FOERST
Driving Simulators

Revision: 7.2 beta-R16012

Programming Tool

Version 7.2 beta

Reference

Chapter 1

Contents

1 Contents **3**

2 Document Conventions **13**

3 Scripting Language Overview **15**

3.1 Introduction . . . 15
3.1.1 Example Workflow . . . 15
3.1.2 Filesystem Locations . . . 17
3.2 Loading Scenarios . . . 18
3.2.1 Title . . . 18
3.2.2 Block . . . 19
3.2.3 Scenario . . . 19
3.2.4 Full Example . . . 19
3.2.5 Current Software Restrictions . . . 20
3.3 Structure of Scenario Files . . . 21
3.3.1 Header Directives . . . 21
3.3.1.1 Name (required) . . . 21
3.3.1.2 NameNo (deprecated) . . . 21
3.3.1.3 NameId (optional) . . . 21
3.3.1.4 Maxlevel (optional) . . . 22
3.3.1.5 Defaultlevel (optional) . . . 22
3.3.1.6 Levels (optional) . . . 22
3.3.1.7 LevelsNo (deprecated) . . . 22
3.3.1.8 LevelsId (optional) . . . 23
3.3.1.9 LevelsSub (optional) . . . 23
3.3.1.10 Encoding (optional) . . . 23
3.3.1.11 Full Example . . . 23
3.3.1.12 Deprecated Directives . . . 24
3.3.2 BEGIN and END . . . 24
3.3.3 The Checksum . . . 24
3.4 The Syntax Check Tool . . . 26
3.5 Configure Notepad++ . . . 26
3.5.1 Syntax Colouring . . . 27
3.5.2 Test Compilation . . . 27
3.6 The Debug Window . . . 27

3.6.1 The Debug Dialog . . . 28
3.7 Language Elements . . . 29
3.7.1 Variables . . . 29
3.7.1.1 Declarations . . . 29
3.7.1.2 Assignments . . . 29
3.7.2 Jumps . . . 31
3.7.3 Conditional Execution . . . 32
3.7.3.1 Vector Comparison . . . 33
3.7.3.2 Switch Blocks . . . 34
3.7.3.3 Loops . . . 34
3.7.4 Sleep . . . 35
3.7.5 Functions . . . 36
3.7.6 Procedures . . . 36
3.7.7 Debug Commands . . . 36
3.8 Examples . . . 38
3.8.1 Example Scenario 1 . . . 38
3.8.2 Example Scenario 2 . . . 40

4 Values and Variables **47**
4.1 Variables and Vectors . . . 47
4.1.1 Scalar Variables . . . 47
4.1.2 Vectors . . . 48
4.1.3 Timers . . . 50
4.1.4 Strings . . . 51
4.1.5 Global Variables . . . 51
4.2 Constants . . . 51
4.2.1 Units . . . 52
4.3 Assignments . . . 52
4.4 Calculations . . . 54

5 Functions **55**
5.1 Calling Functions . . . 55
5.1.1 Function Arguments . . . 55
5.1.2 Optional Arguments . . . 56
5.1.3 Return Values . . . 56
5.2 Function Reference . . . 57
5.2.1 Driver Functions . . . 57
5.2.1.1 Driver_GetAbsolutePosition . . . 58
5.2.1.2 Driver_GetAcceleration . . . 58
5.2.1.3 Driver_GetAcceleratorPedal . . . 58
5.2.1.4 Driver_GetAngles . . . 59
5.2.1.5 Driver_GetBlinker . . . 59
5.2.1.6 Driver_GetBrakePedal . . . 59
5.2.1.7 Driver_GetClutchPedal . . . 60
5.2.1.8 Driver_GetDistance . . . 60
5.2.1.9 *Driver_GetGear (deprecated)* . . . 60
5.2.1.10 Driver_GetGearShift . . . 60
5.2.1.11 Driver_GetHorn . . . 60
5.2.1.12 Driver_GetID . . . 60

5.2.1.13 Driver_GetIgnition . . . 61
5.2.1.14 Driver_GetInstrument . . . 61
5.2.1.15 Driver_GetLateralAcceleration . . . 61
5.2.1.16 Driver_GetLight . . . 61
5.2.1.17 Driver_GetRPM . . . 61
5.2.1.18 Driver_GetSlideAngle . . . 62
5.2.1.19 Driver_GetSpeed . . . 62
5.2.1.20 Driver_GetStarter . . . 62
5.2.1.21 Driver_GetSteeringWheel . . . 62
5.2.1.22 Driver_GetWiper . . . 62
5.2.1.23 Driver_SetAbs . . . 62
5.2.1.24 Driver_SetExtraSystem . . . 63
5.2.1.25 Driver_SetAbsoluteOrientation . . . 63
5.2.1.26 Driver_SetAbsolutePosition . . . 63
5.2.1.27 Driver_SetAcceleratorPedal . . . 64
5.2.1.28 Driver_SetBrake . . . 64
5.2.1.29 Driver_SetDynamicLateral . . . 64
5.2.1.30 Driver_SetDynamicLongitude . . . 65
5.2.1.31 Driver_SetGearShift . . . 65
5.2.1.32 Driver_SetInstrument . . . 65
5.2.1.33 Driver_SetMotorSound . . . 65
5.2.1.34 Driver_SetWorld . . . 66
5.2.1.35 Driver_SetPosition . . . 67
5.2.1.36 Driver_SetPositionDirect . . . 67
5.2.1.37 Driver_SetSpeed . . . 68
5.2.1.38 Driver_SetSteeringWheel . . . 68
5.2.1.39 Driver_SetCollision . . . 68
5.2.2 Error Functions . . . 69
5.2.2.1 Error_GetErrorCount . . . 69
5.2.2.2 Error_GetErrorMask . . . 69
5.2.2.3 Error_GetLastError . . . 70
5.2.2.4 Error_ResetErrors . . . 70
5.2.2.5 Error_SetError . . . 70
5.2.2.6 Error_SetErrorMode . . . 70
5.2.2.7 Error_SetTriggerValue . . . 71
5.2.3 Logging Functions . . . 72
5.2.3.1 Logfile_AddScriptVariable . . . 72
5.2.3.2 Logfile_Reset . . . 72
5.2.3.3 Logfile_SetRate . . . 73
5.2.4 String Functions . . . 74
5.2.4.1 String_GetLength . . . 74
5.2.4.2 String_Part . . . 74
5.2.4.3 String_Transform . . . 74
5.2.5 Surround Functions . . . 75
5.2.5.1 Surround_GetHeadway . . . 75
5.2.5.2 Surround_GetTimeToCollision . . . 75
5.2.5.3 Surround_SetEnvironmentSound . . . 75
5.2.5.4 Surround_GetTimeToLineCrossing . . . 75

5.2.5.5 Surround_GetDistanceToLineCrossing 76
5.2.5.6 Surround_GetRoadAhead 76
5.2.5.7 Surround_GetRoadSpeed 76
5.2.5.8 Surround_GetRoadType 76
5.2.5.9 Surround_SetExternOrientation 76
5.2.5.10 Surround_SetLight 77
5.2.5.11 Surround_SetLightDurations 77
5.2.6 Symbol Functions 78
5.2.6.1 Symbol_Activation 78
5.2.6.2 Symbol_SetBrightness 78
5.2.6.3 Symbol_SetPosition 78
5.2.6.4 Symbol_SetSize 79
5.2.6.5 Symbol_SetTransparency 79
5.2.7 System Functions 80
5.2.7.1 System_ClearText 80
5.2.7.2 System_EnhancedReality 80
5.2.7.3 System_EnhancedRealityParameter 80
5.2.7.4 System_GetControl 81
5.2.7.5 System_GetCountry 81
5.2.7.6 System_GetLanguage 81
5.2.7.7 System_GetRandom 81
5.2.7.8 System_GetSimulation 82
5.2.7.9 System_GetStatus 82
5.2.7.10 System_GetViewSize 82
5.2.7.11 System_InitSampleFile 82
5.2.7.12 System_Menu 83
5.2.7.13 System_Perspective 83
5.2.7.14 System_PlaySample 83
5.2.7.15 System_Reset 84
5.2.7.16 System_Scenario 84
5.2.7.17 System_GetSubVector 84
5.2.7.18 System_SetSubVector 84
5.2.7.19 System_CutSubVector 85
5.2.7.20 System_SetImage 86
5.2.7.21 System_SetImageFile 87
5.2.7.22 System_SetImageText 87
5.2.7.23 System_SetReport 88
5.2.7.24 System_GetReport 88
5.2.7.25 System_Screenshot 88
5.2.7.26 System_SetAlcoholValue 89
5.2.7.27 System_ToggleAlcoholValue 89
5.2.7.28 System_SetBlurEffect 89
5.2.7.29 System_SetCurrentPos 90
5.2.7.30 System_SetDisplayOptions 90
5.2.7.31 System_SetText 90
5.2.7.32 System_SetTopViewArea 91
5.2.7.33 System_SetTopViewMode 91
5.2.7.34 System_GetInfo 91

5.2.7.35 System_Math . . . 91
5.2.7.36 System_VectorMath . . . 92
5.2.7.37 System_DataFile . . . 92
5.2.7.38 System_GetUserData . . . 92
5.2.7.39 System_SetUserData . . . 92
5.2.7.40 System_GetUserVariable . . . 93
5.2.7.41 System_SetUserVariable . . . 93
5.2.7.42 System_ClearUserVariable . . . 93
5.2.7.43 System_Shutdown . . . 93
5.2.7.44 System_Start . . . 94
5.2.7.45 System_InitPhase . . . 94
5.2.7.46 System_Stop . . . 94
5.2.8 Traffic Functions . . . 95
5.2.8.1 Traffic_GetAbsoluteDistance . . . 95
5.2.8.2 Traffic_GetAbsolutePosition . . . 95
5.2.8.3 Traffic_GetAngles . . . 96
5.2.8.4 Traffic_GetDistAndVelocity . . . 96
5.2.8.5 Traffic_GetInfo . . . 96
5.2.8.6 Traffic_GetPosition . . . 97
5.2.8.7 Traffic_GetRouteStatus . . . 97
5.2.8.8 Traffic_InsertCarAtNextJunction . . . 97
5.2.8.9 Traffic_InsertVehicle . . . 98
5.2.8.10 Traffic_ObjectExists . . . 99
5.2.8.11 Traffic_Remove . . . 99
5.2.8.12 Traffic_ResetSigns . . . 100
5.2.8.13 Traffic_SetEvent . . . 101
5.2.8.14 Traffic_GetEventName . . . 101
5.2.8.15 Traffic_SetLights . . . 102
5.2.8.16 Traffic_SetAI . . . 102
5.2.8.17 Traffic_SetBehaviour . . . 102
5.2.8.18 Traffic_SetAbsolutePosition . . . 103
5.2.8.19 Traffic_SetProfile . . . 103
5.2.8.20 Traffic_SetRoute . . . 104
5.2.8.21 Traffic_SetSigns . . . 105
5.2.8.22 Traffic_SetSignProperty . . . 106
5.2.8.23 Traffic_SetSpeed . . . 107
5.2.8.24 Traffic_SetAcceleration . . . 108
5.2.8.25 Traffic_SetDistance . . . 108
5.2.8.26 Traffic_GetPhysicalParameter . . . 108
5.2.8.27 Traffic_SetPhysicalParameter . . . 108
5.2.8.28 Traffic_SetDoor . . . 109
5.2.8.29 Traffic_SetLeaveParking . . . 109
5.2.8.30 Traffic_SetPedestrian . . . 109
5.2.8.31 Traffic_SetSpeedLevel . . . 109
5.2.8.32 Traffic_SetTrackDeviation . . . 110
5.2.8.33 Traffic_IsOnInvisiblePlate . . . 110
5.2.9 World Functions . . . 111
5.2.9.1 World_GetWeather . . . 111

5.2.9.2 World_SetWeather . . . 111
5.2.9.3 World_SetWeatherParameter . . . 111
5.2.9.4 World_GetWeatherParameter . . . 111
5.2.9.5 World_SetWind . . . 112
5.2.9.6 World_SetRoadFriction . . . 112
5.2.9.7 World_SetFogSight . . . 112
5.2.9.8 World_GetFogSight . . . 112
5.2.9.9 World_GetRoadDistance . . . 113
5.2.9.10 World_GetAbsoluteDistance . . . 113
5.2.9.11 World_LookupRoadNumber . . . 114
5.2.9.12 World_LookupRoadTile . . . 114
5.2.9.13 World_GetRoadToPlateVector . . . 114
5.2.9.14 World_SetRoadMode . . . 114
5.2.9.15 World_SetNorth . . . 114
5.2.9.16 World_SetNewLap . . . 115
5.2.9.17 World_GetJunctionCenter . . . 115
5.2.9.18 World_GetWorldName . . . 115
5.2.9.19 World_FindWorldByName . . . 116
5.2.9.20 World_GetLanes . . . 116
5.2.9.21 World_EnumTiles . . . 116
5.2.9.22 World_EnumSignSchemes . . . 117
5.2.9.23 World_GetTileSignVariant . . . 117
5.2.9.24 World_IsRoadOnInvisiblePlate . . . 117
5.2.9.25 World_GetInvisibleSourcePosition . . . 118
5.2.9.26 World_GetRoadCount . . . 118
5.2.9.27 World_GetNextRoad . . . 119
5.2.9.28 World_IsRoadOnJunction . . . 119
5.2.9.29 World_GetRoadLength . . . 119
5.2.9.30 World_GetRoadCurvature . . . 120
5.2.9.31 World_LookupCreate . . . 120
5.2.9.32 World_LookupLoad . . . 120
5.2.9.33 World_LookupSave . . . 120
5.2.9.34 World_Lookup . . . 121
5.2.9.35 World_LookupClear . . . 121
5.2.9.36 World_LookupRecord . . . 121
5.2.9.37 World_LookupSimplify . . . 122

6 Complex Statements **123**
6.1 The STMT Directive . . . 123
6.2 Left Lists . . . 123
6.3 Vector Selectors . . . 124
6.4 Right Lists . . . 124

7 Advanced Topics **127**
7.1 Namespaces . . . 127
7.2 Libraries . . . 128
7.2.1 Library Guidelines . . . 128
7.3 World Editor . . . 130
7.4 Traffic Lights . . . 131

7.4.1 Levels of Functionality . . . 131
7.4.2 Advanced Positioning . . . 131
7.4.3 Traffic Light Programs . . . 132

8 The Network Protocol 135
8.1 Introduction . . . 135
8.2 Connection and Transfer Protocol . . . 136
8.3 Data Protocol . . . 137

A Predefined Constants 139
A.1 World Constants . . . 139
A.2 Language Constants . . . 140
A.3 Country Constants . . . 140
A.4 Object Constants . . . 141
A.5 Object Insertion Flags . . . 142
A.6 Information Request Constants . . . 142
A.6.1 Bit Values for EVTI_LIGHTSTATE . . . 143
A.6.2 Traffic Behaviour Phases for EVTI_PHASE . . . 143
A.7 Static Object Types . . . 144
A.7.1 Values for STATIC_WARNINGSIGN . . . 146
A.7.2 Values for STATIC_MANDATORYSIGN . . . 147
A.7.3 Values for STATIC_INFORMATIONALSIGN . . . 148
A.7.4 STATIC_SMALLINFORMATIONALSIGN Values . . . 148
A.7.5 Values for STATIC_TURNING . . . 148
A.7.6 Values for STATIC_STOP_LINE . . . 149
A.7.7 Values for STATIC_ROADMARK_ARROW . . . 149
A.7.8 Values for STATIC_CROSSWALK . . . 149
A.7.9 Values for STATIC_BILLBOARD . . . 149
A.7.10 Values for SIGNPROPERTY . . . 150
A.7.11 Values for STATIC_MOTORWAYSIGN . . . 150
A.7.12 Values for STATIC_COUNTDOWN_MARKER . . . 151
A.7.13 Values for STATIC_ROADTYPE . . . 151
A.8 Speed Setting Modes . . . 151
A.9 Track Setting Modes . . . 152
A.10 Enhanced Reality Constants . . . 152
A.10.1 Enhanced Reality Parameters . . . 153
A.10.2 Diagram Types . . . 154
A.11 Event Constants . . . 154
A.11.1 Event Triggering Modes . . . 155
A.11.2 Event Option Bits . . . 155
A.12 Weather Constants . . . 155
A.13 Weather Parameter Constants . . . 156
A.14 Road Surface Modes . . . 156
A.15 Road Distance Constants . . . 156
A.16 System Error Constants . . . 157
A.17 Traffic Error Constants . . . 158
A.18 Error Mode Constants . . . 163
A.19 Vehicle Light Constants . . . 163
A.20 Windscreen Wiper Constants . . . 164

A.21 Gear Shift Constants . . . 164
A.22 Instrument Constants . . . 165
A.22.1 Instument Value Constants . . . 166
A.23 Behaviour Profile Constants . . . 167
A.24 Behaviour Constants . . . 168
A.25 Artificial Intelligence Constants . . . 169
A.25.1 Animation Constants . . . 170
A.26 Sound Constants . . . 172
A.27 Traffic Light Constants . . . 173
A.28 End of Route Behaviour . . . 174
A.29 Numeric Constants . . . 174
A.30 Unit Conversion Constants . . . 174
A.31 User Data Item Constants . . . 175
A.32 Report-Menu Line Attribute Bits . . . 175
A.33 Report-Menu useage types . . . 176
A.34 Menu State Bits . . . 176
A.35 System States . . . 176
A.36 Camera Perspectives . . . 177
A.36.1 Top View Modes . . . 178
A.37 Physical Vehicle Parameters . . . 179
A.37.1 Motor Brake Modes . . . 180
A.38 Image Positioning Constants . . . 180
A.39 Text and Image Slot Constants . . . 181
A.40 System Information . . . 181
A.41 Mathematical Functions . . . 182
A.42 String Transformations . . . 183
A.43 File Operations . . . 183
A.44 Lane Types . . . 184
A.45 Extra Systems . . . 184

B World Maps 185
B.1 Circuit . . . 186
B.2 Motorway . . . 187
B.3 Trainer Map . . . 188
B.4 Mountain Course . . . 189
B.5 Small Town . . . 190
B.6 City . . . 191
B.7 Desert . . . 192
B.8 City 2 . . . 193
B.9 Racing Course . . . 199

C Keyboard Shortcuts for F12 Driving Simulators 201
C.1 Fast Driver Identification Input . . . 201
C.2 Global Name Input . . . 201
C.3 Keyboard Lock . . . 201
C.4 Keyboard Codes . . . 202
C.4.1 <Shift+Ctrl+Alt+...> . . . 202
C.4.2 <Plain Keys> . . . 202
C.4.3 <Shift+...> . . . 203

C.4.4 <Ctrl+...> 204
C.4.5 <Alt+...> 205

D Index **207**

E List of Figures **209**

F List of Tables **211**

G List of Listings **213**

Foerst GmbH
Industriegelände 5
D-51674 Wiehl
Telephone: +49 2262 72970-0
Telefax: +49 2262 72970-12
Email: info@fahrsimulatoren.eu
Internet: http://www.fahrsimulatoren.eu

© 1976-2017 Foerst GmbH. All rights reserved.

Information in this document is subject to change without notice. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), or for any purpose, without express written permission of Foerst GmbH.

Chapter 2

Document Conventions

The following document conventions are used througout this book:

Example	Description
SCENARIOS\CUSTOM.TXT	Capital letters indicate filenames and terms used at the command line.
Key Terms	Bold type indicates text that must be typed exactly as shown. This includes language keywords, directives, symbols and operators.
placeholders	Italics indicate variable information supplied by the user.
`Example Code`	Nonproportional text indicates example programs, user input and screen text output.
[*optional items*]	Square brackets indicate that the enclosed item is optional.
{ *choice1* | *choice2* }	Braces and a vertical bar indicate a choice between two or more items. You must choose one of the items unless square brackets surround the braces.
Repeating elements ...	Three dots following an item indicate that you may type more items having the same form.
⟨SHIFT+F1⟩	Small upper case letters surrounded by angle brackets indicate key names.

In order to avoid linguistic problems, we use a special convention concerning directions. With “right” we always mean the legal driving lane. This is the convention used in Central Europe and the U.S. Inhabitants of countries, where one drives on the left (e.g. U.K. and Commonwealth countries), should always translate “left” to “right” and vice versa when reading this book.

Chapter 3

Scripting Language Overview

3.1 Introduction

With the help of this documentation a user of the F12 driving simulator program can create his or her own traffic scenarios. Scenarios are programs written in a special scripting language and exert high-level control over the simulated environment.

A scenario usually selects the virtual world and the starting position for the driver. It places traffic signs, traffic lights and other static objects. It determines other vehicles' initial positions, where they want to go and how fast they want to be. It is quite common to change other vehicles' speed or display text or images as a reaction to the driver reaching a particular point in the virtual world. A scenario usually does not control each and every movement of a simulated vehicle, although tighter control is possible[1].

