentTaskItem.info1)
    If actionListCounter >= actionList.Count Then
      MainForm.WriteEventLog("ActionListDone", taskName + "|" + taskID.ToString)
      actionListDone = True
    End If
    updateState()

  End Sub

  ''' <summary>
  ''' Trigger click conditions for all tasks with the given control
  ''' </summary>
  ''' <param name="controlName">Name of the clicked control</param>
  ''' <remarks></remarks>
  Public Shared Sub reportClick(ByVal controlName As String)
    For Each taskObject In MainForm.taskList
      taskObject.LocalReportClick(controlName)
    Next
  End Sub

  ''' <summary>
```

```
''' Trigger click conditions for the current task with the given control
''' </summary>
''' <param name="controlname">Name of the clicked control</param>
''' <remarks></remarks>
Private Sub LocalReportClick(ByVal controlname As String)

  'Reverse order to not cause any condition to be fulfilled at the same time as a task is started
  clickActionlist(controlname)
  clickEnds(controlname)
  clickFails(controlname)
  clickTrigger(controlname)

End Sub

''' <summary>
''' Check if the task should trigger when the control is clicked, and if so, trigger it.
''' </summary>
''' <param name="controlname">Name of the clicked control</param>
''' <remarks></remarks>
Private Sub clickTrigger(ByVal controlname As String)

  If taskRunning Then
    Return
  End If

  If (Not TriggerTimerDone) And TriggerTimerTarget <> -1 Then
    Return
  End If

  If Not TriggerHasClick Then
    Return
  End If

  For Each condition In Triggers
    If Not condition.type = "click" Then
      Continue For
    End If
    If condition.info1 <> controlname Then
      Continue For
    End If

    If Utilities.checkAllConditionsHolds(Triggers) Then
      StartTask()
    End If

  Next

End Sub

''' <summary>
''' Check if the task should fail when the control is clicked, and if so, fail it.
''' </summary>
''' <param name="controlname">Name of the clicked control</param>
''' <remarks></remarks>
Private Sub clickFails(ByVal controlname As String)
```

```
        If Not taskRunning Then
            Return
        End If

        If Not FailHasClick Then
            Return
        End If

        For Each condition In Fails
            If Not condition.type = "click" Then
                Continue For
            End If
            If condition.info1 <> controlname Then
                Continue For
            End If

            FailTask("User clicked on control: " + controlname)

        Next

    End Sub

    ''' <summary>
    ''' Check if the task should end when the control is clicked, and if so, end it.
    ''' </summary>
    ''' <param name="controlname">Name of the clicked control</param>
    ''' <remarks></remarks>
    Private Sub clickEnds(ByVal controlname As String)

        If Not taskRunning Then
            Return
        End If

        If Not EndHasClick Then
            Return
        End If

        If (Not EndTimerDone) And EndTimerTarget <> -1 Then
            Return
        End If

        If HasActionList And Not actionListDone Then
            Return
        End If

        For Each condition In Ends
            If Not condition.type = "click" Then
                Continue For
            End If
            If condition.info1 <> controlname Then
                Continue For
            End If

            If Utilities.checkAllConditionsHolds(Triggers) Then
```

```
        EndTask()
      End If

    Next

  End Sub

  ''' <summary>
  ''' Check if the actionlist should progress when the control is clicked, and if so, progress it.
  ''' </summary>
  ''' <param name="controlname">Name of the clicked control</param>
  ''' <remarks></remarks>
  Private Sub clickActionlist(ByVal controlname As String)

    If Not taskRunning Then
      Return
    End If

    If Not HasActionList Then
      Return
    End If

    If actionListDone Then
      Return
    End If

    Dim currentTaskItem As Utilities.condition = actionList(actionListCounter)

    If currentTaskItem.type <> "click" Then
      Return
    End If

    If currentTaskItem.info1 <> controlname Then
      Return
    End If

    actionListCounter += 1
    MainForm.WriteEventLog("ActionListClick", taskName + "|" + taskID.ToString + "|" + currentTaskItem.info1)
    If actionListCounter >= actionList.Count Then
      MainForm.WriteEventLog("ActionListDone", taskName + "|" + taskID.ToString)
      actionListDone = True
      updateState()
    End If

  End Sub

End Class
```

AlarmManager

Filename: AlarmManager.vb

```
''' <summary>
''' Manages the alarms.
''' Inkludes all functions and collections regarding alarms.
''' </summary>
```

```
''' <remarks></remarks>
Public Class AlarmManager

    Public Shared updateWithTimer As Boolean = True
    Public Shared updateOnDemand As Boolean = True




    Private Const CATNONE = 0
    Private Const CATINFO = 1
    Private Const CATTRIGGER = 2
    Private Const CATCONFIRMATION = 3
    Private Const CATHANDLING = 8
    Private Const CATSOUND = 4
    Private Const CATMESSAGEBOX = 5
    Private Const CATCONTROL = 6
    Private Const CATSTATUS = 7

    Private Shared AlarmList As New List(Of AlarmItem)

    ''' <summary>
    ''' Contains all information about one alarm.
    ''' </summary>
    ''' <remarks></remarks>
    Public Class AlarmItem
        Public name As String
        Public triggerOnce As Boolean
        Public triggerVariables As New List(Of Utilities.condition)
        Public triggerClicks As New List(Of Utilities.condition)
        Public triggerTimerTarget As Single
        Public triggerSetVar As New List(Of Utilities.condition)
        Public confirmationRules As New confirmRule
        Public handlingRules As New handlingRule
        Public showRules As New showRule
        Public status As New alarmStatus

        Public Sub New()

            With Me
                .name = "Default Alarm Name"
                .triggerVariables = New List(Of Utilities.condition)
                .triggerSetVar = New List(Of Utilities.condition)
                .triggerOnce = False
                .triggerTimerTarget = -1