Scenarios are simple text files[2] and may be edited with almost any Windows™ text editor (e.g. notepad). Some simple custom scenarios are shipped with the simulator software and may be used as boilerplates for new scenarios.

In order to help you develop your own scenarios, a syntax checker and various debug methods are provided. The latter include a single step mode, debugging output and keyboard shortcuts to dump variable values.

3.1.1 Example Workflow

You can directly edit and test scenarios on a simulator computer, but usually a separate development installation is used. This section describes, how the different components of a more complex deployment work together. You do not need all the components mentioned below just for developing traffic scenarios. If you just want to write scenarios and do not care about custom worlds or distribution of your work to several software installations, you can skip this section.

For the illustrative purpose of this section, we assume the following set-up:

[1] Automatic behaviour can be disabled using **Traffic_SetAI**. Then, tight control with e.g. **Traffic_SetSpeed** and **Traffic_SetTrackDeviation** is possible.

[2] either plain ASCII or UTF-8 with BOM

- A control desk with World Editor, Control Desk software and two F12 installations:
  - The Control Desk's image generator installation at C:\PKW. This installation also determines the set of scenarios selectable in the Control Desk software. It is shared via network and locally reimported as the drive letter R:\
  - A development installation at C:\PKW-DEV.
- A simulator with its F12 installation at C:\PKW, which is shared via network and mapped to the drive letter P:\ on the Control Desk pc.
- Another simulator or an image generator with its F12 installation at C:\PKW, which is shared via network and mapped to the drive letter Q:\ on the Control Desk pc.

All four F12 installations are kept on the same software revision, i.e. updates are always performed on all of them. For all but the C:\PKW-DEV installation this is strictly necessary and having C:\PKW-DEV on the same software revision allows us to distribute our work by simply synchronizing some subfolders of the F12 installation folders[3].

All development happens inside the C:\PKW-DEV installation, where it does not interfere with the control desk and the simulator(s). The World Editor is configured to use C:\PKW-DEV as its Project folder (for details refer to the World Editor's seperate manual). A development cycle could look as follows:

1. You create a new world with the World Editor. Details on this step can be found in the World Editor's manual. You do not have to export your work, because it will already be saved in the C:\PKW-DEV folder.
2. You start up your F12 development installation, navigate the menu to the **Dynamic Worlds** program and test drive your new world. A short introduction to using the development installation can be found in section 3.6. Quite propably you will find some bad sight distances, in which case you go back to step (1).
3. You create a new scenario in C:\PKW-DEV\SCENARIOS\CUSTOM (either from scratch or by copying some other scenario) and add it to C:\PKW-DEV\SCENARIOS\CUSTOM\CUSTOM.TXT. Refer to sections 3.2 and 3.3 for an introduction and 7.3 for additional information on using self-made worlds.
4. When you are satisfied with your initial scenario version, you again start up your F12 development installation and test your scenario. You use debugging functionality to find faults in your scenario as described in section 3.7.7. Instead of going back to step (3), smaller errors can be corrected while the simulation is running and reloading afterwards.

[3] One could use a different software version, but then scenarios would have to be copied selectively and worlds would have to be distributed using the World Editor's export functions on all other installations in turn.

5. When your scenario is ready for production, you distribute your work to all the other F12 installations. This is most easily done by turning on all the relevant pcs and executing the batch file DISTRIBUTE.BAT found in the C:\PKW-DEV folder[4].

3.1.2 Filesystem Locations

All filenames used in the remainder of this document will be relative to the F12 program installation folder. As described before, there can be more than one installation on the same computer. In this case, the installation used for script development is meant. If you have installed the program to e.g.

C:\PKW-DEV,

the path

C:\PKW-DEV\SCENARIOS\CUSTOM\CUSTOM.TXT

will simply be referenced as

SCENARIOS\CUSTOM\CUSTOM.TXT.

Scenario files should be stored in subfolders of the SCENARIOS folder. In particular, custom scenarios should be stored in the SCENARIOS\CUSTOM folder or some subfolder thereof. In order to be compatible with future extensions, you are strongly encouraged to use the file extension *.SCE.

[4] You need to synchronize the F12 installation subfolders SCENARIOS, WORLDS and OGFXWC. The provided batch file DISTRIBUTE.BAT must be adjusted to your situation before first use. In our example, you would have to change the header of the batch file to: **set TARGETDRIVES=p q r**

3.2 Loading Scenarios

After having created new scenario files, they must be made known to the system by inserting some directives into the file SCENARIOS\CUSTOM\CUSTOM.TXT. Only then will the program be able to load a scenario and offer it to the user in the menu window.

Custom scenarios are logically organized as programs, blocks and scenarios. Each scenario program consists of one or more blocks, which in turn consist of one or more scenarios. While scenario files deal with the single scenarios, their logical organization is specified in CUSTOM.TXT.

Three directives are used: **Title**, **Block** and **Scenario**. Use a colon character to separate directive and value. Each directive is to be placed on a line of its own.

3.2.1 Title

Using this directive you specify the title of a scenario program as it will be displayed to the user in the menu window. You may specify different titles for different languages by appending a language number to the directive. The available language constants are listed in table 3.1. More about language constants may be found in the reference part under **System_GetLanguage** (section 5.2.7.6).

Table 3.1: Language Constants

Language	Code
Danish	4
Dutch (Belgium)	9
Dutch (Netherlands)	3
English	2
French	8
German	1
Greek	7
Spanish	6
Swedish	5

Listing 3.1: Title Syntax

```
Title: Training Program
Title1: Trainingsprogramm
Title8: Programme de pratique
```

In the German program version the program title “Trainingsprogramm” is displayed in the menu window. In the french version “Programme de pratique” is displayed. For all other languages, “Training Program” is displayed.

3.2.2 Block

Each scenario program consists of 1 - 8 blocks. As blocks are also displayed in the menu window, their display names must be specified. They can be localized for different languages similarly to the title command.

Listing 3.2: Block Syntax

```
Block: City traffic
Block1: Stadtverkehr
Block8: Circulation en ville
```

In the German version the block name “Stadtverkehr” is displayed, “Circulation en ville” in the French version and “City traffic” for all other language versions.

3.2.3 Scenario

Following each **Block** directive, the scenarios making up the scenario block are specified. Their display names (along with other properties) are specified in the .SCE files. Hence, one only has to specify the filesystem path(s) of the scenario file(s). The syntax is

Listing 3.3: Scenario Syntax

```
Scenario: filename
```

3.2.4 Full Example

An example of how the file CUSTOM.TXT might look like is given in listing 3.4 below.

Listing 3.4: Example CUSTOM.TXT

```
Title: Training program
Title1: Trainingsprogramm
Title8: Programme de pratique

Block: City traffic
Block1: Stadtverkehr
Block8: Circulation en ville

Scenario: ./scenarios/custom/wayfindingnav.sce
Scenario: ./scenarios/custom/wayfindingmap.sce
Scenario: ./scenarios/custom/smallville.sce

Block: Motorway
Block1: Autobahn
Block8: Autoroute

Scenario: ./scenarios/custom/merging.sce
```

```
Scenario: ./scenarios/custom/wrongway.sce
Scenario: ./scenarios/custom/construction.sce
```

In this example, the single program “Training program” is defined, which consists of the two blocks “City traffic” and “Motorway”. The first block includes the scenarios defined in WAYFINDINGNAV.SCE, WAYFINDINGMAP.SCE and SMALLVILLE.SCE. The second block includes MERGING.SCE, WRONGWAY.SCE and CONSTRUCTION.SCE.

3.2.5 Current Software Restrictions

Before version 4 of the simulator program, the file CUSTOM.TXT was only allowed to define a single program with up to four scenario blocks. The number of scenarios in each block was unlimited, though. Now both blocks and scenarios are unlimited.

3.3 Structure of Scenario Files

As stated before, the .SCE files defining the scenarios are simple ASCII files. A complete scenario file consists of three parts: Header directives specifying properties of the scenario, the actual source code surrounded by **BEGIN** and **END** directives and a checksum. You can insert comments into your .SCE files. Starting with two slashes // the compiler ignores all text up to the end of the current line.

3.3.1 Header Directives

A scenario file always begins with header directives, the syntax of which is very similar to that found in CUSTOM.TXT.

3.3.1.1 Name (required)

Using this directive, you specify the display name of your scenario (c.f. **Title** and **Block** directives in CUSTOM.TXT). Localization of the display name is again done by appending a language identifier.

Listing 3.5: Name Syntax

```
Name: Right of way
Name1: Vorfahrt
Name8: Priorité
```

3.3.1.2 NameNo (deprecated)

Using this directive, you specify the display name of your scenario (c.f. **Title** and **Block** directives in CUSTOM.TXT). The name is given by reference into the file "pkw*xxx*.txt". Localization of the name is automatically done by using the correct language file xxx. If this entry exists, it will override locally supplied names.

Listing 3.6: NameNo Syntax

```
NameNo: 368
```

3.3.1.3 NameId (optional)

Using this directive, you specify the display name of your scenario (c.f. **Title** and **Block** directives in CUSTOM.TXT). If the name has an entry in "pkw000.txt", it is automatically translated by lookup in "pkw*xxx*.txt", where xxx is the current language. If this entry exists, it will override locally supplied names.

Listing 3.7: NameId Syntax

```
NameId: "Right of way"
```

3.3.1.4 Maxlevel (optional)

Listing 3.8: Maxlevel Syntax

```
Maxlevel[[entry]]: maxlevel
```

You can create scenarios with adjustable difficulty level. If you choose to do so, you must specify the number of levels using the **Maxlevel** directive. The user may then choose between levels 1 . . . *maxlevel* in the simulator menu. From the programmer's point of view, the user's selection is reflected in a global variable **level**, which you may query from your source code. For more about global variables, refer to section 4.1.5.

3.3.1.5 Defaultlevel (optional)

Listing 3.9: Defaultlevel Syntax

```
Defaultlevel[[entry]]: defaultlevel
```

If you have specified multiple levels using **Maxlevel**, you can specify the default level. This level will be initially selected in the menu window.

3.3.1.6 Levels (optional)

Listing 3.10: Levels Syntax

```
Levels[l][[entry]]: "name_1","name_2", ..., "name_n"
```

with n = Maxlevel[entry], l = language number.

When you create a scenario with menu-selectable options, this command allows to replace the simple numbers, shown by default, by text strings.

3.3.1.7 LevelsNo (deprecated)

Listing 3.11: LevelsNo Syntax

```
LevelsNo[[entry]]: nref_1 , nref_2 ,..., nref_n
```

with n = Maxlevel[entry]

When you create a scenario with menu-selectable options, this command allows to replace the simple numbers, shown by default, by text strings from the language specific text reference. This reference corresponds to the references given in the file "pkw*xxx*.txt" with xxx replaced by the active language number.

3.3.1.8 LevelsId (optional)

Listing 3.12: LevelsId Syntax

```
LevelsId[[entry]]: "name_1","name_2", ..., "name_n"
```

with n = Maxlevel[entry]

When you create a scenario with menu-selectable options, this command specifies text strings, which are automatically translated to the active language. This is done by lookup in the file "pkw000.txt" and then replacing with corresponding entries in "pkw*xxx*.txt" with xxx replaced by the active language number.

3.3.1.9 LevelsSub (optional)

Listing 3.13: LevelsSub Syntax

```
LevelsSub[[entry]]: sub_1 , sub_2 ,..., sub_n
```

with n = Maxlevel[entry]

When you create a scenario with menu-selectable options, this command allows to control the number and structure of the shown level lines and choices. Each sub_*i* represents the entry number of the entry to be shown in the next line. The shown entries can thus be influenced by the current choice of the user. To select no further entry line, choose *0*.

3.3.1.10 Encoding (optional)

Using this directive, you specify the encoding of strings embedded in the source code. Allowed are many of the standard encoding names like **windows-1252** (western encoding), **iso-8859-1**, **iso-8859-15** or **utf-8**. The default is **windows-1252** or **utf-8**, if the file starts with a BOM.

Listing 3.14: Encoding Syntax

```
ENCODING: utf-8
```

3.3.1.11 Full Example

An example of how the the header of a scenario file might look like is given in listing 3.15 below.

Listing 3.15: Example Header

```
ENCODING: utf-8

Maxlevel[0]: 2
Maxlevel[1]: 2
```

```
Maxlevel[2]: 2

Defaultlevel[0]: 1
Defaultlevel[1]: 1
Defaultlevel[2]: 1

Levels[0]: "Semitrailer truck", "Tanker truck"

Levels[1]: "without trailer", "with trailer"

Levels[2]: "Weather: Nice", "Weather: Rain"

LevelsSub[0]: 2, 1
LevelsSub[1]: 2, 2
LevelsSub[2]: 0, 0
```

In this example, you can choose the vehicle type at first. If you choose "Tanker truck", you can switch between driving with our without a trailer, else this entry will be skipped. Finally you can choose the weather conditions.

There are three level layers in this example. As specified by **Maxlevel[]**, all three have two levels. The default level in each layer is 1 (**Defaultlevel[]**). When selecting the second level in layer zero, the next line will be the one defined in layer one, else the next line will be the one defined in layer two (**LevelsSub[]**). When defining **LevelsSub[]** = 0, there will be no more lines.

3.3.1.12 *Deprecated Directives*

For backwards compatibility reasons, the compiler also accepts the directives **Block** and **Number**. As the logical structure is now controlled by CUSTOM.TXT, these directives are simply ignored. Do not use them when writing new scenarios.

3.3.2 BEGIN and END

After the header directives, you write the actual source code. To separate the source code from the rest of the file, use the directives **BEGIN** and **END**.

Listing 3.16: BEGIN and END Syntax

```
BEGIN

source code

END
```

3.3.3 The Checksum

If you open an existing scenario file, you will finds lines similar to

Listing 3.17: Checksum

```
[CHECKSUM]
CHECKSUM=xxxxxx
VERSION=...
```

at the end. The main purpose of these lines is to protect your intellectual property as a scenario programmer and the integrity of scenario files written by you. You do not specify the checksum yourself. Rather, the checksum is generated by the scenario compiler upon successful parsing of your scenario file.

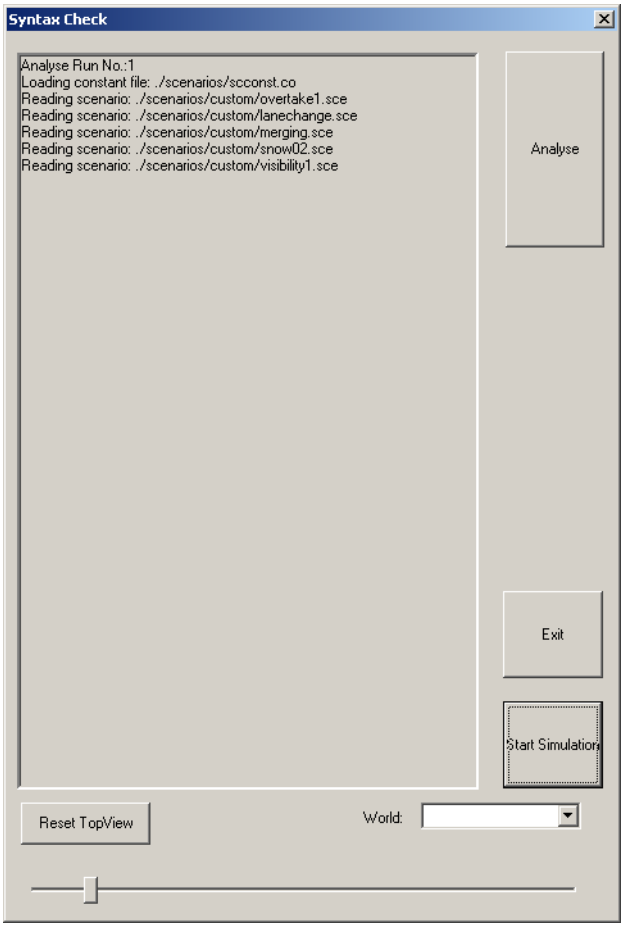
For you as a developer (having purchased a licence for the development environment), the compiler will add a valid checksum to the end of the file or modify it to reflect changes you have made to the scenario. Without such a licence, the compiler will only accept scenario files with correct checksum appended. This means, that users without a developer licence can load and use scenarios written by others, but can neither create nor modify them.

The **VERSION** line is only useful, if you put your scenario scripts under subversion version control, in which case the version control system puts useful information into this line. It is otherwise ignored.

3.4 The Syntax Check Tool

If a scenario file contains syntax errors, the simulator program silently refuses to load it. In order to locate problems with your source code, you can activate a normally hidden dialog. This dialog not only tells you about problems found in your source code, but also allows you to reparse the .SCE files.

Figure 3.1: Syntax Check Tool



Thus, you can (repeatedly) edit your source code with the text editor of your choice and then press the "Analyse" button to check your code again. The simulator program is only run after you press the "Start Simulation" button. Of course, any problems in an .SCE file still persisting at this time will prevent the corresponding scenario to appear in the simulator's menu. At any time, you can exit the program without starting the simulator by pressing the "Exit" button.

The Syntax Check Tool is activated by setting **AnalyseTool = An** in SIMUCONF.CFG.

3.5 Configure Notepad++

Although you can use almost any text editor to edit scenario script files, we recommend using the free and open source editor Notepad++, which is freely available from HTTPS://NOTEPAD-PLUS-PLUS.ORG. For the below to work, you should install its Explorer and NppExec plugins.

3.5.1 Syntax Colouring

Notepad++ allows to define custom languages for syntax colouring. You can use our prepared setting by importing the language file found in the installation archive at INSTALL\A08-NOTEPADPLUS\NPP-FOERSTSCRIPT.XML. In the Notepad++ menu, select **Language/Define your language...**. Press the **Import...** button and navigate to the above XML file.

3.5.2 Test Compilation

In the menu, navigate to **Plugins/NppExec/Advanced Options...** In the command window, enter

Listing 3.18: Command(s)

```
NPP_SAVEALL
C:\Pkw\SceCC.exe -a -D -t "$(FULL_CURRENT_PATH)"
```

Substitute your development F12 installation path for C:\PKW. Save this script as `compileSCE`. Press the cancel button and navigate the menu to **Plugins/NppExec/Execute...** Here you can define a menu item named e.g. `Compile SCE` associated with the script `compileSCE`. After having restarted Notepad++, you can finally assign a keyboard shortcut like e.g. ⟨CTRL+F7⟩ to the command using **Settings/Shortcut Mapper...** Under **Plugin commands**, locate `Compile SCE` and assign ⟨CTRL+F7⟩ as the shortcut.

You can now test compile a scenario script by activating the corresponding editor tab and pressing your keyboard shortcut.

3.6 The Debug Window

Normally, the end user starts the simulator in full screen mode. You as a scenario developer will usually use windowed mode, in which additional controls and informational text is displayed. Open the configuration file SIMUCONF.CFG. In order to deactivate full screen mode, the parameter **GfxMovable** must be set to **On**. When starting in windowed mode, the simulator window will look similar to figure 3.2. The view on the right hand side will be initially invisible and only appear after the rendering engine has been initialized.

In the subwindow at the lower left corner, status variables are displayed. Above the status window, you find the debug window. Here, various error messages, warnings, notifications and debug output are displayed. If you did not resolve any problems with scenario files, this is also the place where you will again be informed about the problem (faulty scenarios do not appear in menus displayed on the right hand side).

Particularly during program startup, output in this window is generated so fast that you may fail to read all of it. In this case or simply for later analysis, you may find all output shown in the debug window during program startup in the log file INITLOGFILE.LOG, which is recreated on each program startup.

You can drive using the keyboard. The most important keys are the arrow keys, numpad-comma and the function keys ⟨F1⟩ - ⟨F7⟩. Some keys only work after

Figure 3.2: Simulator Window

having unlocked the keyboard using ⟨SHIFT+F12⟩. The most common keyboard shortcuts for debugging purposes can be found in section 3.7.7. More details and key codes can be found in appendix C.

3.6.1 The Debug Dialog

As an alternative to the Debug Window, you might prefer another configuration, e.g. a fullscreen view or the demonstration layout, which shows many different views at once. You can select these views by adjusting the sight system using the Configure tool. When you do so, you lose the debug controls on the left hand side of the movable Debug Window. They are replaced by the Debug Dialog, which is activated with ⟨CTRL+F2⟩. Note, that you can access some very advanced undocumented debugging features using this dialog, which have the potential to make the simulation program completely unusable. Normally, you will only need the **Output** pane. While the simulation is running, new debug output is continuously added to the end of the debug output control. The control is only scrolled to make it visible, if the cursor is placed at the end of the control, which can be achieved by pressing ⟨CTRL+END⟩ while the control has the keyboard focus. The debug output control is cleared by pressing ⟨CTRL+DELETE⟩.

3.7 Language Elements

A scenario source code consists of a sequence of directives, each of which must occur on a line of its own. In the following subsections, the different kinds of directives are briefly introduced. In-depth discussions of the commands are found in later chapters of this book.

3.7.1 Variables

3.7.1.1 Declarations

The scripting language knows three kinds of variables: Scalar variables, vectors and timers. Before a variable can be used, it must have been declared.

Listing 3.19: Variable Declaration Syntax

```
VAR name [[=] initial_value]
VEC name [[=] initial_1 [, ..., initial_n]]
TIMER name [[=] initial_value]
STRING name [[=] "initial_value"]
```

Variables declared with **VAR** hold a single scalar (float or integer) value. **TIMER** variables behave almost identical to scalar **VAR** variables, but are automatically incremented during execution of the scenario. They represent time in seconds with a precision of one millisecond. **STRING** variables hold strings like messages or paths.

A **VEC** variable is a vector and is defined to be an infinite series of scalar variables. Starting from some index, all components of a vector are set to zero. You usually access the individual components of a vector by appending suffices to the variable name[5].

According to different purposes, for which vector variables might be used, there are different sets of suffices which you might use to access the components. The different naming schemes are interchangeable, but for code readability reasons you are urged to choose a naming scheme that fits your use of a vector. The naming schemes may be found in table 3.2, where '#' denotes the component index starting with 0.

When used as an argument of a function (c.f. section 3.7.5), the vector's components replace all following scalar variables of the function. As a vector has infinite length, you can only use vectors as last argument of a function. For a more detailed description of vectors and their meaning refer to chapter 4.

3.7.1.2 Assignments

Variables are created with either a specified initial value or are set to zero. Variables can obtain new values either as the result of a function call (c.f. section 3.7.5) or by explicit assignment. There are four forms of assignment:

[5] You can also access the individual components by variable index using the **LETVEC** directive.

Table 3.2: Vector Component Naming Schemes

#	Road Position	Cartesian World Position	Unspecified
0	**.road**	**.x**	**.v1**
1	**.dist**	**.y**	**.v2**
2	**.angle**	**.z**	**.v3**
3	**.track**	**.yaw**	**.v4**
4	**.world**	**.pitch**	**.v5**
5	**.offx**	**.roll**	**.v6**
6	**.offy**		**.v7**
7	**.offz**		**.v8**
8	**.offyaw**		**.v9**
9	**.offpitch**		**.v10**
10	**.offroll**		**.v11**
11			**.v12**
⋮			⋮

1. **LET**, which just copies values,
2. **CALC**, which assigns a variable the result of an operation involving two variables
3. **LETVEC**, which either extracts or sets a vector element by index, and
4. **PACKVEC**, which either packs multiple vectors into a single one or unpacks them again.

Listing 3.20: Assignment Syntax

```
LET destination [=] source
CALC destination = source1 operator source2
LETVEC destination = vector index
LETVEC vector index = source
PACKVEC vectorp = vectoru1, ..., vectorun
PACKVEC vectoru1, ..., vectorun = vectorp
```

For the **LET** command, *destination* may be a vector, vector component, (scalar or string) variable or timer. *source* may be any of the above or a constant value. Not all combinations of *destination* and *source* type are valid, though. For details refer to chapter 4.

For the **CALC** command, *destination*, *source1* and *source2* must be scalars – either variables, timers or constants. *operator* may be any of the common binary operations listed in table 3.3.

For the first form of **LETVEC**, *destination* must be a scalar variable, *vector* must be a vector variable and *index* must be either a nonnegative integer constant or

Table 3.3: Operations for the CALC Command

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	division remainder (modulo)
&	bitwise and[a]
|	bitwise or
^	bitwise exclusive or
<<	bit shift left
>>	bit shift right[b]

[a] For bitwise operations, the values are first rounded to 32bit integers.
[b] Do not rely on the highest bit behaviour (arithmetic or logical).

a scalar variable. *index* is zero-based, i.e. 0 specifies the first vector component, 1 the second...

For the second form of **LETVEC**, *vector* is again a vector variable and *index* is the zero-based index (scalar variable or integer constant). *source* is the scalar value to be assigned to the vector component. *source* must be a scalar variable or costant.

For both forms of **PACKVEC**, *vectorp* and all the *vectorui* must be vectors. The first form packs the vectors *vectorui* into *vectorp*, while the second form unpacks *vectorp*. This works by interleaving the elements of the single vectors. If the number of vectors *vectorui* is the same, the two forms are the inverse of each other.

Instead of **LET**, **CALC** and **LETVEC**, a simplified syntax can (and should) be used:

Listing 3.21: Simplified Assignment Syntax

```
destination = source
destination = source1 operator source2
destination = vector[index]
vector[index] = source
```

3.7.2 Jumps

You can control the flow of execution using the **GOTO**, **GOSUB** and **RETURN** commands. The targets for **GOTO** and **GOSUB** are defined using the **LABEL** directive.

Listing 3.22: Jump Syntax

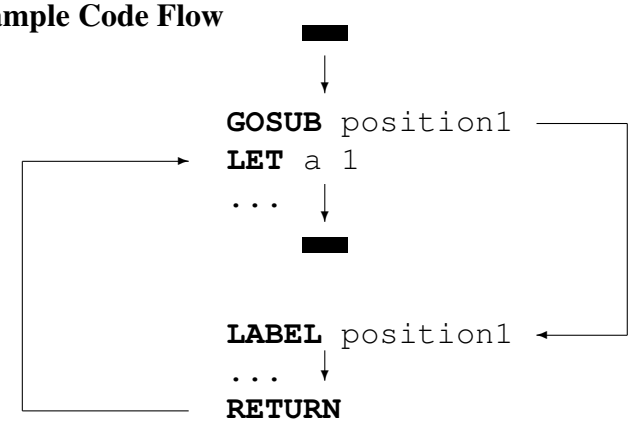
```
LABEL labelname
GOTO labelname
```

```
GOSUB labelname
RETURN
```

The **GOTO** command is used to perform unconditional direct jumps, while the **GOSUB** command is used to call subroutines. After having executed a **GOSUB** command, the next **RETURN** command moves the point of execution back to the line immediately following the **GOSUB** command.

GOSUB subroutine calls may be nested, but the nesting depth is restricted to 10. Exceeding this bound may produce unexpected results.

Figure 3.3: Example Code Flow



3.7.3 Conditional Execution

You can make the execution of parts of your code depend on conditions using the **IF**, **ELSIF**, **ELSE** and **ENDIF** directives. The syntax is:

Listing 3.23: IF/ELSE/ENDIF Syntax

```
IF condition1
    code1
[
ELSIF condition2
    code2
][
...
][
ELSE
    code3
]
ENDIF
```

code1, *code2* and *code3* each are sequences of one or more commands. *code1* is executed if *condition1* is true. The optional *code2* is executed if *condition1* is false and *condition2* is true. The optional *code3* is executed, if none of the preceding conditions are true. The syntax for a condition is

Listing 3.24: Condition Syntax

```
left1 op1 right1 [(AND|OR|XOR) left2 op2 right2][...]
```

op1 and *op2* are operators as listed in table 3.4. One of the left or right operands must be a variable, while the other one can be a variable or a constant. Multiple comparisons combined with **AND**, **OR** or **XOR** are always evaluated from left to right (there is no operator precedence). Multiple levels of conditional execution may be nested. More complex conditions can be built using parantheses.

Table 3.4: Operators for Comparisons

Operator	Meaning
=	equal to
<>	not equal to
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal

The **IF** and **ELSIF** directives also have versions with a **ONCE** suffix. Each such directive carries a hidden variable. If the corresponding code branch has been executed, the hidden variable remembers this and will never choose the code path again – no matter what the condition evaluates to. There is an additional directive **RESETONCE**, which resets all the hidden variables thereby reopening the corresponding code paths again. The syntax for these commands is

Listing 3.25: Extended Condition Syntax

```
IFONCE condition
ELSIFONCE condition
RESETONCE
```

3.7.3.1 Vector Comparison

The comparison of variables is not restricted to scalar variables, but you may also compare a vector variable with another vector variable. If you do so, the vectors are interpreted as road positions (c.f. section 4.1.2) and only the **.road**, **.dist** and **.angle** components are taken into account.

From the **.angle** part only directional information is extracted. In plain terms, a vector is considered bigger than another one, if it points into the same direction and is (with this direction in mind) “behind” the other one.

Of course, roads are only locally lines – if the positions are too far apart, many of the properties of the comparison operators (e.g. transitivity) are lost. If two road positions are far apart, all comparison operations except for inequality evaluate to false.

3.7.3.2 Switch Blocks

If you want to branch to different code paths corresponding to predetermined constant values of a variable, you may use the **SWITCH** directive. The **SWITCH** directive is used together with the **CASE**, **DEFAULT** and **ENDSWITCH** directives. Let $variable$ be a scalar variable, $constant1$, $constant2$ and $constant3$ scalar constants. Then the syntax

Listing 3.26: SWITCH/CASE/ENDSWITCH Syntax

```
SWITCH variable
    CASE constant1 [, constant3 [, ...]]
        code1
[
    CASE constant2
        code2
][
...
][
    DEFAULT
        code3
]
ENDSWITCH
```

is functionally equivalent to

Listing 3.27: SWITCH/CASE/ENDSWITCH Replacement

```
IF variable = constant1 [ \
        OR variable = constant3 [  OR ... ]]
    code1
[
ELSIF variable = constant2
    code2
][
...
][
ELSE
    code3
]
ENDIF
```

The difference is readability and the higher execution speed of the **SWITCH** block.

3.7.3.3 Loops

Using the directives **LABEL**, **GOTO** and **IF**, you can write all the usual loop constructs.

Just in order to make your code more readable, you can alternatively use the **WHILE** and **ENDWHILE** directives. The syntax

Listing 3.28: WHILE/ENDWHILE Syntax

```
WHILE condition
    code
ENDWHILE
```

is equivalent to

Listing 3.29: WHILE/ENDWHILE Replacement

```
LABEL somelabel
IF condition
    code
    GOTO somelabel
ENDIF
```

You may also use the **FOR** and **NEXT** directives. Let $variable$ be a scalar variable, $start$ and $stop$ scalar variables or constants and $step$ a nonzero numeric constant (by default, $step = 1$). The syntax

Listing 3.30: FOR/NEXT Syntax

```
FOR variable = start TO stop [ STEP step ]
    code
NEXT
```

is equivalent to

Listing 3.31: FOR/NEXT Replacement

```
LET variable = start
LABEL somelabel
IF variable OP stop
    code
    CALC variable = variable + step
    GOTO somelabel
ENDIF
```

where OP is "<=" if $step > 0$ and "<=" if $step < 0$.

3.7.4 Sleep

You can delay the execution of your script using the **SLEEP** directive.

Listing 3.32: Sleep Syntax

```
SLEEP time_in_seconds
```

After setting simulator variables, it is necessary to let the simulator process propagate the changes to its internal state before variables dependent on the changed property can be reliably queried. You can use a **SLEEP 0** command to force the simulator to update its internal state.

As a general rule of thumb, the more often you call **SLEEP**, the more responsive the simulation is. For this reason, a **SLEEP 0** instruction is forced after an unspecified number of source code instructions[6] if no explicit **SLEEP** command has been encountered until then.

It is a good idea to call **SLEEP 0** as often as possible. The only exception is when you have to do many time critical calculations at once.

3.7.5 Functions

Most interactions with the actual simulator program are done with the help of functions. There are functions to get and set environmental parameters of the simulation, trigger events, get and set positions of the own and other vehicles and many things more. The general syntax for function calls is:

Listing 3.33: Function Call Syntax

```
[FUNC] [retval [,retval2,...] =] function(parameters)
```

The available functions are discussed in chapter 5.

3.7.6 Procedures

Similar to calling functions, you can write your own subroutines and call them similar to builtin functions. This is done using the keywords **PROC** and **CALL**. **PROC** is similar to **LABEL** and **CALL** is similar to **FUNC**. In particular, **CALL** may be omitted. You read the procedure's arguments from the **argv** vector and write the result to the **retv** vector.

Listing 3.34: Example Procedure

```
PROC AddAndMultiply
    retv[0] = argv[0] + argv[1]
    retv[1] = argv[0] * argv[1]
    RETURN

...

VAR a
VAR b
CALL a,b = AddAndMultiply(2,3)
// now, a = 5 and b = 6

...
```

3.7.7 Debug Commands

In this section we assume that the simulator has been started in windowed mode (c.f. section 3.5). The first possibility to locate bugs in your code is to send text

[6] Unspecified, but ≥ 50.

messages to the debug window[7]. This can be accomplished using the following directives:

Listing 3.35: Debug Command Syntax

```
DEBUGTEXT "text"
DEBUGVAR variable
DEBUGNEWLINE
```

DEBUGTEXT queues the specified text for printing to the debug window.
DEBUGVAR queues the contents of a variable for printing.
DEBUGNEWLINE prints the queued text and variable and a newline character to the debug window.

Further possibilities for debugging are given by one more command

Listing 3.36: Function Call Syntax

```
BREAKPOINT
```

and several keyboard commands you can issue at runtime.

Like the name suggests, the **BREAKPOINT** command introduces a possible breakpoint into your code. When ordered to do so (using keyboard codes) the program will halt at points marked by **BREAKPOINT** commands.

Normally, the keyboard operates in restricted mode. In order to issue debugging commands, the restrictions first have to be unlocked. This is done by pressing ⟨SHIFT+F12⟩. A list with the most important debug commands is presented in table 3.5.

Table 3.5: Debug Key Codes

Key	Command
⟨S⟩	"Step": Single step mode. The program is halted after every instruction until ⟨S⟩ is pressed again or another command is issued.
⟨V⟩	Show the contents of all scalar user variables.
⟨P⟩	Show the contents of all vector variables.
⟨I⟩	Show the contents of read-only scalar variables (currently only **level**).
⟨B⟩	"Breakpoint Mode": Run until the next **BREAKPOINT** command is encountered.
⟨G⟩	"Go": Resume normal operation.
⟨R⟩	"Reload": Stop the currently running scenario and reload it. Does not respect changes in the menu structure. The scenario is only reloaded, if it does not contain syntax errors — watch the debug output.

[7] Among C and C++ programmers, this is also known as the "printf-approach" to debugging.

3.8 Examples

In this section two example programs are presented and discussed in order to illustrate the concepts explained in the preceding sections. You will also see some of the more common functions at work. You are encouraged to look these functions up in the function reference chapter 5 to both discover the full potential of these functions and explore additional possibilities to influence the simulated world.

3.8.1 Example Scenario 1

A very simple and easily comprehensible example scenario is contained in SCENARIOS\CUSTOM\EXAMPLE01.SCE.

Listing 3.37: SCENARIOS\CUSTOM\EXAMPLE01.SCE

```
Name: Example 1
Name1: Beispiel 1

BEGIN

LABEL Start
    VAR minspeed = 10 // minimum speed before scenario starts in km/h
    VAR myobjid = 0

    VEC startpos = 7, 100, 180
    VEC endpos = 5, 40, 0

    Driver_SetWorld (WORLD_SMALLTOWN)   // course in a town
    Driver_SetPositionDirect (startpos) // start position
    SLEEP 0
    Traffic_ResetSigns (2)

    World_SetWeather (WEATHER_FOG)                      // foggy weather
    World_SetFogSight (0, 350)                          // sight distance
                                                        //     in meters
    myobjid = Driver_GetID (0)                          // object id of
                                                        //     own car
    Traffic_SetRoute (myobjid, ENDROUTE_RANDOM, endpos) // set route
    Surround_SetLight (TLST_RED)

LABEL prepare
    VAR v = 0
    v = Driver_GetSpeed ()
    IF v > minspeed
        GOTO go
    ENDIF
    SLEEP 0
    GOTO prepare

LABEL go

LABEL mainloop
    IFONCE mypos > endpos
        GOTO ends
    ENDIF
    GOTO mainloop

LABEL ends
    System_Stop ()

END
[CHECKSUM]
CHECKSUM=104028
VERSION=$Rev: 15803 $ $Date: 2017-06-29 11:59:32 +0200 (Do, 29 Jun 2017) $ $URL: svn+ssh://svnflinu
```

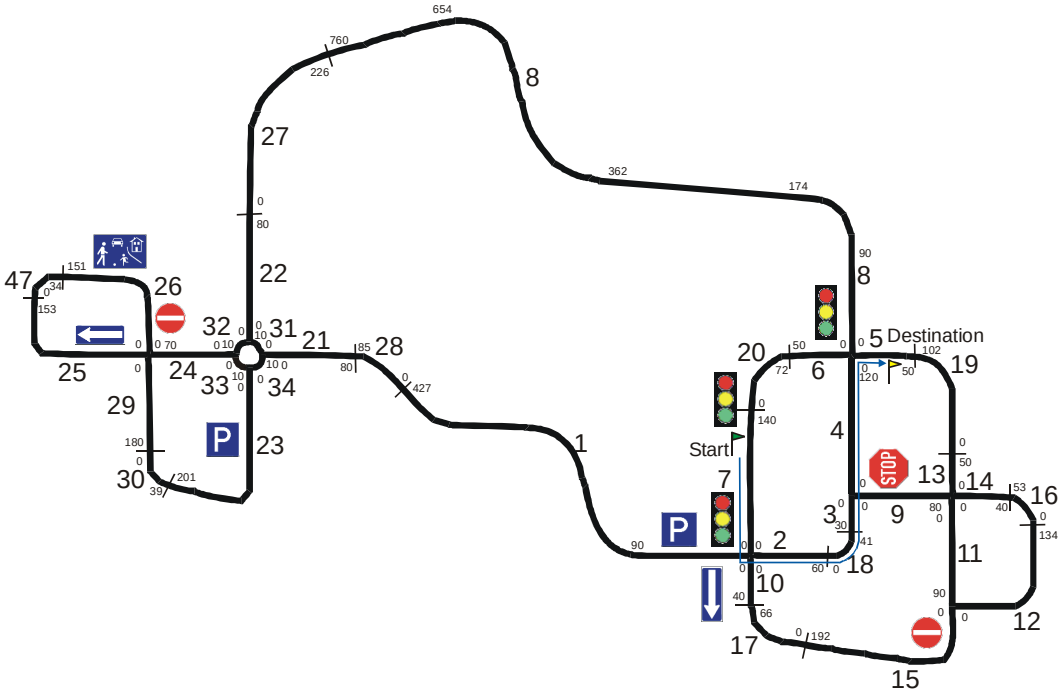
The header directives **Name** and **Name1** define the display name of the scenario for the simulator menu system. The scenario is named “Beispiel 1” in the German program version and “Example 1” in all other language versions.

The scenario source code first selects the map **WORLD_SMALLTOWN** using the function **Driver_SetWorld**. The function **Driver_SetPositionDirect** then positions the driver in the selected map. In order to be able to easily change this start position, it is defined in the variable *startpos*. In the example, it is defined to be road number 7, distance 100m from its origin. The angle of 180 degrees means, that the driver is positioned heading in the direction opposite to that defined by the road distance numbering. A **SLEEP 0** ensures, that the driver is actually positioned in the new, fully initialized, but empty world. The line **Traffic_ResetSigns (2)** then puts a set of standard signs into the world.

Please note, that this differs from former program versions, where the worlds came up with a set of standard signs without calling **Traffic_ResetSigns**

To better understand the meaning of these terms have a look at the map in figure 3.4. The large numbers are the road numbers and the small numbers denote positions on a road by giving the distance in meters from the beginning of a road in the direction of its end. The green flag labelled “Start” marks the position defined above. The driver is heading from larger to smaller distance numbers, i.e. he is directed towards the traffic lights below the green flag.

Figure 3.4: Map of WORLD_SMALLTOWN



Next, the program calls **World_SetWeather** to set the weather conditions in the virtual world. The weather is set to foggy, which is further specified to mean a sight distance of 350m by **World_SetFogSight**. Next, **System_SetCurrentPos** is used to set an anchor for route calculations. Then, a route for the driver is set with **Traffic_SetRoute**.

The process of assigning routes deserves some further comments. Calculating routes of course needs a start and an end point (more precisely directed points).

The latter has been defined with **System_SetCurrentPos** and the first is given by the driver's current position. Upon receiving the **Traffic_SetRoute** command, the program now calculates the shortest route from start to end position (including direction), which does not contain 180 degree turns and honours traffic rules. The latter point can be neglected by passing an option to **Traffic_SetRoute**.

The point defined by endpos is marked with a yellow flag in figure 3.4 and the path selected by the above rules is depicted with a blue arrow between the two flags.

Similarly to the process of assigning a route to the driver, routes can also be assigned to other vehicles moving in the virtual world (simply pass another vehicle's object id to **Traffic_SetRoute**. If set for a computer controlled vehicle, the vehicle will simply follow the route that is set for it. Afterwards, it will show a behaviour defined by the second argument of **Traffic_SetRoute**.