                With .confirmationRules
                    .confirmationNeeded = False
                    .confirmWithMessagebox = False
                    .confirmWithControl = False
                    '.confirmationControlName = ""
                    .confirmWithCondition = False
                    .confirmationVariableList = New List(Of Utilities.condition)
                    .confirmationClickList = New List(Of Utilities.condition)
                    .confirmationSetVar = New List(Of Utilities.condition)
```

```
        End With

        With .handlingRules
          .handlingNeeded = False
          '.handlingWithMessagebox = False
          .handlingWithControl = False
          '.handlingControlName = ""
          .handlingWithCondition = False
          .handlingVariableList = New List(Of Utilities.condition)
          .handlingClickList = New List(Of Utilities.condition)
          .handlingSetVar = New List(Of Utilities.condition)
        End With

        With .showRules.soundRules
          .useSound = False
          .sound = ""
          .doLoop = False
        End With

        With .showRules.messageBoxRules
          .useMessageBox = False
          .text = ""
          .caption = ""
        End With

        With .status
          .alarmID = -1
          .soundID = -1
          .triggered = False
          .confirmed = False
          .triggerTimer = 0
          .hasTriggered = False
        End With

      End With

  End Sub
End Class
''' <summary>
''' Rules regarding confirmation of the alarm.
''' </summary>
''' <remarks></remarks>
Public Class confirmRule
  Public confirmationNeeded As Boolean
  Public confirmWithMessagebox As Boolean

  Public confirmWithControl As Boolean
  'Public confirmationControlName As String

  Public confirmWithCondition As Boolean
  Public confirmationVariableList As New List(Of Utilities.condition)
  Public confirmationClickList As New List(Of Utilities.condition)
  Public confirmationSetVar As New List(Of Utilities.condition)
End Class
```

```
''' <summary>
''' Rules regarding handling of the alarm.
''' </summary>
''' <remarks></remarks>
Public Class handlingRule
    Public handlingNeeded As Boolean
    'Public handlingWithMessagebox As Boolean

    Public handlingWithControl As Boolean
    'Public handlingControlName As String

    Public handlingWithCondition As Boolean
    Public handlingVariableList As New List(Of Utilities.condition)
    Public handlingClickList As New List(Of Utilities.condition)
    Public handlingSetVar As New List(Of Utilities.condition)
End Class


''' <summary>
''' Rules regarding how to show the alarm.
''' </summary>
''' <remarks></remarks>
Public Class showRule
    Public soundRules As New soundRule
    Public messageBoxRules As New messageBoxRule
End Class

''' <summary>
''' Status information of the alarm.
''' </summary>
''' <remarks></remarks>
Public Class alarmStatus
    Public triggered As Boolean
    Public confirmed As Boolean
    Public alarmID As Integer
    Public soundID As Integer
    Public triggerTimer As Integer
    Public hasTriggered As Boolean
End Class

''' <summary>
''' Rules regarding the usage of message boxes in the alarm.
''' </summary>
''' <remarks></remarks>
Public Class messageBoxRule
    Public useMessageBox As Boolean
    Public text As String
    Public caption As String
End Class

''' <summary>
''' Rules regarding the usage of sounds in the alarm.
''' </summary>
''' <remarks></remarks>
Public Class soundRule
```

```
    Public useSound As Boolean
    Public sound As String
    Public doLoop As Boolean
End Class

''' <summary>
''' Initializes the alarm manager and populates it with alarms.
''' </summary>
''' <remarks></remarks>
Shared Sub initializeAlarmSystem()

    findAndAddAlarms()

End Sub


''' <summary>
''' Populates the alarm manager with alarms.
''' </summary>
''' <remarks></remarks>
Private Shared Sub findAndAddAlarms()
    Dim alarmFolderDirLocal As String = MainForm.alarmFolderDir
    Dim AlarmFiles() As String = My.Computer.FileSystem.GetFiles(alarmFolderDirLocal).ToArray()

    For Each alarmName In AlarmFiles
        If alarmName.Substring(alarmName.Length - 4) = MainForm.alarmFileEnding Then
            Dim curAlarm As New AlarmItem
            curAlarm = readAlarm(alarmName)
            cleanupAlarm(curAlarm)
            AlarmList.Add(curAlarm)
            MainForm.WriteSystemLog("Loaded alarm: " + alarmName)
        End If
    Next


    '' Debug
    'For Each curAlarm As AlarmItem In AlarmList
    '    Console.WriteLine("--------------------------------")
    '    Console.WriteLine("Name: " & curAlarm.name)
    '    Console.WriteLine("Timer: " & curAlarm.triggerTimerTarget.ToString)
    '    Console.WriteLine("TriggerOnce: " & curAlarm.triggerOnce.ToString)
    '    For Each variable As Utilities.condition In curAlarm.triggerVariables
    '        Console.WriteLine(variable.type & " : " & variable.info1 & " : " & variable.info2)
    '    Next
    'Next

End Sub

''' <summary>
''' Read alarm from file and return it as AlarmItem.
''' </summary>
''' <param name="fileNameAndDir">File name and directory of the alarm file.</param>
''' <returns>Alarm in the shape of an AlarmItem</returns>
''' <remarks></remarks>
Private Shared Function readAlarm(ByVal fileNameAndDir As String) As AlarmItem
```

```
Dim curAlarm As New AlarmItem
Dim textRow() As String = My.Computer.FileSystem.ReadAllText(fileNameAndDir).Split(vbCrLf)
Dim rowCounter As Integer
Dim category As Integer = CATNONE

For rowCounter = 0 To textRow.GetLength(0) - 1
  textRow(rowCounter) = textRow(rowCounter).Trim()