The driver, of course, cannot be forced to adhere to the route set for him. Instead, the simulator will show hints to the driver similar to those given by a navigation system – red arrows and spoken directional hints. If the driver chooses to leave the precalculated route, it is recalculated on the fly to still navigate him to the set destination point.

Back to the source code, the next command **Surround_SetLight** sets the phase of the next traffic light along the calculated route. In the example, the traffic light at the crossing, where roads 1, 2, 7 and 10 meet, is assigned a new phase. Using the constant **TLIST_RED** the traffic light is set to red. After some seconds, the traffic light then automatically changes to the next phase (depending on the localized program version, this might be red and yellow or green).

The loop between `LABEL prepare` and `GOTO prepare` is a waiting loop. The program leaves the loop by executing `GOTO go`, when the driver first exceeds a speed of 10 km/h (defined in the variable *minspeed*.

Until the driver has arrived at the destination, the program will then loop between `LABEL mainloop` and `GOTO mainloop`. This is accomplished by comparing the vectors **mypos** and *endpos*. The former is a global variable, which always contains the position of the driver's vehicle. The comparision **mypos** > *endpos* thus evaluates to true as soon as the driver has passed *endpos* (in the correct direction). Then, the program jumps to `LABEL ends`, where the simulation is ended with **System_Stop**.

3.8.2 Example Scenario 2

We will now discuss a more advanced scenario.

Listing 3.38: SCENARIOS\CUSTOM\EXAMPLE02.SCE

```
Name: Example 2
Name1: Beispiel 2
Maxlevel: 3
Defaultlevel: 2

BEGIN

LABEL Start
    VAR minspeed = 10 // minimum speed before scenario starts in km/h
    VAR language = 0
    VAR myobjid = 0
    VAR objchild = 0
    VAR objcar = 0
    VAR objcar2 = 0
```

```
    VAR objcar3 = 0
    VAR objcyclist = 0
    VAR objparkcar = 0
    VAR objinsert = 0
    VAR insertflag = 0
    VAR testrouteflag = 0
    VAR roadno = 0

    VAR speed = 50 // variables for inserting vehicles by timer
    VAR counter = 0
    VAR carposition = 60
    TIMER vtimer = 3
    VAR vdensity = 3
    VAR kind = 0

    VEC startpos = 7, 100, 180

    VEC pos1 = 7, 50, 180
    VEC pos2 = 7, 20, 180
    VEC pos3 = 2, 20, 0
    VEC pos4 = 18, 41, 0
    VEC pos5 = 9, 1, 0
    VEC pos6 = 9, 40, 0
    VEC pos7 = 14, 0, 0

    VEC endpos = 16, 40, 180

    IF level >= 3
        pos4.road = 3
        pos4.dist = 27
        pos4.angle = 180
    ELSIF level = 1
        pos4.dist = 38
    ENDIF

    Driver_SetWorld (WORLD_SMALLTOWN)   // course in a town
    Driver_SetPositionDirect (startpos) // start position
    SLEEP 0 // recommended after world and position are set
    Traffic_ResetSigns (2)              // some standard signs
    World_SetWeather (WEATHER_FOG)      // foggy weather}
    World_SetFogSight (0, 350)          // sight of fog in meters}
    language = System_GetLanguage ()
    myobjid = Driver_GetID (0)          // object identifier for own car
    Traffic_SetRoute (myobjid, ENDROUTE_RANDOM, 5,40,0) // set route
    Surround_SetLight (TLST_RED)
    GOSUB traffic

LABEL prepare
    VAR v = 0
    v = Driver_GetSpeed ()
    IF v > minspeed
        GOTO go
    ENDIF
    SLEEP 0
    GOTO prepare

LABEL go

LABEL mainloop

    IF vtimer > vdensity
        GOSUB nextcar
    ENDIF
    IFONCE mypos > pos1
        GOSUB setpos1
    ENDIF
    IFONCE mypos > pos2
        GOSUB setpos2
    ENDIF
    IFONCE mypos > pos3
        GOSUB setpos3
    ENDIF
    IFONCE mypos > pos4
        GOSUB setpos4
```

```
    ENDIF
    IFONCE mypos > pos5
        GOSUB setpos5
    ENDIF
    IFONCE mypos > pos6
        GOSUB setpos6
    ENDIF
    IFONCE mypos > pos7
        GOSUB setpos7
    ENDIF
    IFONCE mypos > endpos
        GOTO ends
    ENDIF
    IF testrouteflag > 0
        roadno = mypos.road // test, whether the user takes the wrong route
        IF roadno = 1 OR roadno = 10 OR roadno = 4 OR roadno = 11 OR roadno = 13
            Error_SetError (ERR_WRONG_WAY)
            SLEEP 2
            GOTO ends
        ENDIF
    ENDIF
    SLEEP 0
    GOTO mainloop

LABEL setpos1
    World_SetFogSight (0, 300)
    RETURN

LABEL setpos2
    testrouteflag = 1
    // from here the test begins, whether
    // the driver is on the correct route.
    World_SetFogSight (0, 250)
    Surround_SetLight (TLST_REDYELLOW)
    Traffic_SetSpeed (objcar, 70, SETSPEED_WISH)
    RETURN

LABEL setpos3
    World_SetFogSight (0, 200)
    System_SetText ("")
    RETURN

LABEL setpos4
    World_SetFogSight (0, 150)
    Traffic_SetSpeed (objparkcar, 40, SETSPEED_WISH) // parked car departs
    Traffic_SetLeaveParking (objparkcar, 4)
    Traffic_SetTrackDeviation (objparkcar, 1.6, SETTRACK_WISH)
    RETURN

LABEL setpos5
    World_SetFogSight (0, 120)
    Traffic_SetSpeed (objcar2, 50, SETSPEED_WISH)
    Traffic_SetSpeed (objcar3, 50, SETSPEED_WISH)
    // bicyle driver starts
    Traffic_SetSpeed (objcyclist, 15, SETSPEED_WISH)
    insertflag = 1
    RETURN

LABEL setpos6
    World_SetFogSight (0, 100)
    RETURN

LABEL setpos7
    IF level >= 3
        World_SetFogSight (0, 80)
    ENDIF
    RETURN

LABEL traffic // insert foreign objects

    objchild=Traffic_InsertVehicle (OBJECT_CHILD, 0, INSFLAG_USEANGLE,\
                                    2, 0, 91) // child
    Traffic_SetRoute (objchild, ENDROUTE_STOP, 10, 30, 180)
    Traffic_SetTrackDeviation (objchild, 4, SETTRACK_IMMEDIATE)
```

```
    Traffic_SetSpeed (objchild, NULL, SETSPEED_WISH)

    objcar=Traffic_InsertVehicle (OBJECT_SMALL_CAR, 0, INSFLAG_DEFAULT,\
                            3, 30, 0) // car
    Traffic_SetRoute (objcar, ENDROUTE_REMOVE, 1, 250, 0)
    Traffic_SetAcceleration (objcar, 8, 8) // acceleration/deceleration
    Traffic_SetSpeed (objcar, NULL, SETSPEED_WISH)

    objcar2=Traffic_InsertVehicle (OBJECT_SMALL_CAR, 0, INSFLAG_DEFAULT,\
                           13, 30, 180) // car
    Traffic_SetRoute (objcar2, ENDROUTE_REMOVE, 4, 100, 0)
    Traffic_SetAcceleration (objcar2, 8, 8) // acceleration/deceleration
    Traffic_SetSpeed (objcar2, NULL, SETSPEED_WISH)

    objcar3=Traffic_InsertVehicle (OBJECT_SMALL_CAR, 0, INSFLAG_DEFAULT,\
                           19, 10, 180) // car
    Traffic_SetRoute (objcar3, ENDROUTE_REMOVE, 4, 90, 0)
    Traffic_SetAcceleration (objcar3, 8, 8) // acceleration/deceleration
    Traffic_SetSpeed (objcar3, NULL, SETSPEED_WISH)

    objcyclist=Traffic_InsertVehicle (OBJECT_CYCLIST, 0, INSFLAG_DEFAULT,\
                            9, 15, 0)
    Traffic_SetRoute (objcyclist, ENDROUTE_REMOVE, 16, 20, 180)
    Traffic_SetAcceleration (objcyclist, 1.5, 1.5) // acc-/deceleration
    Traffic_SetSpeed (objcyclist, NULL, SETSPEED_WISH)
    Traffic_SetTrackDeviation (objcyclist, 2.3, SETTRACK_IMMEDIATE)
    // parked car
    Traffic_InsertVehicle (OBJECT_PARK_CAR, 0, INSFLAG_DEFAULT, 3, 20, 180)

    // parked car will depart later
    objparkcar=Traffic_InsertVehicle (OBJECT_SMALL_CAR, 0, INSFLAG_DEFAULT,\
                            3, 10, 180)
    // parked at the edge of the street
    Traffic_SetTrackDeviation (objparkcar, 3.5, SETTRACK_IMMEDIATE)
    Traffic_SetRoute (objparkcar, ENDROUTE_REMOVE, 4, 100, 0)
    Traffic_SetSpeed (objparkcar, NULL, SETSPEED_WISH)
    Traffic_SetAcceleration (objparkcar, 6, 6) // acceleration/deceleration
    RETURN

LABEL nextcar
    IF insertflag = 0
        RETURN
    ENDIF
    IF counter > 20
        Error_SetError (ERR_WRONG_WAY)
        // Error and abortion, because waiting too long
        // before crossing the street
        SLEEP 2
        GOTO ends
    ENDIF
    kind = counter % 2
    kind = kind + 1 // first a small car, then a truck
    objinsert = Traffic_InsertVehicle (kind, 0, INSFLAG_DEFAULT,\
                           11,carposition, 180)
    Traffic_SetRoute (objinsert, ENDROUTE_REMOVE, 5, 20, 0)
    Traffic_SetSpeed (objinsert, speed, SETSPEED_WISH)
    Traffic_SetAcceleration (objinsert, 10, 8)
    counter = counter + 1
    IF speed > 30
        CALC speed = speed - 3
    ENDIF
    vdensity = vdensity + 0.5
    IF carposition < 80
        CALC carposition = carposition + 2
    ENDIF
    vtimer = 0
    RETURN


LABEL ends
    System_Stop ()
END
[CHECKSUM]
CHECKSUM=653673
```

```
VERSION=$Rev: 15803 $ $Date: 2017-06-29 11:59:32 +0200 (Do, 29 Jun 2017) $ $URL: svn+ssh://svnflinu
```

This scenario includes several difficulty levels. The global variable **level** is used to reflect the difficulty level, which has been set for a scenario. The number of selectable difficulty levels is set by **Maxlevel** in the header of the scenario. The level being offered is set with **Defaultlevel**.

When the scenario is initialized, the subroutine at `LABEL traffic` is called. Here, some objects like foreign vehicles and pedestrians are created using the function **Traffic_InsertVehicle**. The parameters are object type, offset (here always 0), a bit mask modifying details of the creation process, road number, road distance from beginning of the road and angle. The return value of this function is the identifier of the newly created object and is stored e.g. in the variable *objchild*.

To set a speed for an object, the function **Traffic_SetSpeed** can be used. Its parameters are an object identifier, the desired speed in km/h and a mode constant. Use a mode of 1 to set the absolute speed of an object.

To make an object drive as fast as being set with **Traffic_SetSpeed** with only a short delay, use **Traffic_SetAcceleration** and set a high acceleration value. The parameters of this functions are an object identifier, acceleration and deceleration (both measured in meters per second squared).

Usually a newly created object is positioned on the right[8] lane of a road. To vary the transverse position of an object (e.g. to position a cyclist some centimeters away from the edge of the road, use the function **SetTrackDeviation**, which can also be done during driving to achieve special effects. Another way to set the transverse position at the time of creation would have been to use more parameters to **Traffic_-InsertVehicle**.

In the loop at `LABEL mainloop` the own position is compared with several position vectors. The vectors *pos1*, *pos2* etc. contain positions along the route. If the driver passes one of these positions the corresponding subroutine at `LABEL setpos1`, `LABEL setpos2` etc. is called. The actions performed by the subroutines are meant to be executed only once (and not on each loop after having passed the position). Thus, the conditional branching to the subroutines is performed using **IFONCE** instead of a simple **IF**. The actions started by the subroutines include reducing the fog sight, switching a traffic light and starting a parked car and a cyclist.

The loop at `LABEL mainloop` also tests, whether the driver stays on the predefined route or leaves it. This is done by extracting the current road from **mypos** and comparing it with several faulty choices. In addition, this demonstrates the use of a global flag to conditionally perform checks. An alternative to the road number approach would have been to use the function **Traffic_GetRouteStatus**. If the driver has left his path, an error message is displayed with **Error_SetError** and after a delay of 2 seconds the simulation is aborted with **System_Stop**.

In the subroutine at `LABEL setpos5` the flag *insertflag* is set to 1. Afterwards, the subroutine at `LABEL nextcar` creates a new vehicle on a main road, which the driver has to cross. This occurs whenever the timer variable *vtimer* exceeds *vdensity*. After each vehicle creation, the timer is reset and *vdensity* is increased. Consequently, the traffic density on the main road decreases with time. In addition, the *speed* of new vehicles is reduced on each creation.

[8] Remember our conventions concerning left and right.

It is not a good idea to infinitely insert cars into the world. On the one hand, the simulation becomes slower when having to calculate many vehicles. On the other hand, the graphics database only contains a limitted pool of different vehicles, which makes the simulation look unrealistic when too many vehicles are visible at once. In this scenario the problem is circumvented by stopping the simulation after 20 inserted cars. Until then, a reasonable driver should have been able to cross the main street.

Chapter 4

Values and Variables

4.1 Variables and Vectors

4.1.1 Scalar Variables

Variables in scenarios are usually floating point variables, but can of course also hold integer values. When using variables as parameters of functions expecting integer parameters, the values are rounded before being passed to the function.

The name of a variable must consist of alphanumeric characters and the underscore character. The first character must be a letter (i.e. must be in [**a-z**,**A-Z**]. All other characters in a variable name may also be numbers [**0-9**] or the underscore character _. The script compiler is case sensitive, but in order to avoid confusion, variable names only differing in case are rejected.

Several otherwise legal names, so-called reserved words, are used by the system and may not be used as variable names. Reserved words are language directives like **VAR**, **IF** or **FUNC**, the names of functions and, of course, predefined global variables like **mypos**.

Every variable except for predefined global variables (c.f. section 4.1.5) must be declared before it can be used. When a variable is declared, it may be given an initial value. If no explicit initial value is specified, the variable is initialized to zero. Note, that a variable is reinitialized whenever the flow of execution passes the defining line. This is also true, if you did not explicitly specify an initial value.

Listing 4.1: VAR Syntax

```
VAR name [initial value]
```

Listing 4.2: VAR Examples

```
VAR weather
VAR weather 1
VAR weather WEATHER_RAIN
```

The first examples initializes the variable *weather* to 0 (which means sunny). The second one initializes it to 1, which means foggy. The third example uses a predefined constant (c.f. appendix A).

4.1.2 Vectors

Special kinds of variables, called vectors, consist of more than one value. For the sake of future compatibility, you should imagine a vector as an infinite series of simple variables, most of which are set to 0. Of course, only a finite number of values of a vector are actually stored and can be programmatically set, but the maximal number of writeable elements may be increased in the future. Like simple variables, vectors must be declared.

Listing 4.3: VEC Syntax

```
VEC name [[=] v1, v2 [,v3 [,...] ] ]
```

If you specify initial values at all, you must specify at least two of them. At the time of writing, the maximal number of initial values is 5. As with simple variables, unspecified initial values default to zero.

Listing 4.4: VEC Examples

```
VEC endpos
VEC endpos = 3, 100
VEC endpos = 3, 100, 0
```

The first example initializes the vector endpos to all zero (0, 0, ...). The two following examples are completely equivalent. When used as a road position, they specify 100m from the beginning of road no. 3. The first example (if used as such) specifies an invalid road position.
After a vector has been declared, you can use it as a whole to

1. copy a vector to another vector,
2. use a vector to specify multiple function arguments (c.f. chapter 5) or
3. store the result values of a function in a vector.

In many situations, though, you will want to access the single components of a vector. This can be done by appending a suffix to the vector's name[1] like in

Listing 4.5: Vector Element by Suffix

```
VAR somevariable
VEC somevector
...
somevariable = somevector.angle
```

which will assign the variable *somevariable* the value of *somevector*'s component named **angle**.

[1] If you are acquainted with an object oriented language, you will be reminded of accessing member variables of an object. Note though, that you can access the same member variable using different names.

Table 4.1: Vector Component Naming Schemes

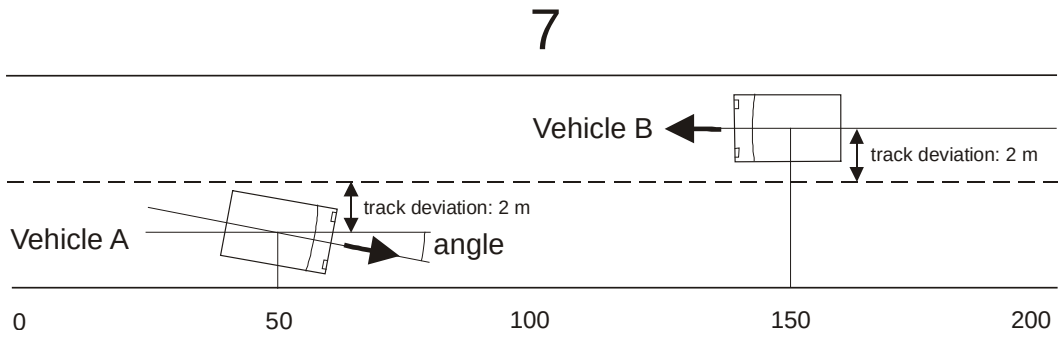
#	Road Position	Cartesian World Position	Unspecified
0	**.road**	**.x**	**.v1**
1	**.dist**	**.y**	**.v2**
2	**.angle**	**.z**	**.v3**
3	**.track**	**.yaw**	**.v4**
4	**.world**	**.pitch**	**.v5**
5	**.offx**	**.roll**	**.v6**
6	**.offy**		**.v7**
7	**.offz**		**.v8**
8	**.offyaw**		**.v9**
9	**.offpitch**		**.v10**
10	**.offroll**		**.v11**
11			**.v12**
⋮			⋮

Most often, vectors are used to store positions on roads. For this usage the naming scheme denoted **Road Position** in table 4.1 is best suited. The roads in world maps are identified by strictly positive numbers stored in the **.road** component. In order to determine the position on a road, roads always have a beginning and an end, so that you can specify a position by specifying the distance from the beginning of the road. This value, measured in meters, is stored in the **.dist** component. Assigning beginning and end to a road also defines a natural direction on a road (from beginning to end). A vehicle's orientation is specified by giving the angle between the natural direction of the road and the direction of the vehicle. This angle is stored in degrees in the **.angle** component. Positive values mean "to the right".[2] To completely specify the position on a road, the transverse direction is still missing. This is stored in **.track**, which measures the distance from the middle of the road to the right. For many purposes you may set this component to 0, which instructs the program to choose a sensible value itself. The **.world** component contains the current map number (c.f. function **Driver_SetWorld**). When used as a function argument, this component only has an effect for **Driver_SetPosition** and **Driver_SetPositionDirect**, where a nonzero value can be used to replace a preceding **Driver_SetWorld** call. The **.off...** components specify an optional cartesian offset only supported by some functions. The meaning of the numbers is the same as for the absolute cartesian positions described below.

[2]Remember our conventions about left and right. If one drives right in your country, angles are measured in clockwise direction. If you are driving left, angles are mathematically positive, i.e. anticlockwise.

A word of caution: Whenever you pass a vector road position to the simulator, you should always make sure that all unspecified components are zero or you have received the vector from the simulator. Otherwise, your scenarios may show unexpected behaviour in future software releases, where additional vector components might be used to store internal states or the like. ***Foerst GmbH*** will ensure, that 0 will always produce neutral results with respect to future extensions.

Figure 4.1: Example Road Positions



In figure 4.1 you find an example for road positions (drawn for countries where one drives right). Vehicle A has position vector (7, 50, angle, 2) and Vehicle B is at position (7, 150, 180, 2).

A second common usage for vectors is to store cartesian world positions, which might be obtained e.g. with the function **Traffic_GetAbsolutePosition**. For this usage, the second naming scheme in table 4.1 is best suited. Lengths are specified in m and angles in degree. The meaning of the coordinates is z forward, x to the right (to the left) and y upwards. The rotational angles are yaw around the y axis from z to x (i.e. clockwise/anticlockwise), $pitch$ around x from y to z (i.e. nose down) and $roll$ around z from x to y (i.e. tilt left/right).

Using the third naming scheme, you can access all of a vector's supported components. Although you could always use this naming scheme for any purpose, for readability purposes you are encouraged to use one of the other schemes wherever applicable.

4.1.3 Timers

Timer variable are declared using

Listing 4.6: TIMER Syntax

```
TIMER name [[=] initial value]
```

In almost all respects, timers behave like simple variables declared using **VAR**. The only difference is, that timers are automatically incremented after each command execution by the amount of time (measured in seconds) the command has taken to execute. If you e.g. assign a timer variable the value 2 and then execute some commands taking a total time of $9ms$, the timer variable will afterwards have a value of 2.009. The precision of timer variables is $1ms = 0.001s$.

4.1.4 Strings

String variables are declared using

Listing 4.7: STRING Syntax

```
STRING name [[=] "initial value"]
```

As the name suggests, they hold strings. You can give them an initial value using an optional quoted string. The default initial value is the empty string.

4.1.5 Global Variables

Global variables are variables you do not (and cannot) declare, but are nevertheless present in every scenario. At the time of writing, there are three global variables:

Listing 4.8: Global Variables

```
VEC mypos
VEC levels
VAR level       // level = levels.v1
```

The first one always contains the current road position of the driver's vehicle. The second reflects the options the user has chosen in the simulator's menu. If your scenario doesn't specify multiple levels using the **Maxlevel** header directive, **levels** will be all zeroes. **level** exists for compatibility reasons.

In addition, there are two local global vectors:

Listing 4.9: Local Global Variables

```
LOCALVEC argv
LOCALVEC retv
```

The first one holds the arguments given to the current **PROC** as specified by the corresponding **CALL**. You can set the second to specify the return value(s) of the current **PROC**.

4.2 Constants

Apart from using variables, you can specify values directly in your source code. We have already seen one usage above (giving initial values to vectors and variables), but constants may also be used as arguments to functions or can be assigned to variables after their declaration. Two kinds of constants are used – string constants and numeric constants.

String constants are specified by enclosing the text in quotation marks. Quotation marks and backslashes must be escaped. A newline character can be embedded using `\n`.

Listing 4.10: Example String Constant

```
"A \"first\" line and\nanother line begun using \\n."
```

There are many different ways to specify numeric constants. Apart from the usual ways to write floating point numbers, integers may also be written as hexadecimal numbers. In this case, you must specify the suffix **H**.

Listing 4.11: Examples of Numeric Constants

```
3      // integer constant
3.1415 // floating point number
2.9E8  // scientific notation: 2.9 * 10^8
27     // another integer
1BH    // the same integer in hexadecimal notation
```

In addition to directly specifying numeric constants, you can also use predefined constant mnemonics like in

Listing 4.12: Using a Predefined Constant

```
LET weather WEATHER_RAIN
```

Lists of all predefined constants may be found in appendix A.

You are strongly advised to use such constant definitions wherever applicable. This both enhances the readability of your code and ensures compatibility with future extensions. The numeric values associated with predefined constantants are subject to change in future software versions. They are listed in appendix A only in order to help the reader interpret debug output, **not** to be used in scenario files. For bitmask arguments to functions, use constants and combine them using the **CALC** directive.

4.2.1 Units

All physical quantities are measured in SI (metric) units with the exception of speeds, which are measured in km/h, as this is the metric unit usually used in traffic rules. For the convenience of programmers located in countries using the traditional British unit system, several conversion factors have been defined as constants. In particular, speeds may be converted using the constant **KILOMETER_PER_MILE**. There are constants for other unit conversions as well. For a complete list refer to appendix A.30.

4.3 Assignments

As was already briefly discussed in section 3.7.1.2, you assign values to variables using the **LET** and **LETVEC** directives.

Listing 4.13: Assignment Syntax

```
LET variable source
LET vector = v1, v2 [, v3 [,...]]
LETVEC scalar = vector index
LETVEC vector index = scalar
PACKVEC vectorp = vectoru1, ..., vectorun
PACKVEC vectoru1, ..., vectorun = vectorp
```

For the first variant, *variable* is the name of a string or scalar variable, a timer, a vector component or a vector. In the latter case, *source* must also be the name of a vector variable. For scalar variables, timers and vector components, *source* can be a scalar variable or timer name, a constant or a vector component. For string variables, *source* must be a string constant, another string variable or a scalar variable (which is then converted to a string).

The second variant is for assigning values to all vector components at once. You **must** specify at least two values, while further values are optional and default to zero. With this syntax, *v1*, *v2*, ... must be constants.

The **LETVEC** directive in its first form takes a vector variable *vector* and an integer *index* (either scalar variable or integer constant) and stores the corresponding element of *vector* in the scalar variable *scalar*. The index is zero-based, i.e.

Listing 4.14: Example LETVEC Usage

```
LETVEC scalar = vector 3
```

copies the 4th component of *vector* into *scalar*.

The second form of the **LETVEC** directive is used for the opposite direction: it takes a scalar specified by *scalar* (variable or constant) and stores it in the vector component specified by *vector* and *index*.

There are shorthand notations for the **LETVEC** directives:

Listing 4.15: Alternatives to LETVEC

```
// same as LETVEC scalar = vector index:
LET scalar = vector[index]
// same as LETVEC vector index = scalar
LET vector[index] = scalar
```

All functions and procedures take a single vector as their argument. If you need to pass multiple vectors, you can use the **PACKVEC** directive. It is best explained using an example:

Listing 4.16: Example PACKVEC Usage

```
VEC a = 1,2,3,4
VEC b = 5,6,7,8,9
VEC c
VEC d
```

```
VEC e

PACKVEC c = a,b
PACKVEC d,e = c
```

After the first **PACKVEC** has been executed, a and b are packed into c, i.e. $c = (1, 5, 2, 6, 3, 7, 4, 8, 0, 9)$. The second **PACKVEC** unpacks c into d and e, i.e. $d = (1, 2, 3, 4)$ and $e = (5, 6, 7, 8, 9)$.

4.4 Calculations

You can perform basic arithmetic and logical operations as shown below.

Listing 4.17: CALC Syntax

```
[CALC] variable = value1 operator value2
```

variable must be the name of a simple variable or timer, while *value1* and *value2* must be simple variables, timers or constants. *operator* may be one of the operations listed in table 4.2. You can and should omit the **CALC** keyword.

Table 4.2: Operations for the CALC Command

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	division remainder (modulo)
&	bitwise and[a]
|	bitwise or
^	bitwise exclusive or
<<	bit shift left
>>	bit shift right[b]

[a] For bitwise operations, the values are first rounded to 32bit integers.
[b] Do not rely on the highest bit behaviour (arithmetic or logical).

Chapter 5

Functions

5.1 Calling Functions

The general syntax of a function call is

Listing 5.1: Function Call Syntax

```
[FUNC] [r1 [,r2 [,...]] =] function(v1 [, v2 [,...]])
```

This will execute the function *function* passing *v1*, ... as arguments. The return value(s) are stored in *r1*, ... You can omit the **FUNC** keyword. As a side effect, you can then e.g. perform calculations inside the argument list. For details, c.f. chapter 6.

5.1.1 Function Arguments

The function *function* is executed with arguments *v1*, *v2*, ... The arguments can be constants, scalar variables or strings. You can also use a vector as an argument. In this case, the vector must be the last argument and its components replace all scalar arguments from the point of its occurrence. As an example, consider a function *somefunction*, which takes four scalar arguments. Then,

Listing 5.2: Vector as Argument, Version 1

```
VAR s
VEC v
...
FUNC somefunction (s, v)
```

is equivalent to

Listing 5.3: Vector as Argument, Version 2

```
VAR s
VEC v
...
FUNC somefunction (s, v.v1, v.v2, v.v3)
```

Note though, that the latter is not legal code – you cannot directly use vector components as function arguments. If you want to do so, first copy the vector component to scalar helper variables.

Listing 5.4: Vector as Argument, Version 3

```
VAR s
VEC v
VAR h1
VAR h2
VAR h3
...
LET h1 v.v1
LET h2 v.v2
LET h3 v.v3
FUNC somefunction (s, h1, h2, h3)
```

With the documentation of each function later in this chapter, it will be noted which kinds of arguments each function expects. There are three types of arguments: **integer**, **float** and **string**. You find a list of possible arguments for given argument type in table 5.1. The column labelled **VEC** means that the argument can be provided by a vector (component).

Table 5.1: Argument Types

	VAR TIMER	STRING	VECTOR	constant
integer	x		x	must be integer
float	x		x	float or integer
string	a	x		quoted string

x: *allowed* a: *automatic conversion*

Passing a non-integral constant for an integer argument will result in a compiler error. For variables, their contents will be rounded instead.

5.1.2 Optional Arguments

Some arguments in the function reference later in this chapter are marked as optional. This means, that you can omit these arguments. Omitting an optional argument is equivalent to specifying zero as its value – for functions with optional arguments this will evoke some kind of default behaviour.

5.1.3 Return Values

All functions return one or more scalar result values. For a single return value, you may assign the return value to a scalar variable

Listing 5.5: Storing Scalar Function Results

```
VAR r
FUNC r = somefunction()
```

or you discard the return value by omitting the result variable:

Listing 5.6: Discarding Function Results

```
FUNC somefunction()
```

The various possibilities to store result values of functions returning multiple scalars are best explained using an example. To this end, consider a function *examplefunc*, which takes no arguments and returns three scalars.

Listing 5.7: Multiple Function Results

```
VAR r1
VAR r2
VAR r3
VEC v
FUNC r1, r2, r3 = examplefunc()
FUNC v = examplefunc()
FUNC examplefunc()
FUNC r1, -, r3 = examplefunc()
FUNC -, -, r3 = examplefunc()
```

The first example stores the three result values in the variables *r1*, *r2* and *r3*. The second example instead stores them in the first three components of the vector *v*. The third example discards all return values and the last two examples selectively store only some of the return values. Use the hyphen to specify result values you wish to discard.

In the Function Reference below, many functions are marked as having no return values. This means, that they do not return values. Like all other functions, they do have a return value – in this case an error code (c.f. appendix A.16).

5.2 Function Reference

In this section, an exhaustive list of all functions is presented. The functions are grouped into several categories according to the kinds of objects they query and/or manipulate.

Functions marked as *deprecated* are only listed for compatibility reasons with older scenario files. Do not use them in new scenario programs. They may be removed in future software versions.

5.2.1 Driver Functions

The functions listed in this section all manipulate or query properties of the driver's own vehicle.

5.2.1.1 Driver_GetAbsolutePosition

Get the own vehicle's position in cartesian world coordinates. The coordinate system is left-handed and y is the upwards direction. The z-direction will at times be called "north". All coordinates are measured in m. When heading north, x is to the right[1].

Arguments: none

Return Values:

x	float	Cartesian x-coordinate in m
y	float	Cartesian y-coordinate in m
z	float	Cartesian z-coordinate in m

See also: **Traffic_GetAbsolutePosition**, **Driver_GetAngles**

5.2.1.2 Driver_GetAcceleration

Get the own vehicle's forward acceleration. When currently moving in forward direction, positive values mean acceleration and negative values mean deceleration.

Arguments: none

Return Values:

accel	float	Current acceleration in m/s^2

5.2.1.3 Driver_GetAcceleratorPedal

Obtain the driver's accelerator pedal position in percent. A value of 0 means the driver does not touch the accelerator pedal, a value of 100 is kick-down position.

Arguments: none

Return Values:

perc	float	Acceleration pedal position in %

[1] For commonwealth countries: The coordinate system is right-handed and x is to the left.

5.2.1.4 Driver_GetAngles

Obtain the rotational orientation of the driver's car relative to the x, y, and z axes. Angles are measured in degree in the range $(-180, 180]$.

Arguments: none

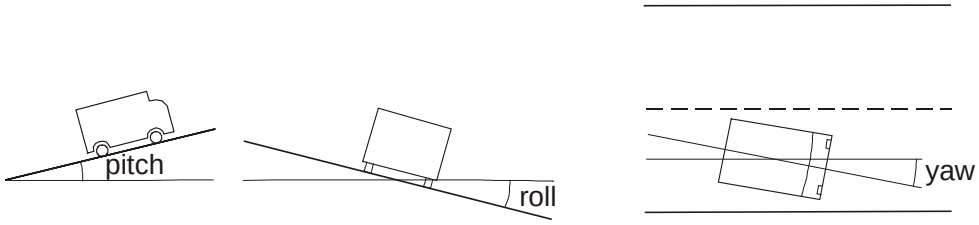
Return Values:

pitch	float	Angle about the x-axis
yaw	float	Angle about the y-axis
roll	float	Angle about the z-axis

See also: **Traffic_GetAngles**, **Driver_GetAbsolutePosition**

Remarks: The angles' names are motivated by a car initially standing upright and facing north. Then, the angles' meanings are as depicted in figure 5.1. When specifying a full orientation, the single rotations must be performed in the order yaw, pitch, roll.

Figure 5.1: Meaning of Angles



5.2.1.5 Driver_GetBlinker

Returns either the current or the recent state of the driver's car's blinker lights.

Arguments:

mode	integer, opt.	past or current, see Remarks

Return Values:

blinker	integer	kind of blinker, see Remarks

Remarks: If mode is 1, the current state ist returned. If mode is 0, the return value is the last nonzero state during a time interval, which by default is $3000ms$. This value can be tuned by setting **BlinkerTorzeit** in SIMUCONF.CFG.

The meanings of possible return values are:

0: no blinker

1: right blinker

2: left blinker

3: hazard warning lights

5.2.1.6 Driver_GetBrakePedal

Obtain the driver's brake pedal position in percent. A value of 0 means the driver does not touch the brake pedal, a value of 100 is emergency stop.

Arguments: none

Return Values:

perc	float	Brake pedal position in %

5.2.1.7 Driver_GetClutchPedal

Obtain the driver's clutch pedal position in percent. A value of 0 means the driver does not touch the clutch pedal, at a value of 100 the clutch is disengaged.

Arguments: none

Return Values: *perc* float Clutch pedal position in %

5.2.1.8 Driver_GetDistance

Obtain the full distance driven by the driver since the start of simulation.

Arguments: none

Return Values: *dist* float Distance in m.

5.2.1.9 *Driver_GetGear (deprecated)*

Queries the currently selected gear. 0 means idling, 6 reverse.

Arguments: none

Return Values: *gear* integer see above

5.2.1.10 Driver_GetGearShift

Queries the type of gear box in the driver's car.

Arguments: none

Return Values: *geartype* integer see Remarks

Remarks: possible return values are:

0: manual gearshift

1: automatic gearshift

2: tiptronic

5.2.1.11 Driver_GetHorn

Queries whether the driver's horn is activated.

Arguments: none

Return Values: *horn* integer 1 if driver is honking, 0 else

5.2.1.12 Driver_GetID

Get object identifiers for vehicles (in particular the driver's).

Arguments: *index* integer index of the car: 0 is the driver's car, 1 the first foreign car inserted, ...

Return Values: *id* integer the object ID. Valid object Ids are strictly positive. Test the ID for positivity before using it.

5.2.1.13 Driver_GetIgnition

Queries the status of the driver's ignition.

Arguments: none

Return Values: *ignition* integer 1 if turned on, 0 if turned off.

5.2.1.14 Driver_GetInstrument

Get values displayed on or selected by various cockpit instruments.

Arguments: *instrument* integer use one of the constants in appendix A.22.

Return Values: *take-over* integer 1 if taken over by software, 0 otherwise

value float current value

value_orig float would-be value if not taken over.

See also: **Driver_SetInstrument**

5.2.1.15 Driver_GetLateralAcceleration

Get the current lateral or centrifugal acceleration of the car (or felt inside the car, resp.) in m/s^2

Arguments: none

Return Values: *accel* float

5.2.1.16 Driver_GetLight

Query the current state of the driver's headlights

Arguments: none

Return Values: *light* integer 0: turned off

1: dimmed headlights

2 or 3: high beam

5.2.1.17 Driver_GetRPM

Query the driver's motor's revolutions per minute.

Arguments: none

Return Values: *rpm* float

5.2.1.18 Driver_GetSlideAngle

Under normal driving conditions, the orientation of the own car equals its direction, but under certain conditions (aquaplaning, ice) this does not hold anymore. The relative angle between the two directions can be queried using this function.

Arguments: none

Return Values: *angle* float angle in degrees — positive values mean car's nose is right of direction of motion. range: $[-180, 180]$

5.2.1.19 Driver_GetSpeed

Returns the speed of the own car in km/h.

Arguments: none

Return Values: *speed* float

5.2.1.20 Driver_GetStarter

Query the current starter position.

Arguments: none

Return Values: *starter* integer 1 if starter is switched on, 0 otherwise

5.2.1.21 Driver_GetSteeringWheel

This function returns the own vehicle's current steering wheel position in degrees. Values greater than zero mean steering right. Since the range of valid positions of a steering wheel may be greater than 360 degrees, values outside the range $[-180, 180]$ can occur.

Arguments: none

Return Values: *angle* float

5.2.1.22 Driver_GetWiper

Returns the current status of the driver's windscreen wipers.

Arguments: none

Return Values: *wiper* integer The possible return values are listed in appendix A.20

5.2.1.23 Driver_SetAbs

Switch the driver's ABS (anti-blocking-system) on and off.

Arguments: *flag* integer 1 turns the ABS on, 0 switches it off

Return Values: none

5.2.1.24 Driver_SetExtraSystem

Switch the extra systems on and off.

Arguments: *flag* integer 1 turns the system on, 0 switches them off

Remarks: see appendix A.45 for available systems.

Return Values: none

5.2.1.25 Driver_SetAbsoluteOrientation

Sets the driver's world orientation.

Arguments: *yaw* float
pitch float angles (in degree) about the cartesian axes
roll float

Return Values: none

See also: **Driver_GetAngles**, **Driver_SetAbsolutePosition**

5.2.1.26 Driver_SetAbsolutePosition

Sets the driver's world position.

Arguments: *x* float
y float cartesian coordinates in m
z float

Return Values: none

See also: **Driver_GetAbsolutePosition**

Remarks: Use with caution if at all. Conversion of road parameters into cartesian parameters is a fast operation, but the reverse is not. In addition, the function easily fails if you specify invalid coordinates.

5.2.1.27 Driver_SetAcceleratorPedal

Programatically set the position of the accelerator pedal.

Arguments: *perc* integer — position of the accelerator pedal in percent. 100 means full acceleration.

Return Values: none

See also: **Driver_GetAcceleratorPedal**, **Driver_SetDynamicLongitude**

Remarks: This function sets the position of the accelerator pedal and takes control away from the human driver. Human control is restored by passing -1 for *perc*. In order to execute this function, first completely take away control from the human driver (function **Driver_SetDynamicLongitude**. Afterwards, you may restore control over other settings to the human driver by again calling **Driver_SetDynamicLongitude**.

Listing 5.8: Driver_SetAcceleratorPedal (Example Usage)

```
FUNC Driver_SetDynamicLongitude (1)
FUNC Driver_SetAcceleratorPedal (20)
FUNC Driver_SetDynamicLongitude (0)
```

5.2.1.28 Driver_SetBrake

Programatically set the position of the brake pedal.

Arguments: *perc* integer — position of the brake pedal in percent. 100 means maximum braking.

Return Values: none

See also: **Driver_GetBrakePedal**, **Driver_SetAcceleratorPedal**, **Driver_SetDynamicLongitude**

Remarks: This function sets the position of the brake pedal and takes control away from the human driver. Human control is restored by passing -1 for *perc*. In order to execute this function, first completely take away control from the human driver. For an example, c.f. function **Driver_SetAcceleratorPedal**.

5.2.1.29 Driver_SetDynamicLateral

Programatically (de)activate transverse driver control.

Arguments: *flag* integer — 1 transfers control to the computer, 0 back to the human driver.

Return Values: none

See also: **Driver_SetDynamicLongitude**

Remarks: Perhaps slightly counterintuitively, passing *flag* 1 disables the transversal dynamic model.

5.2.1.30 Driver_SetDynamicLongitude

Programatically (de)activate longitudinal driver control.

Arguments: *flag* integer — 1 transfers control to the computer, 0 back to the human driver.

Return Values: none

Remarks: Perhaps slightly counterintuitively, passing flag 1 disables the longitudinal dynamic model. Only then, velocities can be directly controlled. For *flag* 0 (the default), speed is controlled by acceleration, which in turn depends on accelerator and brake pedal positions as well as the current gear.

5.2.1.31 Driver_SetGearShift

Sets the type of gear box in the driver's car.

Arguments: *geartype* integer — use one of the constants in appendix A.21.

Return Values: none

See also: **Driver_GetGearShift**

5.2.1.32 Driver_SetInstrument

Sets the values displayed on or selected by various cockpit instruments.

Arguments:

instrument	integer	use one of the constants in appendix A.22.
overrule	integer	pass 1 to manually set *instrument* to *value* or 0 to return control to the simulator/driver.
value	float	new instrument value, only used if *overrule*=1. The meaning depends on the instrument, the valid ranges may also be found in appendix A.22.

Return Values: none

5.2.1.33 Driver_SetMotorSound

Switch motor sound on or off.

Arguments: *sound* integer — motor sound is switched on for *sound* = 1 and off for *sound* = 0.