  If textRow(rowCounter).Length = 0 Then
    Continue For
  ElseIf textRow(rowCounter).Substring(0, 1) = "'" Then
    Continue For
  ElseIf textRow(rowCounter).Substring(0, 1) <> "[" Then
    ErrorHandler.ShowInitialSignError(rowCounter, fileNameAndDir)
    MainForm.Close()
  End If

  Dim tagEnd As Integer = textRow(rowCounter).IndexOf("]")
  If tagEnd = -1 Then
    ErrorHandler.ShowNoEndSignError(rowCounter, fileNameAndDir)
    MainForm.Close()
  End If

  Dim tagContents As String = textRow(rowCounter).Substring(1, tagEnd - 1)
  Dim tagList() As String = tagContents.Split(":")

  If tagList.Length < 1 Then
    ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
    MainForm.Close()
  End If
  If tagList.Length > 3 Then
    ErrorHandler.ShowParamNumError(rowCounter, fileNameAndDir)
    MainForm.Close()
  End If


  'info
  '--- name

  'trigger
  '--- conditionlist
  '--- triggerOnce
  '--- timer

  'confirmation
  '--- confirmationNeeded
  '--- confirmWithMessagebox
  '--- confirmWithControl
  '--- confirmationControlName
  '--- confirmWithVariable
  '--- confirmationVariableList

  'handling
  '--- handlingNeeded
```

```
'--- handlingWithMessagebox
'--- handlingWithControl
'--- handlingControlName
'--- handlingWithVariable
'--- handlingVariableList

'sound
'--- useSound
'--- sound
'--- doLoop

'messageBox
'--- useMessageBox
'--- text
'--- caption

If category = CATNONE Then
   Select Case tagContents
      Case "info"
         category = CATINFO
         Continue For
      Case "trigger"
         category = CATTRIGGER
         Continue For
      Case "confirmation"
         category = CATCONFIRMATION
         Continue For
      Case "handling"
         category = CATHANDLING
         Continue For
      Case "sound"
         category = CATSOUND
         Continue For
      Case "messageBox"
         category = CATMESSAGEBOX
         Continue For
         'Case "control"
         '   category = CATCONTROL
         '   Continue For
         'Case "status"
         '   category = CATSTATUS
         '   Continue For
      Case Else
         ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
         MainForm.Close()
   End Select

   'info
   '--- name

ElseIf category = CATINFO Then
   Select Case tagList(0)
      Case "/info"
         category = CATNONE
         Continue For
```

```
            Case "name"
               curAlarm.name = Utilities.readTextFromTaglist(tagList, rowCounter, fileNameAndDir)
               Continue For
            Case Else
               ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
               MainForm.Close()
         End Select

         'trigger
         '--- conditionlist
         '--- triggerOnce
         '--- timer

      ElseIf category = CATTRIGGER Then
         Select Case tagList(0)
            Case "/trigger"
               category = CATNONE
               Continue For

            Case "varLess", "varMore", "varEqual", "varMoreEqual", "varLessEqual", "varNotEqual"
               curAlarm.triggerVariables.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
               Continue For

            Case "setVar", "addVar", "subVar"
               curAlarm.triggerSetVar.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
               Continue For

            Case "incVar", "decVar"
               curAlarm.triggerSetVar.Add(Utilities.readVarnameFromTaglist(tagList, rowCounter,
fileNameAndDir))
               Continue For

            Case "click"
               curAlarm.triggerClicks.Add(Utilities.readClickConditionFromTaglist(tagList, rowCounter,
fileNameAndDir))
               Continue For

            Case "timer"
               If curAlarm.triggerTimerTarget <> -1 Then
                  ErrorHandler.ShowMultipleTimerError(rowCounter, fileNameAndDir)
                  MainForm.Close()
               End If
               Dim timerCondition As Utilities.condition = Utilities.readTimerConditionFromTaglist(tagList,
rowCounter, fileNameAndDir)
               curAlarm.triggerTimerTarget = CSng(timerCondition.info1)
               Continue For

            Case "triggerOnce"
               curAlarm.triggerOnce = Utilities.readBoolFromTaglist(tagList, rowCounter, fileNameAndDir)
               Continue For

            Case Else
               ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
```

```
            MainForm.Close()
        End Select

        'confirmation
        '--- confirmationNeeded
        '--- confirmWithMessagebox
        '--- confirmWithControl
        '--- confirmationControlName
        '--- confirmWithConditions
        '--- confirmationVariableList
        '--- confirmationClickList

    ElseIf category = CATCONFIRMATION Then
        With curAlarm.confirmationRules

            Select Case tagList(0)
                Case "/confirmation"
                    category = CATNONE
                    Continue For

                Case "useMessagebox"
                    .confirmWithMessagebox = Utilities.readBoolFromTaglist(tagList, rowCounter, fileNameAndDir)

                    'Case "controlName"
                    '    .confirmationControlName = Utilities.readControlFromTaglist(tagList, rowCounter,
fileNameAndDir)
                    '    .confirmWithControl = True

                    'confirmationVariableList
                Case "varLess", "varMore", "varEqual", "varMoreEqual", "varLessEqual", "varNotEqual"
                    .confirmationVariableList.Add(Utilities.readVarnameAndNumberFromTaglist(tagList,
rowCounter, fileNameAndDir))
                    .confirmWithCondition = True

                Case "setVar", "addVar", "subVar"
                    .confirmationSetVar.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))

                Case "incVar", "decVar"
                    .confirmationSetVar.Add(Utilities.readVarnameFromTaglist(tagList, rowCounter,
fileNameAndDir))

                Case "click"
                    .confirmationClickList.Add(Utilities.readClickConditionFromTaglist(tagList, rowCounter,
fileNameAndDir))
                    .confirmWithControl = True

                Case Else
                    ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
                    MainForm.Close()
            End Select