Return Values: none

5.2.1.34 Driver_SetWorld

Sets the map used in the scenario.

Arguments: *world* integer use one of the constants in appendix A.1

Return Values: none

See also: **Driver_SetPositionDirect**, **Driver_SetPosition**

Remarks: This usually is the first function called in any scenario script. Most other functions refer to data (such as road numbers) defined by the map. Note, that this function alone only sets an internal flag. The actual change of world is performed when the driver is positioned inside the new world. At the beginning of a scenario, this cannot be done smoothly. Hence, you must use **Driver_SetPositionDirect**.

Instead of using this function, you can pass a fifth argument to **Driver_SetPositionDirect** or **Driver_SetPosition**.

Listing 5.9: Standard Scenario Opening Sequence

```
FUNC Driver_SetWorld(some_world)
FUNC Driver_SetPositionDirect(some_position)
SLEEP 0
```

Also note, that not all maps listed in appendix A.1 are equally well suited for usage in a scenario script – for instance not all worlds support foreign vehicles. We recommend the usage of one of the following worlds:

- **WORLD_ROUNDCOURSE**
- **WORLD_MOTORWAY**
- **WORLD_MOUNTAIN**
- **WORLD_SMALLTOWN**
- **WORLD_TOWN**

5.2.1.35 Driver_SetPosition

Set the current position of the driver's vehicle (smooth variant).

Arguments:

roadno	integer
distance	float
angle	float
track	float, opt.
world	integer, opt.

Return Values: none

See also: **Driver_SetPositionDirect**

Remarks: The arguments have the same meaning as described for road position vector components (section 4.1.2). Hence, you can pass a road position vector instead of single arguments.

The actual change of position is not executed immediately, but with a slight delay. If you want an immediate change of graphics, call **Driver_SetPositionDirect** instead.

If you pass a nonzero *world* argument, the effect is the same as calling **Driver_SetWorld** immeadiately before this function.

Execution of this function may have many side effects. Hence, a `SLEEP 0` is recommended after the function call.

5.2.1.36 Driver_SetPositionDirect

Set the current position of the driver's vehicle (immediately).

Arguments:

roadno	integer
distance	float
angle	float
track	float, opt.
world	integer, opt.

Return Values: none

See also: **Driver_SetPosition**

Remarks: The arguments have the same meaning as described for road position vector components (section 4.1.2). Hence, you can pass a road position vector instead of single arguments.

The change of graphics to the new position is executed immediately, which might appear disturbing. For a smooth change, call **Driver_SetPosition** instead.

If you pass a nonzero *world* argument, the effect is the same as calling **Driver_SetWorld** immeadiately before this function.

Execution of this function may have many side effects. Hence, a `SLEEP 0` is recommended after the function call.

5.2.1.37 Driver_SetSpeed

Programatically set the driver's speed.

Arguments: *v* float new speed in km/h

Return Values: none

See also: **Driver_SetDynamicLongitude**

Remarks: This function immediately sets the current speed of the driver's vehicle. This obviously only makes sense when control has been taken away from the human driver. Call **Driver_SetDynamicLongitude** first.

5.2.1.38 Driver_SetSteeringWheel

Programatically set the position of the steering wheel.

Arguments:

flag	integer	1 takes control of the steering wheel, 0 restores it to the driver.
angle	float	Angle in degrees. Values greater than 0 mean steering to the right.

Return Values: none

See also: **Driver_SetDynamicLateral**

Remarks: This function sets the position of the steering wheel. It can only be executed while transverse control is transferred to the computer using **Driver_SetDynamicLateral**.

5.2.1.39 Driver_SetCollision

Invoke an accident procedure as if the driver has made a collision. This shows the error display and performs the accident reset, which may move the driver away from the current position.

Arguments:

object	integer	involved opponent object id, or 0 if none
error	integer	the error code to display and record for this accident, see appendix A.17
data	float, opt.	an optional data value (meaning depends on error)
message	string, opt.	optional string value (meaning depends on error)

Return Values: none

Remarks: This function can only be used during a ride. It will interrupt a ride and perform the reset even if accidents are configured to be ignored. The resulting situation after the reset depends upon the configured accident-reset-mode. In many scenarios an accident will also terminate the ride.

Execution of this function may have side effects. Hence, a `SLEEP 0` is recommended after the function call.

5.2.2 Error Functions

The functions in this section are concerned with errors the driver has made (e.g. disregarded traffic lights, speed limit, etc.) and more general system states. You can query errors, determine default actions of the simulator software and programmatically display messages.

5.2.2.1 Error_GetErrorCount

Get the number of errors the driver has made.

Arguments:	*error*	integer	one of the error codes in appendix A.17 or 0
	error2	integer,opt.	further error codes, c.f. remarks.
	...		
Return Values:	*number*	integer	number of errors
	fvalue	float	the numeric value associated with the last occurence
	svalue	string	the string value associated with the last occurence
	duration	float	The total duration (where that makes sense)

Remarks: The function returns the number of occurrences of the error $error$ or the total number of errors if $error = 0$. If you specify multiple errors, the corresponding numbers are summed up and no additional data is returned.

5.2.2.2 Error_GetErrorMask

Get a bit mask indicating which of the most common traffic rule violations the driver has committed.

Arguments: none

Return Values:	*violate*	integer	sum of values for various rule violations, see remarks

Remarks: The result is the sum of one or more of the following constants:

- 1: accident
- 2: too fast
- 4: traffic lights disregarded
- 8: right of way disregarded
- 16: no overtaking disregarded
- 32: special constant: no simulation running, i.e. function call makes no sense.

5.2.2.3 Error_GetLastError

Get the most recent traffic rule violation the driver has committed.

Arguments: none

Return Values:

error	integer	one of the error codes in appendix A.17 or a system error in case of failure (c.f. appendix A.16)
fvalue	float	the optional value associated with the error.
svalue	float	the optional string associated with the error.

5.2.2.4 Error_ResetErrors

Clears the internal list of errors made during the current ride. Only errors made after the function call will be displayed for evaluation.

Arguments: none

Return Values: none

5.2.2.5 Error_SetError

Programmatically tell the system about an error the driver has made.

Arguments:

error	integer	one of the error codes in appendix A.17
data	float, opt.	an optional data value (meaning depends on error)
message	string, opt.	optional string value (meaning depends on error)

Return Values: none

See also: **Error_SetErrorMode**, **Driver_SetCollision**

Remarks: Many mistakes the driver can make are automatically detected by the simulator. In some situations, it may be useful to tell the system about an error which otherwise would not have been noticed. The system reacts just like for an error it would have detected itself, e.g. by presenting a message to the driver, logging the error, etc.

Use **Error_SetErrorMode** in order to control the system's reaction for a given error.

5.2.2.6 Error_SetErrorMode

Adjust the system's reaction for particular error codes.

Arguments:

error	integer	one of the error codes in appendix A.17
mode	integer	how to react, see remarks

Return Values:

oldmode	integer	previous mode

Remarks: The system normally reacts in four ways to the occurrence of an error: display during the ride, summary screen afterwards, printed result list and trainer log files. You can selectively (de)activate some or all of these reactions. As *mode* pass the sum of the constants representing the ways to react (c.f. appendix A.18) or use one of the convenience constants **ERRMODE_IGNORE** and **ERRMODE_NORMAL**.

5.2.2.7 Error_SetTriggerValue

Adjust the trigger value used for particular error codes.

Arguments:	*error*	integer	one of the error codes in appendix A.17
	trigger	float	the new trigger value
Return Values:	*oldtrigger*	float	previous trigger value

Remarks: Some faults are detected, when a certain value exceeds (or falls below) some limit. These limits are adjustable for some error codes. Examples for adjustable limits are the speed (in km/h), above which aquaplaning can occur (**ERR_AQUAPLANING**), the minimal headway (in m) for **ERR_HEADWAY**, the minimal speed (in km/h) for **ERR_TOO_SLOW**, the TTC warning level (in s) for **ERR_TIME_TO_COLLISION** and the rpm limit for **ERR_REVOLUTIONS_HIGH** and **ERR_RPM_HIGH_WRONG_GEAR**.

5.2.3 Logging Functions

Logfiles can be used either for debugging purposes or to store selected information from the virtual world in order to later analyze details of the driver's performance. In order to include information from the scripting language, you have to turn on dynamic logging (**Logfile_DYNAMIC** in SIMUCONF.CFG).

5.2.3.1 Logfile_AddScriptVariable

Associates a numeric scripting language variable with a column in the dynamic log file.

Arguments:

col	integer	number of the column. 0 is the first column.
varname	string	name of the variable
name	string	name of the column (header in the log file)
precis	integer	precision of the output (see remarks)

Return Values: none

See also: Logfile_Reset

Remarks: This function tells the logging system to include a named numeric variable in the logfile output. The precis argument determines the precision as follows:

1: 4.0 digits "...0"
2: 5.1 digits "...0.0"
3: 6.2 digits "...0.00"
4: 7.3 digits "...0.000"

You should reset the logfile after having associated all variables you are interested in (using **Logfile_Reset(1)**). Otherwise, the logfile might start with garbage lines recorded while you still were in the process of associating variables.

5.2.3.2 Logfile_Reset

Reset properties of a logfile.

Arguments:

flags	integer	see remarks

Return Values: none

Remarks: Set *flags* to the sum of the constants associated to the actions you want the function to perform.

1: Start a new empty log file.
2: Clear all user variable associations.
4: Set column associations to factory default.

5.2.3.3 Logfile_SetRate

Set the rate of logging or trigger one-shot logging.

Arguments:

normal	integer	base logging rate in Hz, see remarks
event	integer, opt.	logging rate with running event, see remarks

Return Values: none

Remarks: The system knows two logging rates. Usually, logging is performed with the *normal* rate. At times, where events are running (e.g. deer from the left), logging is performed at the (usually higher) *event* rate.

Setting the rates to $-1, 0$ will completely turn off logging. Setting them to $0, 0$ records the momentary values and then stops logging.

The effect of calling this function only lasts for the duration of the current scenario.

5.2.4 String Functions

The functions in this section are used to manipulate strings.

5.2.4.1 String_GetLength

Returns the length of a string.

Arguments:	*input*	string	the input string
Return Values:	*length*	integer	the string's length in characters.
See also:	**String_Part**		

5.2.4.2 String_Part

Extract a substring.

Arguments:	*input*	string	the input string
Arguments:	*from*	integer	the index of the substring's first character, starting with 0.
Arguments:	*count*	integer, opt.	the maximum number of characters to extract. 0 or unspecified means all.
Return Values:	*result*	string	the resulting substring.
Remarks:	The valid ranges for *from* and *count* can be queried using **String_GetLength**.		
See also:	**String_GetLength**		

5.2.4.3 String_Transform

Manipulate strings.

Arguments:	*trafo*	integer	what to do with the input string, see appendix A.42
	input	string	the input string
	...	opt.	optional additional argument(s) depending on $trafo$
Return Values:	*result*	string	the resulting string

5.2.5 Surround Functions

The functions in this section are concerned with the driver's immediate surrounding inside the virtual world. They allow you to easily query and manipulate those objects of the virtual world, which have the most significance for the driver in the current situation.

5.2.5.1 Surround_GetHeadway

Get the distance to the next moving object infront of the driver (which might be another car, a deer, a pedestrian, . . .)

Arguments: none

Return Values: *dist* float distance in m

5.2.5.2 Surround_GetTimeToCollision

Get the time to collision with the next moving object infront of the driver if neither speed nor direction changes for either the moving object or the driver.

Arguments: none

Return Values: *ctime* float time to collision in s

5.2.5.3 Surround_SetEnvironmentSound

Switch environmental sound (e.g. motor sounds of surrounding vehicles) on or off.

Arguments: *sound* integer sound is switched on for *sound* = 1 and off for *sound* = 0.

Return Values: none

5.2.5.4 Surround_GetTimeToLineCrossing

Returns the time (in s) until the own vehicle will cross the line at the center of the road if current speed and direction do not change. If this value cannot be calculated, 0 is returned.

Arguments: none

Return Values: *time1* float time to center or right road border

time2 float time to next lane border

toleft int next line crossing to the left? (0/1)

5.2.5.5 Surround_GetDistanceToLineCrossing

Distance (in m) to the next lines. 9999.9 if not measurable.

Arguments: none

Return Values:

leftlane	float	distance to next lane border to the left.
leftroad	float	distance to road center line.
rightlane	float	distance to next lane border to the right.
rightroad	float	distance to road border.

5.2.5.6 Surround_GetRoadAhead

Query the absolute position of a point on the road before the driver in a given distance.

Arguments:

dist	float	distance from the driver in m

Return Values:

x	float	
y	float	
z	float	
angle	float	
curve	float	road curvature (in degrees per meter)

Remarks: The point the position of which is returned is the point exactly in the middle of the road in a distance of *dist* m before the driver.

x, y and z are the cartesian coordinates of the point and *angle* is the road's yaw angle as described for **Driver_GetAngles**.

5.2.5.7 Surround_GetRoadSpeed

Get the speed limit at the current position of the driver.

Arguments:

smooth	int., opt.	respect future limits and acceleration times?

Return Values:

slimit	float	speed limit in km/h

5.2.5.8 Surround_GetRoadType

Get the type of the road.

Arguments:

road	int., opt.	road to query, if 0: mypos.road

Return Values:

type	float	type of the road, combination of values from appendix A.7.13

5.2.5.9 Surround_SetExternOrientation

Set the current inclination of the world surface (**WORLD_EMPTY** only).

Arguments:

incline	float	inclination angle in degrees. > 0 means ascending.

Return Values: none

5.2.5.10 Surround_SetLight

Sets the phase of the next traffic light infront of the driver's vehicle and returns the distance to the traffic lights.

Arguments:

phase	integer	see appendix A.27 for a list of valid phases.
tlslot	integer	Adjust the phase for a specific traffic light slot by specifying $tlslot = slot + 1$.

Return Values:

dist	float	distance to the next traffic light, < 0 if none found in the driver's vicinity.
ophase	integer	previous state of the traffic light, c.f. appendix A.27.

Remarks: If you pass **TLST_NONE** for *phase*, the phase remains unchanged and only the distance is returned.

If you specify an explicit slot, no distance is returned.

5.2.5.11 Surround_SetLightDurations

Sets the phase durations of the next set of traffic lights infront of the driver's vehicle.

Arguments:

green	float	Duration of the green phase in seconds. For complex light signalling systems, this sets the duration of the green phase in the main traffic direction. Pass 0 to not set the green phase's duration and -1 to reset to the default value.
amber	float	Duration of the amber phase in seconds. Pass 0 to not set the amber phase's duration and -1 to reset to the default value.
tlslot	integer, opt.	Adjust the durations for the program controlling a specific traffic light slot by specifying $tlslot = slot + 1$.

Return Values: none

5.2.6 Symbol Functions

In addition to the standard feedback the simulator gives to the driver, you may choose to provide your own customized feedback using custom symbols. You can create up to 20 custom symbols by creating appropriate bitmaps and saving them in the folder GFX\INFOSIGNS. Use filenames INFOSIGN*n*.BMP, where *n* is a number in the range $0, \ldots, 19$.

You can then use these bitmaps from your code with the functions described in this section.

5.2.6.1 Symbol_Activation

(De)activate a given custom symbol (i.e. show or hide it).

Arguments:

n	integer	identifies the custom sign, $0 \leq no < 20$
active	integer	1 shows the symbol, 0 hides it.

Return Values: none

5.2.6.2 Symbol_SetBrightness

Set the brightness of a custom symbol.

Arguments:

no	integer	identifies the custom sign, $0 \leq no < 20$
bright	float	brightness in percent (i.e. from 0 to 100)

Return Values: none

See also: Symbol_Activation

5.2.6.3 Symbol_SetPosition

Set the position of a custom symbol on the screen.

Arguments:

no	integer	identifies the custom sign, $0 \leq no < 20$
posx	float	horizontal position in percent
posy	float	vertical position in percent

Return Values: none

See also: Symbol_Activation

Remarks: The position refers to the center of the custom sign. 0 means left (top) edge, 100 right (bottom) edge of the center screen.

For multiscreen systems, choosing *xpos* in the range $[-100, 0)$ positions the symbol on the left screen, while values in $(100, 200]$ position it on the right screen.

5.2.6.4 Symbol_SetSize

Set the size of a custom symbol by scaling it.

Arguments:

no	integer	identifies the custom sign, $0 \leq no < 20$
size	float	size in percent (i.e. 100 means native bitmap size)

Return Values: none

See also: Symbol_Activation

5.2.6.5 Symbol_SetTransparency

Set the transparency of a custom symbol, i.e. the degree, to which the landscape shines through.

Arguments:

no	integer	identifies the custom sign, $0 \leq no < 20$
trans	float	transparency in percent. 0 means solid, 100 invisible.

Return Values: none

See also: Symbol_Activation

5.2.7 System Functions

This section contains functions used to manipulate or query properties of the simulation as a whole. In addition, several helper functions are listed in this category.

5.2.7.1 System_ClearText

Removes any text previously displayed onscreen.

Arguments: none

Return Values: none

Remarks: This command also stops spoken messages.

5.2.7.2 System_EnhancedReality

Turns an enhanced reality feature on or off.

Arguments:	*function*	integer	Which feature to control, see appendix A.10.
	onoff	integer	Turn the feature on (1) or off (0). Specify (2) to query the current state. For some features, (3) activates them during replay after the ride, but not during the ride.
Return Values:	*wason*	integer	Previous activation state or error code.

5.2.7.3 System_EnhancedRealityParameter

Sets an enhanced reality parameter.

Arguments:	*function*	integer	Which feature to control, see appendix A.10.
	parameter	integer	Which parameter to set, see appendix A.10.1.
	...	variable	The new parameter value(s). Depends on parameter.
Return Values:	none		

5.2.7.4 System_GetControl

Retrieves the current button state of the control unit (i.e. allows you to use the control unit for custom purposes).

Arguments: none

Return Values: *status* integer see remarks

Remarks: The possible return value is a sum of:

0: no button pressed
1: **Line** button pressed
2: **Mode** button pressed
4: **Horn** button pressed
8: **Enter** button pressed
16: **Sideview right** button pressed
32: **Sideview left** button pressed
64: **Coin acceptor** activated
128: **User button 1** pressed
256: **User button 2** pressed
512: **User button 3** pressed
1024: **Reverse Line** button pressed
2048: **Reverse Mode** button pressed

Remarks: Note that **Line**, **Mode**, **Enter** and **Coin acceptor** are also recorded as error-messages in the logfile.

5.2.7.5 System_GetCountry

Retrieves the country setting (which is used for display purposes and varying traffic rules).

Arguments: none

Return Values: *country* integer see appendix A.3 for a list of possible return values.

5.2.7.6 System_GetLanguage

Retrieves the language setting (you should perhaps use this function before displaying text messages).

Arguments: none

Return Values: *language* integer see appendix A.2 for a list of possible return values.

5.2.7.7 System_GetRandom

Returns a pseudo-random number.

Arguments: none

Return Values: *rand* integer (almost) random number, $0 \leq rand < 100$.

5.2.7.8 System_GetSimulation

Return the full absolute position of the driver's vehicle.

Arguments: none

Return Values:

x	float	
y	float	
z	float	
pitch	float	
yaw	float	
roll	float	

Remarks: The results are equivalent to calling both **Driver_GetAbsolutePosition** and **Driver_GetAngles**.

5.2.7.9 System_GetStatus

Return the simulator's current program status.

Arguments: none

Return Values:

status	integer	one of the state constants in appendix A.35, possibly or'ed with flag constants

5.2.7.10 System_GetViewSize

Get the main screen size in pixels.

Arguments: none

Return Values:

x	integer	horizontal extent
y	integer	vertical extent
ratio	float	physical screen ratio

5.2.7.11 System_InitSampleFile

Loads a sound file for later use.

Arguments:

filename	string	path to the sound file. You can either use absolute path names or relative paths. The latter are relative to the simulator program installation directory. The predifined sound files are stored at the relative path SAMPLES.

Return Values:

id	integer	sound identifier, can be used instead of predefined sound constants (c.f. appendix A.26)

5.2.7.12 System_Menu

Show or hide the menu and lock/unlock menu display and driver-controlled start of ride.

Arguments:

vmask	integer	new values, see remarks
smask	integer	select values to be set, see remarks

Return Values:

omask	integer	old bit values, see remarks.

Remarks: The state and accessibility of the menu is controlled by a set of bits defined in appendix A.34.

vmask controls the new values of these bits, while *smask* controls, which of them are set at all. Let the old value be denoted by *omask*. The new value is then computed as follows:

$nmask = (omask\& \sim smask)|(vmask\&smask)$

5.2.7.13 System_Perspective

Sets a window's camera perspective.

Arguments:

window	integer	$1, 2, 3, \ldots$
perspective	integer	see appendix A.36

Return Values: none

Remarks: If *window* is a positive number > 0, it refers to a real output window of the local simulator process. Then, *perspective* can be positive (a camera according to appendix A.36 or negative, which assigns a standard functionality. Standard functionalities are 1 for front monitor, 2 for front left, ...

If *window* is a negative number < 0, it refers to a standard functionality, e.g. -1 for front monitor, -2 for front left, ... Then, *perspective* must be a positive camera identifier.

If *window* is $= 0$, *perspective* must be a positive camera identifier and is mapped to the front monitor. In addition, further viewports may be adjusted to match the front monitor.

See also: **System_SetTopViewArea**

5.2.7.14 System_PlaySample

Play a sound.

Arguments:

id	integer	sound identifier – either one of the constants in appendix A.26 or the return value of **System_InitSampleFile**.
vol	integer	volume of the sound, $0 \leq vol \leq 64$. If $vol < 0$, the sample will be stopped.
offset	float	offset in seconds from the beginning

Return Values: none

See also: **System_InitSampleFile**

5.2.7.15 System_Reset

Resets all simulator states in order to start a new simulation.

Arguments: none

Return Values: none

5.2.7.16 System_Scenario

Select a given scenario without interfering with the currently running ride. Changes are instantaneously visible in the menu. May be used to set the next scenario. Scenario selection is realized by selecting the custom scenario with the given name (which, of course, must be known to the simulator).

Arguments:

scenario	string	path of the scenario to be selected (e.g. ./SCENARIOS/TRAINER/DEMO2.SCE)
level	integer,opt.	level to be selected (if multilevel), 0 means default.
sublevel	integer,opt.	sublevel to be selected (if multilevel), 0 means default.

...

Return Values: none

Remarks: Using this function is discouraged, because the two step starting process suffers from a race condition. Use **System_Start** with mode 1 or 2 instead.

5.2.7.17 System_GetSubVector

Returns a specified part of the argument vector.

Arguments:

startidx	integer	The index of the first element to copy.
size	integer	The number of elements to copy, negative size means copy everything after startidx.
orig	vector	The source vector.

Return Values:

subvec	vector	The resulting vector.

5.2.7.18 System_SetSubVector

Inserts the specified number of elements into the vector.

size	integer	The number of elements to insert.
position	integers	Position where to insert the elements.
values ...	integer	The elements to insert.
orig	vector	The source vector.

Return Values:

subvec	vector	The resulting vector.

5.2.7.19 System_CutSubVector

Cuts the specified number of elements from the vector.

size	integer	The number of elements to cut out.
position	integers	Position where to cut the elements.
orig	vector	The source vector.

Return Values:

subvec	vector	The resulting vector.

5.2.7.20 System_SetImage

Display a user defined image.

Arguments:

imageno	integer	see remarks
xpix	integer, opt.	horizontal offset in pixels
ypix	integer, opt.	vertical offset in pixels
flags	integer, opt.	set to 0, see appendix A.38
bslot	integer, opt.	set to 0, see appendix A.39
width	integer, opt.	width in pixels
height	integer, opt.	height in pixels
trans	float, opt.	transparency 0...1

Return Values: none

See also: System_SetImageFile, System_GetViewSize

Remarks: The user defined bitmap is shown at a position defined by *xpix* and *ypix*. *xpix* and *ypix* define (in pixel units) an offset from an absolutely centered position. Thus, omitting these two arguments shows the bitmap at a centered position (on the center screen). The image to be shown must be located in the subfolder BILDER of the program installation folder. The filename must be B*number*.BMP, where $number = imageno + 2000$. Passing 0 for *imageno* removes any previously shown bitmap.

Attention: the bitmap must be accessible by the computer showing the image. When using the Control Center, for instance, this is not necessarily the same computer as the one running the simulation.

The following example code fragment displays the 40x40 bitmap B2005.BMP at the lower right corner of the screen.

Listing 5.10: System_SetImage (Example Usage)

```
VAR sizex 0
VAR sizey 0
VAR posx 0
VAR posy 0

FUNC sizex, sizey = System_GetViewSize ()

CALC posx = sizex / 2
CALC posx = posx - 20

CALC posy = sizey / 2
CALC posy = posy - 20

FUNC System_SetImage (5, posx, posy)
```

5.2.7.21 System_SetImageFile

Display a user defined image.

Arguments:

filename	string	see remarks
xpix	integer, opt.	horizontal offset in pixels
ypix	integer, opt.	vertical offset in pixels
flags	integer, opt.	set to 0, see appendix A.38
bslot	integer, opt.	set to 0, see appendix A.39
width	integer, opt.	width in pixels
height	integer, opt.	height in pixels
trans	float, opt.	transparency 0...1

Return Values: none

See also: **System_SetImage**, **System_GetViewSize**

Remarks: This function works pretty much the same as **System_SetImage**. The main difference is, that the file is referenced by filename rather than index. *filename* should be either an absolute path or start with ./BILDER/ to refer to files located in the subfolder BILDER. The Extension may be omitted. Searching is performed including the current country and language setting.

The image file for this function can be PNG, BMP, JPG, TGA or PCX. The color depth may be 15, 16, 24 or 32 bits. Paletted images (e.g. color depth 8 or less) are not supported. The formats PNG and TGA may contain an alpha channel.

5.2.7.22 System_SetImageText

Use this function to graphically display formatted text using system fonts on top of or mixed with other graphics.

Arguments:

text	string	the text to display
font	string	font name, e.g. Arial
size	int	font size (height) in pixel
r	float	red color component 0...1
g	float	green color component 0...1
b	float	blue color component 0...1
t	float	transparency 0...1
xpix	integer, opt.	horizontal offset in pixels
ypix	integer, opt.	vertical offset in pixels
flags	integer, opt.	bitmask from SETIMG_..., see appendix A.38
bslot	integer, opt.	set to 0, see appendix A.39
width	integer, opt.	width in pixels, not working yet
height	integer, opt.	height in pixels, not working yet

Return Values: none

5.2.7.23 System_SetReport

Use this function to display formatted text in the style of the evaluation screen.

Arguments:

mode	int	the type of text to display, see appendix A.33
linecount	int	number of lines in the array
text	vector	a vector with descriptions for each line of the report menu

Return Values: none

See also: **System_SetText**, **System_GetReport**

Remarks: The vector describes each line of the report display. Each line contains four values in the vector: The textstring, a flag value (see appendix A.32), an indent value and a reserved value. The report display is displayed immediately. To clear the display, a linecount of 0 must be set. Note that a visible display of another mode has to be removed before a new mode display can be issued.

5.2.7.24 System_GetReport

Use this function to get the current set of text lines of the report/evaluation screen.

Arguments:

mode	int	the type of text to fetch, see appendix A.33

Return Values:

count	integer	number of lines in the returned vector
lines	vector	a vector containing the lines of the report menu

See also: **System_SetReport**

Remarks: The returned vector describes each line of the report display. Each line contains four values in the vector: The textstring, a flag value (see appendix A.32), an indent value and a reserved value. Note that if the current displayed menu is of another mode than the requested one, the result will be empty.

5.2.7.25 System_Screenshot

Trigger a screenshot.

Arguments: none

Return Values:

error	integer	An error code.
basepath	string	The screenshot is saved in one or more files with paths created by appending to this.

5.2.7.26 System_SetAlcoholValue

Apply an alcohol effect to the simulation system.

Arguments:

promille	integer	Alcohol promille*10. For an effect of 0.8%. a value of 8 ist used.
compare	integer, opt.	Alcohol promille*10 for comparison purposes. Useful for evaluation screens when $promille = 0$.

Return Values: none

Remarks: This function allows the alcohol effect to be set directly with the same effects known from other alcohol simulation programs. A value of 0 can be used to switch off the alcohol effect, and a value of -1 can be used to switch to the original menu value.

5.2.7.27 System_ToggleAlcoholValue

Apply an alcohol effect to the simulation system or turn it off again.

Arguments:

promille	integer	Alcohol fallback promille*10. See remarks below.

Return Values: none

Remarks: If different alcohol and comparison values have been set using **System_SetAlcoholValue**, the two values can be exchanged using this function. Usually, this has the effect of turning the alcohol effect on or off. If no comparison value has been set, 0 (if currently on) or $promille$ (if currently on) is used instead. If the alcohol effect is currently off and no positive $promille$ value is specified, the function fails.

5.2.7.28 System_SetBlurEffect

Apply a blurred effect to the visual system.

Arguments:

perc	integer	percentage of visual impediment.

Return Values:

oldperc	integer	value before the function call.

Remarks: This function is meant to simulate a driver condition unfit for driving (e.g. after consumption of alcohol). This is achieved by combining a grey veil reducing the picture contrast with a true blurring effect. The latter is only possible using special hardware (implemented by beamer defocussing).

!Not implemented under F12! Currently, perc is effectively a boolean value – everything below 50% means no impediment, everything above full impediment.

5.2.7.29 System_SetCurrentPos

Sets an internal variable to a given road position.

Arguments:

roadno	integer	
dist	float	
angle	float	
track	float	

Return Values: none

Remarks: The arguments are just the components of a road position vector.

The "current position" set by this function has nothing to do with the current position of the driver or any other vehicle. Instead, it is used as an anchor or default position for other functions like **Traffic_SetRoute**.

5.2.7.30 System_SetDisplayOptions

Set various visual aspects of the simulation.

Arguments:

newval	integer	bit mask with new values
mask	integer, opt.	bit mask determining what is changed

Return Values:

oldval	integer	old values

Remarks: Only bits which are set in mask are changed at all. Hence, the default value of 0 for *mask* turns the function into a pure read function. The currently defined bits for this function are:

1: Show instrument board. The default depends on configuration settings.

2: Error messages. On by default. Globally turns on/off all messages about traffic rule violations during the ride (i.e. for all kinds of errors).

4: Show result table after the ride. On by default.

5.2.7.31 System_SetText

Display a short text message.

Arguments:

message	string	see remarks

Return Values: none

Remarks: This function displays a short text message at the top of the center screen. Passing an empty string clears a previously set message.

5.2.7.32 System_SetTopViewArea

Sets the visible area for a top view camera.

Arguments:	*cx*	float	x-extent of the view (in m world coordinates)
Return Values:	none		
See also:	**System_Perspective** (**CAMERA_TOP_VIEW**)		

5.2.7.33 System_SetTopViewMode

Sets the (auto-)position for a top view camera.

Arguments:	*mode*	integer	positioning mode, see appendix A.36.1
	x	float	x-coordinate (meaning depends on mode)
	y	float	y-coordinate (meaning depends on mode)
Return Values:	none		
See also:	**System_Perspective** (**CAMERA_TOP_VIEW**)		

5.2.7.34 System_GetInfo

Returns various parameter of the system.

Arguments:	*infotype*	integer	type of information, see remarks
	arg	float	optional argument (meaning depends on *infotype*)
Return Values:	*val*	float, opt.	depends on *infotype*
Remarks:	The possible information types are listed in appendix A.40.		

5.2.7.35 System_Math

Calculate results of mathematical functions.

Arguments:	*func*	integer	the function to calculate, see appendix A.41
	x1	float	first argument of the function
	x2	float, opt.	second argument (meaning depends on *func*)
	...	float, opt.	further arguments (meaning depends on *func*)
Return Values:	*result*	float	calculation result
Remarks:	This function only returns scalar values. Use **System_VectorMath** for vectorized operations.		

5.2.7.36 System_VectorMath

Calculate results of mathematical functions.

Arguments:	*func*	integer	the function to calculate, see appendix A.41
	x1	float	first argument of the function
	x2	float, opt.	second argument (meaning depends on func)
	...	float, opt.	further arguments (meaning depends on func)
Return Values:	*result*	float	calculation result
	...	float	opt. further result(s), depending on func and arguments

5.2.7.37 System_DataFile

Access data files.

Arguments:	*op*	integer	Operation, see appendix A.43
	spec	string	The file or search mask, searching and deleting support wildcards. Files are always specified without their .sdf file suffix and paths are relative to the Scenario/data folder.
	data1	any, opt.	For write operations, the data to write. For search, specify 1 to search recursively.
	...		Further data to write.
Return Values:	*err*	int	An error code, see appendix A.16. For search, positive values specify the number of found files.
	indata	vector	For read, the read in data vector. For search, a list of file names matching the search mask.

5.2.7.38 System_GetUserData

Return information items associated with the current driver.

Arguments:	*itemid*	integer	type of information, see appendix A.31
Return Values:	*value*	string	the item's value

5.2.7.39 System_SetUserData

Set various information items associated with the current user.

Arguments:	*value*	string	value to set
	valtype	integer	type of information, see appendix A.31
Return Values:	none		
See also:	**System_GetUserData**		

5.2.7.40 System_GetUserVariable

Get a scripting language variable associated to a user (by variable name).

Arguments:	*varname*	string	The variable to query
Return Values:	*status*	integer	SYSERR_OK if successful, error code else.
	values	vararg	The actual variable vector.
See also:	**System_SetUserVariable**, **System_ClearUserVariable**		

5.2.7.41 System_SetUserVariable

Associate a scripting language variable vector to a user (for named access).

Arguments:	*varname*	string	The variable name
	values	vararg	The actual variable vector.
Return Values:	none		
See also:	**System_GetUserVariable**, **System_ClearUserVariable**		

5.2.7.42 System_ClearUserVariable

Delete a named scripting variable from a user.

Arguments:	*varname*	string	The variable name
Return Values:	none		
See also:	**System_GetUserVariable**,**System_SetUserVariable**		

5.2.7.43 System_Shutdown

Programmatically stop the simulator program and shut down the computer.

Arguments:	*mode*	integer	see remarks
Return Values:	none		
Remarks:	If *mode* is 0, the function shuts down the main simulator computer. If *mode* is 1, possible additional computers in a multi-computer simulator are also shut down. If *mode* is 2, the simulator program is ended without shutting down the computer.		

5.2.7.44 System_Start

Start a previously selected scenario.

Arguments: *mode* integer, opt. see remarks

Arguments: none

Return Values: none

Remarks: Sending this command via a network connection with the default $mode = 0$ has the same effect as pressing the ⟨ENTER⟩ key. In particular, the motor is not started. The scenario is run in the "remotely controlled" modus, which lasts until the end of the scenario.

With $mode = 1$ the arguments following $mode$ are interpreted just like the arguments for **System_Scenario**, making the sequence **System_Scenario**, **System_Start** atomic. This solves some race conditions occuring with network transport.

$mode = 2$ is similar to $mode = 1$, but the path and level vector are URL-encoded and passed as a single string in the form $path?level1\&level2\& \ldots$

See also: **System_Scenario**

5.2.7.45 System_InitPhase

Start a previously selected scenario.

Arguments: *on* integer see remarks

Arguments: none

Return Values: none

Remarks: A scenario should call this with $on = 1$ as the first thing after starting. When it has completed its initialization, it then must call this with $on = 0$ (otherwise the screen will stay black and the system appears to hang).

5.2.7.46 System_Stop

Stop execution of a scenario.

Arguments: *notme* integer, opt. If this is set to 1 and the calling scenario is privileged, it is not stopped.

Return Values: none

Remarks: This function should be executed at the end of any scenario. Note, that a calling scenario keeps running until the next SLEEP or until it is preempted.

5.2.8 Traffic Functions

The functions in this section are mainly used to query and manipulate foreign vehicles and other objects controlled by the simulation (i.e. not the driver's vehicle). Nevertheless, many of them can also be used on the driver's vehicle. Objects are referenced using their object ID. Valid object IDs are positive, nonzero integers. For many functions, this provides a means to easily test the function's result for success. The object ID of the driver's vehicle can be obtained using **Driver_GetID**.

5.2.8.1 Traffic_GetAbsoluteDistance

Get the distance between two objects as the crow flies.

Arguments:	*id1*	integer	first object identifier
	id2	integer	second object identifier
Return Values:	*dist*	float	distance in m

See also: **Traffic_GetDistAndVelocity**

5.2.8.2 Traffic_GetAbsolutePosition

Get an object's position in cartesian world coordinates. The coordinate system is left-handed and y is the upwards direction. The z-direction will at times be called "north". All coordinates are measured in m. When heading north, x is to the right.

Arguments:	*id*	integer	object identifier
	idref	integer, opt.	reference object identifier
	useoff	bool, opt.	if set to 1, the position offset will be taken into account
Return Values:	*x*	float	Cartesian x-coordinate in m
	y	float	Cartesian y-coordinate in m
	z	float	Cartesian z-coordinate in m
	yaw	float	Angle about the y-axis in degree
	pitch	float	Angle about the x-axis in degree
	roll	float	Angle about the z-axis in degree

Remarks: If a nonzero *idref* parameter is specified, the coordinates are returned relative to the corresponding object. In this case, "north" means the reference object's forward direction.

See also: **Driver_GetAbsolutePosition**, **Traffic_GetAngles**

5.2.8.3 Traffic_GetAngles

Obtain the rotational orientation of an object relative to the x, y, and z axes. Angles are measured in degree in the range $(-180, 180]$.

Arguments:	*id*	integer	object identifier
Return Values:	*pitch*	float	Angle about the x-axis
	yaw	float	Angle about the y-axis
	roll	float	Angle about the z-axis

See also: **Driver_GetAngles**, **Traffic_GetAbsolutePosition**

Remarks: For a pictorial illustration of the angles' meaning refer to figure 5.1 on page 59.

Note that this function uses a non-typical order of angles, while others use yaw, pitch, roll; like **Traffic_GetAbsolutePosition**.

5.2.8.4 Traffic_GetDistAndVelocity

Returns the distance between an object and the driver's vehicle in m measured along the shortest road connection between the two (i.e. disregarding traffic rules). Also returns the object's current speed in km/h.

Arguments:	*id*	integer	object identifier
Return Values:	*dist*	float	distance from driver's vehicle
	speed	float	speed

See also: **Traffic_GetAbsoluteDistance**

5.2.8.5 Traffic_GetInfo

Returns various properties of an object.

Arguments:	*id*	integer	object identifier, or 0 for the driver
	infotype	integer	type of information, see remarks
	infoparm	float, opt.	some values of *infotype* take an additional parameter.
Return Values:	*val*	float	depends on *infotype*
	val2	float	optional, depends on *infotype*
	...		...

Remarks: The possible information types are listed in appendix A.6. For information types returning an object id, a return value 0 means, that no object of the requested type is in reach.

5.2.8.6 Traffic_GetPosition

Retrieve an object's position.

Arguments: *id* integer object identifier

Return Values: *roadno* integer

dist float

angle float

track float

Remarks: The result values are the components of a road position vector. For the driver's vehicle, you can use the global variable **mypos** instead.

5.2.8.7 Traffic_GetRouteStatus

Find out, whether an object is on a preset route.

Arguments: *id* integer object identifier

Return Values: *status* integer 1 if the object is on a preset route, 0 else

Remarks: Routes can be set using **Traffic_SetRoute**. Foreign vehicles follow a preset route and this function returns 1 if a route has been set and has not yet been completed. Applied to the driver's vehicle, you can test with this function whether the driver has deviated from his assigned route.

5.2.8.8 Traffic_InsertCarAtNextJunction

This inserts a car (randomized type) at the next junction.

Arguments: *direction* integer see remarks

Return Values: *dist* float see remarks

Remarks: The car will come either from the left (*direction* = 1), from the right (*direction* = 2) or from a random direction (*direction* = 3). If the insertion succeeds, the return value is the distance in m of the insertion point from the driver's current position. If it fails, the return value is a negative error code.

As the maps used for scenarios become more advanced, this function looses its advantages. We do not encourage its use.

5.2.8.9 Traffic_InsertVehicle

This inserts a moving object into the virtual world.

Arguments:

type	integer	type of object, c.f. appendix A.4
offset	float	see remarks
flags	integer	bitmask, see remarks
roadno	integer	
dist	float	
angle	float	
track	float	
world	integer,opt.	world id, ignored.

Return Values:

id	integer	object identifier of the inserted object (≤ 0 means failure)

Remarks:

This function inserts a moving object of type *type* into the world. Its position and direction are determined by *offset*, *flags* and the road position passed in the remaining arguments. Let us call this position *pos*. If *roadno*, ... are omitted or *flags* includes **INSFLAG_ARGISROUTE**, *pos* is taken to be the driver's current position instead.

If all vehicles of the specified *type* are already inserted into the world, the vehicle which is the farthest from the driver, but at least 500m away, will be removed and inserted at the specified position. If no vehicle can be found there will be an error.

By using **OBJECT_POINTER** as *type* it is possible to insert a specific vehicle. Then the limitation of 500m is reduced to 50m when all specific vehicles are already in the world.

The default behaviour (*flags* = **INSFLAG_DEFAULT**) is to insert the object at a position determined by *pos*, translated by *offset* in the direction determined by *pos***.angle**. This introduces an ambiguity if a crossroads is encountered at a shorter distance in this direction.