            If .confirmWithMessagebox Or .confirmWithControl Or .confirmWithCondition Then
                .confirmationNeeded = True
            End If
```

```
        Continue For

      End With

      'handling
      '--- handlingNeeded
      '--- handlingWithMessagebox
      '--- handlingWithControl
      '--- handlingControlName
      '--- handlingWithConditions
      '--- handlingVariableList
      '---handlingClickList

    ElseIf category = CATHANDLING Then
      With curAlarm.handlingRules
        Select Case tagList(0)
          Case "/handling"
            category = CATNONE
            Continue For

          'Case "useMessagebox"
          '   .handlingWithMessagebox = Utilities.readBoolFromTaglist(tagList, rowCounter, fileNameAndDir)

            'Case "controlName"
            '   .handlingControlName = Utilities.readControlFromTaglist(tagList, rowCounter,
fileNameAndDir)
            '   .handlingWithControl = True

            'handlingVariableList
          Case "varLess", "varMore", "varEqual", "varMoreEqual", "varLessEqual", "varNotEqual"
            .handlingVariableList.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))
            .handlingWithCondition = True

          Case "setVar", "addVar", "subVar"
            .handlingSetVar.Add(Utilities.readVarnameAndNumberFromTaglist(tagList, rowCounter,
fileNameAndDir))

          Case "incVar", "decVar"
            .handlingSetVar.Add(Utilities.readVarnameFromTaglist(tagList, rowCounter, fileNameAndDir))

          Case "click"
            If .handlingClickList.Count >= 1 Then
              ErrorHandler.ShowAlarmMultipleClickError(rowCounter, fileNameAndDir)
              MainForm.Close()
            End If
            .handlingClickList.Add(Utilities.readClickConditionFromTaglist(tagList, rowCounter,
fileNameAndDir))
            .handlingWithControl = True

          Case Else
            ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
            MainForm.Close()
        End Select
```

```
        If .handlingWithControl Or .handlingWithCondition Then '.handlingWithMessagebox Or
            .handlingNeeded = True
        End If
    End With
    Continue For

    'sound
    '--- useSound
    '--- sound
    '--- doLoop

ElseIf category = CATSOUND Then
    With curAlarm.showRules.soundRules
        Select Case tagList(0)
            Case "/sound"
                category = CATNONE
                Continue For

            Case "useSound"
                .useSound = Utilities.readBoolFromTaglist(tagList, rowCounter, fileNameAndDir)
                Continue For

            Case "soundName"
                .sound = Utilities.readSoundFromTaglist(tagList, rowCounter, fileNameAndDir)
                Continue For

            Case "doLoop"
                .doLoop = Utilities.readBoolFromTaglist(tagList, rowCounter, fileNameAndDir)
                Continue For

            Case Else
                ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
                MainForm.Close()
        End Select
    End With

    'messageBox
    '--- useMessageBox
    '--- text
    '--- caption

ElseIf category = CATMESSAGEBOX Then
    With curAlarm.showRules.messageBoxRules
        Select Case tagList(0)
            Case "/messageBox"
                category = CATNONE
                Continue For

            Case "useMessagebox"
                .useMessageBox = Utilities.readBoolFromTaglist(tagList, rowCounter, fileNameAndDir)
                Continue For

            Case "text"
                .text = Utilities.readTextFromTaglist(tagList, rowCounter, fileNameAndDir)
                Continue For
```

```
            Case "caption"
              .caption = Utilities.readTextFromTaglist(tagList, rowCounter, fileNameAndDir)
              Continue For

            Case Else
              ErrorHandler.ShowTagError(rowCounter, fileNameAndDir, tagList(0))
              MainForm.Close()
          End Select
        End With
      Else

      End If

    Next

    Return curAlarm
  End Function

  ''' <summary>
  ''' Clean up the alarm from erroneus or unneeded logic. Prints errors to the system log but doesn't interrupt
the program.
  ''' </summary>
  ''' <param name="alarm">The alarm to be cleaned up.</param>
  ''' <remarks></remarks>
  Private Shared Sub cleanupAlarm(ByRef alarm As AlarmItem)

    'Standard interactions
    With alarm.confirmationRules
      If .confirmationNeeded = False Then
        If .confirmWithMessagebox <> False Then
          .confirmWithMessagebox = False
          MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", confirmWithMessagebox was set even
if no confirmation was needed. Automatically unsetting.")
        End If
        If .confirmWithControl <> False Then
          .confirmWithControl = False
          MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", confirmWithControl was set even if no
confirmation was needed. Automatically unsetting.")
        End If
        'If .confirmationControlName <> "" Then
        '    .confirmationControlName = ""
        '    MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", confirmationControlName was set
even if no confirmation was needed. Automatically unsetting.")
        'End If
        If .confirmWithCondition <> False Then
          .confirmWithCondition = False
          MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", confirmWithCondition was set even if
no confirmation was needed. Automatically unsetting.")
        End If
        If .confirmationVariableList.Count > 0 Then
          .confirmationVariableList = New List(Of Utilities.condition)
          MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", confirmation variable conditions were
set even if no confirmation was needed. Automatically unsetting.")
        End If
```

```
            If .confirmationClickList.Count > 0 Then
              .confirmationClickList = New List(Of Utilities.condition)
              MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", confirmation click conditions were set
even if no confirmation was needed. Automatically unsetting.")
            End If
            If .confirmationSetVar.Count > 0 Then
              .confirmationSetVar = New List(Of Utilities.condition)
              MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", trying to set variables on confirmation
even if no confirmation was needed. Automatically unsetting.")
            End If
          End If
        End With

        With alarm.handlingRules
          If .handlingNeeded = False Then
            'If .handlingWithMessagebox <> False Then
            '    .handlingWithMessagebox = False
            '    MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", handlingWithMessagebox was set
even if no handling was needed. Automatically unsetting.")
            'End If
            If .handlingWithControl <> False Then
              .handlingWithControl = False
              MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", handlingWithControl was set even if
no handling was needed. Automatically unsetting.")
            End If
            'If .handlingControlName <> "" Then
            '    .handlingControlName = ""
            '    MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", handlingControlName was set even if
no handling was needed. Automatically unsetting.")
            'End If
            If .handlingWithCondition <> False Then
              .handlingWithCondition = False
              MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", handlingWithCondition was set even if
no handling was needed. Automatically unsetting.")
            End If
            If .handlingVariableList.Count > 0 Then
              .handlingVariableList = New List(Of Utilities.condition)
              MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", handling variable conditions were set
even if no handling was needed. Automatically unsetting.")
            End If
            If .handlingClickList.Count > 0 Then
              .handlingClickList = New List(Of Utilities.condition)
              MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", handling click conditions were set
even if no handling was needed. Automatically unsetting.")
            End If
            If .handlingSetVar.Count > 0 Then
              .handlingSetVar = New List(Of Utilities.condition)
              MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", trying to set variables on handling
even if no handling was needed. Automatically unsetting.")
            End If
          End If
        End With

        With alarm.showRules.soundRules
          If .useSound = False Then
```

```
          If .sound <> "" Then
            .sound = ""
            MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", sound was set even if no sound is used.
Automatically unsetting.")
          End If
          If .doLoop <> False Then
            .doLoop = False
            MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", doLoop was set even if no sound is
used. Automatically unsetting.")
          End If
        End If
      End With

      With alarm.showRules.messageBoxRules
        If .useMessageBox = False Then
          If .text <> "" Then
            .text = ""
            MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", text was set even if no message box is
used. Automatically unsetting.")
          End If
          If .caption <> "" Then
            .caption = ""
            MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", caption was set even if no message box
is used. Automatically unsetting.")
          End If
        End If
      End With

      'Special interactions
      If alarm.confirmationRules.confirmationNeeded = False And alarm.handlingRules.handlingNeeded = False
Then
        If alarm.showRules.soundRules.doLoop <> False Then
          alarm.showRules.soundRules.doLoop = False
          MainForm.WriteSystemLog("Error in alarm " + alarm.name + ", sound doLoop was set even though
confirmation and handling aren't needed. Automatically unsetting since sound otherwise will play infinitely.")
        End If
      End If


    End Sub

    ''' <summary>
    ''' Report a clicked control with this function to trigger, confirm or handle alarms depending on their states and
conditions.
    ''' </summary>
    ''' <param name="controlName">The name of the control.</param>
    ''' <remarks></remarks>
    Public Shared Sub reportClick(ByVal controlName As String)
      For index As Integer = 0 To AlarmList.Count - 1
        With AlarmList(index)
          If .status.triggered = True Then
            If .status.confirmed = False Then
              If .confirmationRules.confirmWithControl = True Then
                For Each condition In .confirmationRules.confirmationClickList
                  If condition.type = "click" And condition.info1 = controlName Then
```

```
                    confirmAlarm(index)
                End If
            Next
        End If
    ElseIf .status.confirmed = True Then
        If .handlingRules.handlingWithControl = True Then
            For Each condition In .handlingRules.handlingClickList
                If condition.type = "click" And condition.info1 = controlName Then
                    If Utilities.checkAllConditionsHolds(.handlingRules.handlingVariableList) Then
                        handleAlarm(index)
                    End If
                End If
            Next
        End If
    End If
Else
    For Each condition In .triggerClicks
        If condition.type = "click" And condition.info1 = controlName Then
            triggerAlarm(index)
        End If
    Next
End If
End With
Next
End Sub

''' <summary>
''' Updates all timer based checks, call from a repeating timer.
''' </summary>
''' <remarks></remarks>
Public Shared Sub updateTimerTick()
    If updateWithTimer Then
        checkTriggerVars()
        checkConfirmVars()
        checkHandledVars()
    End If

    updateTimerCounters()
End Sub

Public Shared Sub updateState()
    If updateOnDemand Then
        checkTriggerVars()
        checkConfirmVars()
        checkHandledVars()
    End If

End Sub

''' <summary>
''' Call to check if the conditions to trigger an alarm has been met, and if so, trigger it.
''' </summary>
''' <remarks></remarks>
Private Shared Sub checkTriggerVars()
    For index As Integer = 0 To AlarmList.Count - 1
```

```
      With AlarmList(index)
        If .status.triggered = True Then
          Continue For
        End If

        If Utilities.checkOneConditionHolds(.triggerVariables) Then
          triggerAlarm(index)
        End If
      End With
    Next
  End Sub

  ''' <summary>
  ''' Call to check if the conditions to confirm an alarm has been met, and if so, confirm it.
  ''' </summary>
  ''' <remarks></remarks>
  Private Shared Sub checkConfirmVars()
    For index As Integer = 0 To AlarmList.Count - 1
      If AlarmList(index).status.triggered = False Then
        Continue For
      End If
      If AlarmList(index).status.confirmed = True Then
        Continue For
      End If
      If AlarmList(index).confirmationRules.confirmationNeeded = False Then
        Continue For
      End If
      If AlarmList(index).confirmationRules.confirmWithCondition = False Then
        Continue For
      End If

      If Utilities.checkOneConditionHolds(AlarmList(index).confirmationRules.confirmationVariableList) Then
        confirmAlarm(index)
      End If
    Next

  End Sub

  ''' <summary>
  ''' Call to check if the conditions to handle an alarm has been met, and if so, handle it.
  ''' </summary>
  ''' <remarks></remarks>
  Private Shared Sub checkHandledVars()
    For index As Integer = 0 To AlarmList.Count - 1
      If AlarmList(index).status.triggered = False Then
        Continue For
      End If
      If AlarmList(index).status.confirmed = False Then
        Continue For
      End If
      If AlarmList(index).handlingRules.handlingNeeded = False Then
        Continue For
      End If
      If AlarmList(index).handlingRules.handlingWithCondition = False Then
        Continue For
```

```
      End If
      If AlarmList(index).handlingRules.handlingWithControl = True Then
        Continue For
      End If

      If Utilities.checkAllConditionsHolds(AlarmList(index).handlingRules.handlingVariableList) Then
        handleAlarm(index)
      End If
    Next
  End Sub

  ''' <summary>
  ''' Update the timing counters for triggering the alarms.
  ''' </summary>
  ''' <remarks></remarks>
  Private Shared Sub updateTimerCounters()
    For index As Integer = 0 To AlarmList.Count - 1

      With AlarmList(index)
        If .triggerTimerTarget = -1 Then
          Continue For
        End If

        With .status
          If .triggered = True Then
            Continue For
          End If

          .triggerTimer += Utilities.UpdateTimerInterval
          If .triggerTimer >= AlarmList(index).triggerTimerTarget * 1000 Then
            .triggerTimer = 0
            triggerAlarm(index)
          End If
        End With
      End With

    Next
  End Sub

  ''' <summary>
  ''' Call to trigger an alarm.
  ''' </summary>
  ''' <param name="index">The index of the alarm in the AlarmList.</param>
  ''' <remarks></remarks>
  Public Shared Sub triggerAlarm(ByVal index As Integer)

    'Console.WriteLine("Alarm: " + Now.ToString("MM/dd/yyyy HH:mm:ss.fffffff") + " " +
CStr(Utilities.stopWatchItem.ElapsedTicks / TimeSpan.TicksPerMillisecond))

    With AlarmList(index)
      If .triggerOnce And .status.hasTriggered Then
        Return
      End If
      .status.hasTriggered = True
      .status.alarmID = MainForm.GetActionID()
```

```
        .status.triggered = True
        .status.confirmed = False
        MainForm.WriteEventLog("AlarmTriggered", .name + "|" + .status.alarmID.ToString)
        If .showRules.soundRules.useSound = True Then
          Dim soundID As Integer = SoundManager.PlayNewSound(.showRules.soundRules.sound,
.showRules.soundRules.doLoop, 100)
          .status.soundID = soundID
        End If
        Utilities.eventSetVars(.triggerSetVar)

        If .confirmationRules.confirmationNeeded = False Then
          .status.confirmed = True
        End If
        If .handlingRules.handlingNeeded = False Then
          .status.alarmID = -1
          .status.confirmed = False
          .status.triggered = False
        End If

        If .showRules.messageBoxRules.useMessageBox = True Then
          Dim MSGResult As System.Windows.Forms.DialogResult =
MessageBox.Show(.showRules.messageBoxRules.text, .showRules.messageBoxRules.caption,
MessageBoxButtons.OK)

          If .confirmationRules.confirmWithMessagebox = True Then
            MainForm.WriteEventLog("AlarmMSGBoxReply", AlarmList(index).name + "|" +
AlarmList(index).status.alarmID.ToString + "|" + MSGResult.ToString)
            confirmAlarm(index)
          End If
        End If

      End With
    End Sub

    ''' <summary>
    ''' Call to confirm an alarm.
    ''' </summary>
    ''' <param name="index">The index of the alarm in the AlarmList.</param>
    ''' <remarks></remarks>
    Public Shared Sub confirmAlarm(ByVal index As Integer)
      With AlarmList(index)
        .status.confirmed = True
        MainForm.WriteEventLog("AlarmConfirmed", .name + "|" + .status.alarmID.ToString)
        Utilities.eventSetVars(.confirmationRules.confirmationSetVar)
        If .status.soundID <> -1 Then
          SoundManager.StopAndCloseSound(.status.soundID)
          .status.soundID = -1
        End If
      End With

    End Sub

    ''' <summary>
    ''' Call to handle an alarm.
    ''' </summary>
```

```
    ''' <param name="index">The index of the alarm in the AlarmList.</param>
    ''' <remarks></remarks>
    Public Shared Sub handleAlarm(ByVal index As Integer)

        With AlarmList(index)
            Utilities.eventSetVars(.handlingRules.handlingSetVar)
            MainForm.WriteEventLog("AlarmHandled", .name + "|" + .status.alarmID.ToString)
            If .status.soundID <> -1 Then
                SoundManager.StopAndCloseSound(.status.soundID)
                .status.soundID = -1
            End If
            .status.alarmID = -1
            .status.confirmed = False
            .status.triggered = False
        End With

    End Sub

    ''' <summary>
    ''' Find the index of an alarm, using it's name to search. Returns -1 if the alarm is missing.
    ''' </summary>
    ''' <param name="name">The name of the alarm.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Shared Function findAlarmIndexByName(ByVal name As String) As Integer
        For index As Integer = 0 To AlarmList.Count - 1
            If AlarmList(index).name = name Then
                Return index
            End If
        Next
        Return -1
    End Function

    ''' <summary>
    ''' Returns an array of 2 booleans. The first is true if the alarm is triggered. The second is true if the alarm is
confirmed. They are both false if the alarm is handled or unused.
    ''' </summary>
    ''' <param name="index">The index of the alarm in the AlarmList</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Shared Function getAlarmStatus(ByVal index As Integer) As Boolean()
        Dim returnBoolArr(0 To 1) As Boolean
        If index = -1 Then
            Return {False, False}
        End If
        If index < 0 Then
            MainForm.WriteSystemLog("Tried to read status for alarm with index lower than 0, index = " &
CStr(index) & ". Returnng default status.")
            Return {False, False}
        End If
        If index > AlarmList.Count - 1 Then
            MainForm.WriteSystemLog("Tried to read status for alarm with index higher than the current existing,
index = " & CStr(index) & ", highest index = " & CStr(AlarmList.Count - 1 & ". Returnng default status."))
            Return {False, False}
        End If
```

```
        returnBoolArr(0) = AlarmList(index).status.triggered
        returnBoolArr(1) = AlarmList(index).status.confirmed
        Return returnBoolArr
    End Function

End Class
```

SoundManager

Filename: SoundManager.vb

```
''' <summary>
''' Manages the sounds for the software.
''' </summary>
''' <remarks></remarks>
Public Class SoundManager

    Private Shared curID As Integer = 0
    Public Shared SoundList As New List(Of String)

    'Function to send messages to the media player in order for sounds to start, stop, pause, etc.
    Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal lpstrCommand As
String, ByVal lpstrReturnString As String, ByVal uReturnLength As Integer, ByVal hwndCallback As Integer) As
Integer

    ''' <summary>
    ''' Registers the sounds from the folder.
    ''' </summary>
    ''' <remarks></remarks>
    Shared Sub LoadSounds()

        Dim soundFolderDirLocal As String = MainForm.soundFolderDir
        Dim SoundFiles() As String = My.Computer.FileSystem.GetFiles(soundFolderDirLocal).ToArray()

        For Each soundName In SoundFiles
            If MainForm.soundFileEndings.Contains(soundName.Substring(soundName.Length - 4)) Then
                Dim soundNameList() As String = soundName.Split("\")
                SoundList.Add(soundNameList(soundNameList.Count - 1))
                MainForm.WriteSystemLog("Loaded sound: " + soundName)
            End If
        Next
    End Sub

    ''' <summary>
    ''' Check if a sound with a given name exists.
    ''' </summary>
    ''' <param name="soundName">Name of the sound.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function SoundExist(ByVal soundName As String) As Boolean
        For Each soundListName In SoundList
            If soundListName = soundName Then
                Return True
            End If
        Next
        Return False
```

```
    End Function

    ''' <summary>
    ''' Open the sound file and prepare it for playing. Returns an ID identifying that instance of the sound.
    ''' </summary>
    ''' <param name="soundName">Name of the sound.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function OpenSound(ByVal soundName As String) As Integer

        If Not SoundExist(soundName) Then
            MessageBox.Show("Tried to use a sound that does not exist (" + soundName + ").", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
            MainForm.Close()
        End If

        Dim ID As Integer = curID
        curID = curID + 1

        soundName = MainForm.soundFolderDir + "\" + soundName
        mciSendString("Open " & Chr(34) & soundName & Chr(34) & " type mpegvideo alias " & CStr(ID), "", 0, 0)

        Return ID

    End Function

    ''' <summary>
    ''' Play the sound with a given ID.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
    ''' <param name="doLoop">Should the sound loop until turned off?</param>
    ''' <remarks></remarks>
    Shared Sub PlaySound(ByVal ID As Integer, ByVal doLoop As Boolean)

        If doLoop Then
            mciSendString("play " & CStr(ID) & " repeat", "", 0, 0)
        Else
            mciSendString("play " & CStr(ID), "", 0, 0)
        End If

    End Sub

    ''' <summary>
    ''' Stop the sound. Does not reset the time of the playback to 0.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
    ''' <remarks></remarks>
    Shared Sub StopSound(ByVal ID As Integer)
        mciSendString("stop " & CStr(ID), "", 0, 0)
    End Sub

    ''' <summary>
    ''' Close the sound, making it impossible to play until opened again.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
```

```
    ''' <remarks></remarks>
    Shared Sub CloseSound(ByVal ID As Integer)
        mciSendString("close " & CStr(ID), "", 0, 0)
    End Sub

    ''' <summary>
    ''' Set the playback time, or "seek" to a certain time in the sound.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
    ''' <param name="position">Position in milliseconds.</param>
    ''' <param name="playing">Should the sound be playing afterwards?</param>
    ''' <param name="doLoop">Should the sound loop until stopped?</param>
    ''' <remarks></remarks>
    Shared Sub SoundSetPosition(ByVal ID As Integer, ByVal position As Integer, ByVal playing As Boolean, ByVal
doLoop As Boolean)

        StopSound(ID)
        mciSendString("seek " & CStr(ID) & " to " & CStr(position), "", 0, 0)
        If playing Then
            PlaySound(ID, doLoop)
        End If
    End Sub

    ''' <summary>
    ''' Get the playback status of the sound.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function SoundStatusMode(ByVal ID As Integer) As String

        Dim returnString As String = Space(128)
        mciSendString("status " & CStr(ID) & " mode", returnString, 128, 0)
        Return returnString.Trim

    End Function

    ''' <summary>
    ''' Get the playback time or position of the sound in milliseconds.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function SoundStatusPosition(ByVal ID As Integer) As Integer

        Dim returnString As String = Space(128)
        mciSendString("status " & CStr(ID) & " position", returnString, 128, 0)
        returnString = returnString.Trim
        If IsNumeric(returnString) Then
            Return CInt(returnString)
        End If
        Return 0

    End Function
```

```
''' <summary>
''' Get the length of the sound in milliseconds.
''' </summary>
''' <param name="ID">ID of the sound.</param>
''' <returns></returns>
''' <remarks></remarks>
Shared Function SoundStatusLength(ByVal ID As Integer) As Integer

    Dim returnString As String = Space(128)
    mciSendString("status " & CStr(ID) & " length", returnString, 128, 0)
    returnString = returnString.Trim
    If IsNumeric(returnString) Then
        Return CInt(returnString)
    End If
    Return 0

End Function

''' <summary>
''' Get the volume of the sound, 0-1000.
''' </summary>
''' <param name="ID">ID of the sound.</param>
''' <returns></returns>
''' <remarks></remarks>
Shared Function SoundStatusVolume(ByVal ID As Integer) As Integer

    Dim returnString As String = Space(128)
    mciSendString("status " & CStr(ID) & " volume", returnString, 128, 0)
    returnString = returnString.Trim
    If IsNumeric(returnString) Then
        Return CInt(returnString)
    End If
    Return 0

End Function

''' <summary>
''' Set the volume of the sound.
''' </summary>
''' <param name="ID">ID of the sound.</param>
''' <param name="volume">Volume between 0 and 1000.</param>
''' <remarks></remarks>
Shared Sub SoundSetVolume(ByVal ID As Integer, ByVal volume As Integer)

    If volume < 0 Then
        MainForm.WriteSystemLog("Tried to set volume lower than 0, automatically setting to 0")
        volume = 0
    ElseIf volume > 1000 Then
        MainForm.WriteSystemLog("Tried to set volume higher than 1000, automatically setting to 1000")
        volume = 1000
    End If

    mciSendString("setaudio " & CStr(ID) & " volume to " & CStr(volume), "", 0, 0)

End Sub
```

```
    ''' <summary>
    ''' Open and play a sound at once and set the volume of it. Returns the ID of the instance of the sound.
    ''' </summary>
    ''' <param name="soundName">Name of the sound.</param>
    ''' <param name="doLoop">Should the sound loop until stopped?</param>
    ''' <param name="volume">Volume of the sound, 1-1000.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Shared Function PlayNewSound(ByVal soundName As String, ByVal doLoop As Boolean, ByVal volume As
Integer) As Integer
        Dim ID As Integer = OpenSound(soundName)
        PlaySound(ID, doLoop)
        SoundSetVolume(ID, volume)

        Return ID
    End Function

    ''' <summary>
    ''' Stop and close the sound at once.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
    ''' <remarks></remarks>
    Shared Sub StopAndCloseSound(ByVal ID As Integer)
        StopSound(ID)
        CloseSound(ID)
    End Sub

    ''' <summary>
    ''' Set the playback time or position as a percentual value instead of milliseconds.
    ''' </summary>
    ''' <param name="ID">ID of the sound.</param>
    ''' <param name="percentPosition">Position in %.</param>
    ''' <param name="playing">Should the sound be playing afterwards?</param>
    ''' <param name="doloop">Should the sound loop until stopped?</param>
    ''' <remarks></remarks>
    Shared Sub SoundSetPercentPosition(ByVal ID As Integer, ByVal percentPosition As Single, ByVal playing As
Boolean, ByVal doloop As Boolean)

        Dim length As Integer = SoundStatusLength(ID)
        Dim position As Integer = CInt(length * percentPosition / 100)
        SoundSetPosition(ID, position, playing, doloop)

    End Sub

End Class
```

ErrorHandler

Filename: ErrorHandler.vb

```
Public Class ErrorHandler

    ''' <summary>
    ''' Show error message in a message box. Error: Invalid tag
```

```
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <param name="tagName">The name of the erroneus tag.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowTagError(ByVal rowCounter As Integer, fileNameAndDir As String, ByVal tagName As
String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Invalid tag " + tagName + ".", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: Wrong number of parameters.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowParamNumError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Wrong number of parameters.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The row in the file is started with the wrong sign.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowInitialSignError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Expected ' or [ at start of row.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The variable doesn't exist.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <param name="varName">Name of the variable.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowUnknownVarError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String,
ByVal varName As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Variable " + varName + " does not exist.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The value of the variable isn't numeric.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <param name="varValue">The value of the variable.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowNotNumericVarError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String,
ByVal varValue As String)
```

```
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Value " + varValue + " is not numeric.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The object doesn't exist.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <param name="objName">The name of the object.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowUnknownObjError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String,
ByVal objName As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Object " + objName + " does not exist.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The value of the timer isn't numeric.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowTimerNotNumError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Timer value is not numeric.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The timer value isn't positive.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowTimerNotPosError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"Timer value is not positive.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: There are more than one timer for the same action.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowMultipleTimerError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"There are more than a single timer. Only a single timer is supported.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The value is not boolean compatible.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
```

```
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowBoolError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf + "The value is not true or false.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The last non-comment sign of the row is erroneus.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowNoEndSignError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf + "Expected ] at the end of the tag.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The sound doesn't exist.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <param name="soundName">The name of the sound.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowSoundExistError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String, ByVal soundName As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf + "Sound " + soundName + " does not exist.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: There are more than a single click condition, making it impossible to trigger both at the same time.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowTaskMultipleClickError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf + "There are more than a single click condition, only a single click condition is supported in the Trigger and End sections.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: There are more than a single click condition, making it impossible to trigger both at the same time.
    ''' </summary>
    ''' <param name="rowCounter">What row in the file the error is made at.</param>
    ''' <param name="fileNameAndDir">The name of the file the error is made in.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowAlarmMultipleClickError(ByVal rowCounter As Integer, ByVal fileNameAndDir As String)
```

```
        MessageBox.Show("Error on row " + CStr(rowCounter + 1) + " in the file: " + fileNameAndDir + vbCrLf +
"There are more than a single click condition, only a single click condition is supported in the Handling section.",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub

    ''' <summary>
    ''' Show error message in a message box. Error: The variable with the given name doesn't exist.
    ''' </summary>
    ''' <param name="varName">The name of the missing variable.</param>
    ''' <remarks></remarks>
    Public Shared Sub ShowVarNotExist(ByVal varName As String)
        MessageBox.Show("Error in variable search: Variable with name " + varName + " could not be found.",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Sub


End Class
```