The insertion position is then determined by a random process. In the future, you will be able to influence this behaviour by specifying the **INSFLAG_LEFT**, **INSFLAG_RIGHT** or **INSFLAG_STRAIGHT** flags (*not implemented yet*).

If you specify the **INSFLAG_USEROUTE** flag, the newly inserted vehicle is assigned a route to the current system position (see **System_SetCurrentPos**) and crossroads decisions are done with the help of this route (*not implemented yet*). If *flags* includes **INSFLAG_ARGISROUTE**, the route is determined by the additional arguments instead of the current system position.

Specifying **INSFLAG_ONCOMING** results in an object driving in the opposite direction as specified in *pos*.

Normally, the object is inserted with default track and angle modified to be either straight on or straight back. The values passed are used verbatim if you specify the flags **INSFLAG_USETRACK** and **INSFLAG_USEANGLE**, respectively.

If **INSFLAG_USECLEARANCE** is specified, *offset* is not used to translate the specified position, but as a minimal clearance enforced for the new vehicle (instead of the default $10cm$).

If **INSFLAG_XRELATIVE** is specified, the distance to the next crossroads is added to *offset* before inserting the object (*not implemented*).

5.2.8.10 Traffic_ObjectExists

Test an object whether it (still) exists inside the virtual world.

Arguments:	*id*	integer	object identifier
Return Values:	*exist*	integer	1 if the object exists, 0 else

5.2.8.11 Traffic_Remove

This function removes objects from the virtual world.

Arguments:	*flags*	integer	what to remove
	id	integer	object identifier
Return Values:	none		

Remarks: You cannot remove the driver's vehicle. The flags parameter is a bitmask, i.e. the sum of constants determining what to remove:

- 1: remove all foreign vehicles
- 2: remove all vehicles except for those infront of and near to the driver
- 4: remove all animals and pedestrians
- 8: remove all obstacles (animals, pedestrians, parked cars, trees, pillars)
- 16: remove the object identified by *id*. If this flag is not set, *id* is ignored.

5.2.8.12 Traffic_ResetSigns

This function resets traffic signs to a defined state.

Arguments:

mode integer

If 0, all traffic signs are removed from the world, but some standard collosion statics like pillars can remain. (This is also the default setting when changing the world.)

If 1, the initial state (defined by the map) is restored.

If 2, additional signs and traffic lights are added as suitable, for WORLD_MOUNTAIN, WORLD_TRAINER, WORLD_SMALLTOWN, WORLD_TOWN.

If 3, an additional driver start position is set in WORLD_MOTORWAY, WORLD_TRAINER, WORLD_SMALLTOWN, WORLD_TOWN.

If -1, all static objects are removed.

You can also specify an explicit static object identifier, in which case that specific object is removed.

Return Values: none

Remarks: You should not use *mode* 1,2 or 3, because their behaviour is only well defined for the above listed worlds. Even there, the funcionality may be removed in the future.

Note, that explicitly specifying an object id does not conflict with other *mode* settings, because valid static object identifiers are always larger than all vehicle identifiers.

5.2.8.13 Traffic_SetEvent

Create an object, which will act in such a way as to create a particular (usually dangerous) traffic situation.

Arguments:	*type*	integer	event type, see appendix A.11
	dist_create	float	distance in m from driver's vehicle, where the event will occur
	dist_start	float	distance in m from driver to event, at which the event will be triggered (i.e. the event will happen this far from the driver).
	dist_mode	integer, opt.	instead of using *dist_start*, one can base the event distance on the current speed. For details c.f. appendix A.11.1
	options	integer, opt.	option bitmask, depends on the type of event (c.f. appendix A.11.2). Many events don't support any flags here.
	opt_arg	any, opt.	optional float, integer, bit mask or string, depends on the type of event.
	abspos	vector, opt.	optional complete roadposition to specify the absolute position of the event-obstacle. Is increased by *dist_create*.
Return Values:	*status*	integer	error code for the operation, SYSERR_OK on success.

Remarks: Obviously, *dist_create* must be greater than *dist_start* (otherwise the event would start behind the driver). If you base the distance on speed, specify 0 for *dist_start*. If you base the distance on reaction time, use *dist_start* for the reaction time between 0.01 and 4.0 Seconds.

During events, special logging and reaction time measurements are performed. Note, that evaluation error codes such as **ERR_REACTION** must be activated for this to have any effect. This is *not* done by this function, c.f. **Error_SetErrorMode** instead.

5.2.8.14 Traffic_GetEventName

Get a textual representation of a numbered event type.

Arguments:	*type*	integer	event type, see appendix A.11
	dflt	integer	if nonzero, the function returns the default name of the event (which might have been overridden when the event has been triggered).
Return Values:	*name*	string	result, empty string for invalid *type*.

See also: **Traffic_SetEvent**

5.2.8.15 Traffic_SetLights

Switch on or off lights of a given vehicle.

Arguments:	*id*	integer	object identifier
	action	integer	action to perform, see appendix A.19 for a list of valid actions.

Return Values: none

5.2.8.16 Traffic_SetAI

You can control several aspects of a simulated driver using this function. Apart from setting various aspects of the driver's general behaviour you can start (and stop) AI (artificial intelligence) programs according to which the driver behaves.

Arguments:	*id*	integer	object identifier
Arguments:	*ai_aspect*	integer	behaviour / program to start. Negative values stop the behaviour/program.
Arguments:	...	float, opt.	AI programs may have parameters, which can be passed as additional arguments.

Return Values: none

Remarks: For a list of behaviours and AI programs refer to appendix A.25. You can set multiple behaviour settings at once using **Traffic_SetBehaviour**.

5.2.8.17 Traffic_SetBehaviour

The behaviour of simulated drivers is determined by a set of rules. Normally, they follow all rules internally defined. This makes them appear (almost) clever, i.e. not provoking dangerous situations, but otherwise trying to reach their destinations as fast as allowed and possible. Using this function, you can turn off certain behaviour rules or turn them on again.

Arguments:	*id*	integer	object identifier
	behave_remove	integer	behaviour rules not to obey anymore
	behave_restore	integer, opt.	behaviour rules to reactivate

Return Values: none

Remarks: Both *behave_remove* and *behave_restore* are bitmasks, i.e. you can specify multiple behaviour rules by adding the values. For a list of behaviour flags refer to appendix A.24.

5.2.8.18 Traffic_SetAbsolutePosition

Sets the object's world position.

Arguments:

id	integer	object identifier
x	float	cartesian coordinates in m
y	float	
z	float	
yaw	float, opt.	
pitch	float, opt.	
roll	float, opt.	

Return Values: none

See also: **Traffic_GetAbsolutePosition**

Remarks: Use with caution if at all. Conversion of road parameters into cartesian parameters is a fast operation, but the reverse is not. In addition, the function easily fails if you specify invalid coordinates. Not specifying the orientation for not sperically symmetric objects causes unspecified behaviour.

5.2.8.19 Traffic_SetProfile

Set the profile for the traffic controller that is responsible for the objects behaviour.

Arguments:

id	integer	object identifier or 0 to set for all.
profiletype	integer	profile type to activate

Return Values: none

Remarks: For a list of *profiletype*s refer to appendix A.23. Note that the profile is reset for all vehicles at the start of a new ride.

5.2.8.20 Traffic_SetRoute

Set the destination of a given object. For details about route calculations and differences between routes for the driver and foreign objects see remarks.

Arguments:

id	integer	object identifier
flags	integer	flags, see remarks below
road	integer, opt.	see remarks below
dist	float, opt.	
angle	float, opt.	
track	float, opt.	

Return Values: none

Remarks: The parameters *road*, *dist*, *angle* and *track* (if specified) determine the destination of the object identified by *id*. If they are omitted, the values previously set with **System_SetCurrentPos** are used instead. Internally, a route from the object's current position to this destination is calculated and stored with the object.

Foreign vehicles follow this route and show a behaviour determined by *flags* after having reached their destination. The *flags* value is a bitwise OR of the constants in appendix A.28. For the driver, a navigation system is activated, which guides the driver along the calculated route.

The route from current to destination positions (both directed) is calculated to be the shortest possible route without 180° turns. Usually, traffic rules are taken into account for this calculation. This behaviour can be overridden by specifying the **ENDROUTE_IGNORE_SIGNS** flag.

See also: **System_SetCurrentPos**, **Driver_GetID**

5.2.8.21 Traffic_SetSigns

Insert traffic signs and other static objects into the virtual world at a given (directed) position.

Arguments:

type	integer	Static object type (c.f. appendix A.7)
value	integer, opt.	additional value, see remarks below
road	integer, opt.	see remarks below
dist	float, opt.	
angle	float, opt.	
track	float, opt.	
world	integer, opt.	c.f. appendix A.1, currently ignored.
offx	float, opt.	cartesian offset to the right.
offy	float, opt.	cartesian offset in upwards direction.
offz	float, opt.	cartesian offset in forwards direction.
offyaw	float, opt.	rotational offset to the right.
offpitch	float, opt.	rotational offset nose down.
offroll	float, opt.	rotational offset tilt left.

Return Values:

objid	integer	A strictly positive object identifier, if a single object was inserted. Otherwise, a negative error code is returned.

Return Values: none

See also: Traffic_ResetSigns

Remarks: If road is specified to contain a valid road number (> 0), *road*, *dist*, *angle* and *track* are taken to be the position of insertion. Otherwise, the point of insertion is determined by the current driver position. *dist*, *angle* and *track* are then taken to be relative to the driver position. Setting *dist* to 0 (the default value) then is equivalent to specifying *dist* = 200.

For some kinds of static objects, additional information must be passed in the *value* parameter. E.g. for speed limit signs, the limit (in km/h) is specified. For traffic lights, either 0 or 1 is passed to specify the relative phase (normal or inverted).

Some signs change the current world's net of legal paths as used by e.g. Traffic_SetRoute. Normally, such changes are taken into account as soon as the sign is inserted. This requires a lengthy calculation immediately following the sign insertion. If you insert multiple signs, this can noticeably slow down the flow of execution. You can circumvent the problem by first inserting the special sign **STATIC_BEGIN_TRANSACTION**, which turns off automatic net recalculation. After you have inserted all your signs, don't forget To insert the sign **STATIC_END_TRANSACTION**, which does the necessary adjustments to the street net and turns automatic mode back on again.

Some signs like **STATIC_STOP_LINE** are automatically centered on the lane, the lane value only needs to be roughly correct for the desired track. They are also properly sized to the width of the driving lane. They may me sized differently by adding the desired width to the value-parameter. This value must be the width in cm between 280cm and 590cm. It's also possible to apply an offset to the coordinates in order to move them away from their pre-calculated road/lane-center.

5.2.8.22 Traffic_SetSignProperty

Sets a property for a certain kind of traffic signs.

Arguments:	*type*	integer	Static object type (c.f. appendix A.7)
	value	integer	additional value, as in **Traffic_SetSigns**
	prop	integer	property type (c.f. appendix A.7.10)
	...	any, opt.	depending on type, value and prop
Return Values:	none		

5.2.8.23 Traffic_SetSpeed

Set and enforce the speed of an object.

Arguments:

id	integer	object identifier
speed	float	desired speed in km/h
mode	integer	how to set the speed, see remarks below

Return Values: none

See also: TrafficSetAcceleration

Remarks: The mode argument determines, how the speed is set. Valid values are:

SETSPEED_WISH: Set the desired speed of an object. The object's current speed is not altered, but it will try to reach the specified speed by accelerating or decelerating. The object does not obey traffic rules or speed signs. The behaviour can be normalized by setting a speed less than 0.

SETSPEED_IMMEDIATE: Set the forced speed and immediately change the object's speed to this value (unconditional!). The object does not obey traffic rules or speed signs. The behaviour can be normalized by setting a speed lower 0. A speed of 0 on objects outside the road put them into parking mode. Please note, that setting a high speed for a previously standing vehicle may have strange side effects, e.g. when the vehicle is not aligned with the road or has a large steering angle due to a planned lane change. If you really want to build a catapult, be sure to take accurate aim.

SETSPEED_WISH_RELATIVE: Same as **SETSPEED_WISH**, but speed is added to the current speed to obtain the desired speed. Use mode **SETSPEED_WISH** or **SETSPEED_IMMEDIATE** to reset the behaviour.

SETSPEED_IMMEDIATE_RELATIVE: Same as **SETSPEED_IMMEDIATE**, but speed is added to the current speed to obtain the desired speed. Use mode **SETSPEED_WISH** or **SETSPEED_IMMEDIATE** to reset the behaviour.

Add **SETSPEED_FLAG_OBEY_LIMITS** to above *mode* in order to obey speed limits.

Add **SETSPEED_FLAG_IMMEDIATE_LOAD_CHANGE** to above *mode* in order to allow immediate acceleration change (otherwise the vehicle may not react instantly). The downside is, that this may cause braking light flicker. Consider using **Traffic_SetAI** with **AI_REACTIONTIME** instead.

Add **SETSPEED_FLAG_FORCE_PARKING** to above *mode* **SETSPEED_IMMEDIATE** to enable parking even on the road. This is primarily for performance reasons with lots of cars just waiting in the wings: parking cars ignore everything around them saving precious computation time.

See appendix A.8 for numerical values.

5.2.8.24 Traffic_SetAcceleration

Sets the acceleration and deceleration used by a computer controlled object to change speed.

Arguments:	*id*	integer	object identifier
	decel	float	acceleration used for braking in m/s^2
	accel	float	acceleration used for speeding up in m/s^2
Return Values:	none		

5.2.8.25 Traffic_SetDistance

Determine the distance a simulated driver will keep to the vehicle before him.

Arguments:	*id*	integer	object identifier.
	factor	float	see remarks
	minimum	float	minimal distance in m, see remarks
Return Values:	none		
Remarks:	The standard distance a vehicle keeps quadratically depends on the current speed. You can modify this distance using *factor* (0.5 is the default value). Independently, a vehicle tries never to drive closer than *minimum* to its predecessor.		

5.2.8.26 Traffic_GetPhysicalParameter

Queries physical parameters of a vehicle.

Arguments:	*id*	integer	object identifier.
	parm	integer	which parameter to query, c.f. appendix A.37
Return Values:	*value*	float	If the vehicle supports this parameter, the function returns its value. Otherwise, it returns a (negative) error code.

5.2.8.27 Traffic_SetPhysicalParameter

Sets a physical parameter of a vehicle.

Arguments:	*id*	integer	object identifier.
	parm	integer	which parameter to set, c.f. appendix A.37
	value	float	The new value.
Return Values:	none		

5.2.8.28 Traffic_SetDoor

Open and close the door of a parking vehicle.

Arguments:

id	integer	object identifier.
flag	integer	1 opens the door, 0 closes it.

Return Values: none

Remarks: Currently, this function only works if *id* refers to an object of type **OBJECT_PARK_CAR2**.

5.2.8.29 Traffic_SetLeaveParking

Causes a parking vehicle to leave its gap and enter the road.

Arguments:

id	integer	object identifier
accel	float	initial acceleration in m/s^2

Return Values: none

Remarks: If *accel* is set to 0, a default value is used.

5.2.8.30 Traffic_SetPedestrian

Causes an object (for which this makes sense) to walk or stand.

Arguments:

id	integer	object identifier
flag	integer	1 makes the object walk, 0 makes it stand still.

Return Values: none

5.2.8.31 Traffic_SetSpeedLevel

Set the speed level of all foreign vehicles.

Arguments:

reserved	integer	must be 0
level	float	see remarks below

Return Values: none

See also: Traffic_SetSpeed

Remarks: Normally, all vehicles try to drive as fast as allowed, but not faster than desired. This behaviour can be modified using this function: The current speed limit is multiplied by *level* to obtain the limit meantioned above. Hence $level = 1$ yields the default behaviour, $level > 1$ lets all foreign vehicles exceed speed limits and $level < 1$ lets them drive overcautiously.

5.2.8.32 Traffic_SetTrackDeviation

Set track deviation (i.e. the **.track** component of the position).

Arguments:

id	integer	object identifier
track	float	desired track deviation in m
mode	integer	see remarks below
stretch	float, opt.	a route change will be stretched over at least this distance in m.

Return Values: none

Remarks: If mode is **SETTRACK_IMMEDIATE**, the track deviation is instantly changed (as if you specified a nonzero **.track** component for **Traffic_InsertVehicle**).

If mode is **SETTRACK_WISH**, the current track deviation does not change, but the vehicle will change its route to reach the desired track deviation.

See appendix A.9 for further options and numerical values.

5.2.8.33 Traffic_IsOnInvisiblePlate

Queries whether a given vehicle is on an invisible plate.

Arguments:

id	integer	object identifier

Return Values:

isinvis	integer	The boolean result (0 or 1) or a negative error code.

5.2.9 World Functions

These functions are used to set weather conditions. Nighttime is a weather condition in the sense of this section.

5.2.9.1 World_GetWeather

Query the currently active weather conditions.

Arguments:

pure	integer	set to 1 to ignore daytime.

Return Values:

weather	integer	see appendix A.12 for a list of possible values.

5.2.9.2 World_SetWeather

Set the current weather conditions.

Arguments:

weather	integer	see appendix A.12 for a list of possible values.
dist	float, opt.	If *dist* is 0 (or omitted), the change of weather occurs immediately. Otherwise, the weather is changed smoothly over a driven distance given by *dist* (in m). A smooth change is not possible to or from winter weather (snow).

Return Values: none

Remarks: Also resets the skybox orientation (see **World_SetNorth**)

5.2.9.3 World_SetWeatherParameter

Set parameters affecting the current weather conditions.

Arguments:

parameter	integer	What to set, see appendix A.13 for a list of possible values.
args		The value(s) to set, depends on *parameter*.
...		

Return Values: none

5.2.9.4 World_GetWeatherParameter

Query a parameter affecting the active weather conditions.

Arguments:

parameter	integer	What to retrieve, see appendix A.13 for a list of possible values.

Return Values:

values	The current value(s), depends on *parameter*.
...	

5.2.9.5 World_SetWind

Set the (lateral) wind force acting on the driver.

Arguments:	*bft1*	float	Wind force at vehicle's front in Beaufort. Positive is wind from the left (i.e. pushing right).
	bft2	float	Wind force at vehicle's rear in Beaufort. For road trains, rear means last trailer's rear.
	roadno	integer, opt.	Reference position for the wind direction (default is driver's position).
	distance	float, opt.	
	angle	float, opt.	
	track	float, opt.	
	world	integer, opt.	
Return Values:	none		

5.2.9.6 World_SetRoadFriction

Set the friction coefficients wheel/road overriding the default given by the current weather.

Arguments:	*adhesive*	float	Adhesive friction coefficient. ≤ 0 means default (either weather or 10% more than sliding).
	sliding	float, opt.	Sliding friction coefficition ≤ 0 means default (either weather or 10% less than adhesive).
Return Values:	none		

5.2.9.7 World_SetFogSight

Set the range of vision for foggy weather.

Arguments:	*reserved*	integer	must be set to 0.
	dist	float	the range of vision in m
Return Values:	none		

5.2.9.8 World_GetFogSight

Get the range of vision at foggy weather.

Arguments:	none		
Return Values:	*fog sight*	float	sight in m

5.2.9.9 World_GetRoadDistance

Calculates the length of the shortest path from one position to another.

Arguments:	*road1*	integer	road number of starting point.
	dist1	float	distance value of starting point.
	angle1	float	road angle of starting point.
	world1	integer	world identifier of start point. Set to 0 to use current world.
	road2	integer	road number of end point.
	dist2	float	distance value of end point.
	angle2	float	road angle of end point.
	world2	integer	world identifier of end point. Set to 0 to use current world.
	flags	integer,opt.	bitwise combination of flags from appendix A.15.
Return Values:	*distance*	float	length of shortest route, $\geq$ 10.000 means infinite (no route).
	dir	integer	1, if reaching the destination from the opposite direction (see flags). 0 else.

5.2.9.10 World_GetAbsoluteDistance

Get the distance between two road positions as the crow flies.

Arguments:	*road1*	integer	road number of starting point.
	dist1	float	distance value of starting point.
	angle1	float	road angle of starting point.
	track1	float	road track of starting point.
	world1	integer	world identifier of start point. Set to 0 to use current world.
	road2	integer	road number of end point.
	dist2	float	distance value of end point.
	angle2	float	road angle of end point.
	track2	float	road track of end point.
	world2	integer	world identifier of end point. Set to 0 to use current world.
Return Values:	*distance*	float	distance in m.

5.2.9.11 World_LookupRoadNumber

Translate between tile-specific road numbers and global numbers.

Arguments:	*world*	integer	World id, see appendix A.1.
	tilename	string	Name of the tile (valid names depend on the world).
	tilerdno	integer	Road number on the tile.
Return Values:	*globrdno*	integer	Positive road number if successful, 0 else.
See also:	**World_LookupRoadTile**		

5.2.9.12 World_LookupRoadTile

Translate between global and tile-specific road numbers.

Arguments:	*world*	integer	World id, see appendix A.1.
	globrdno	integer	Global road number
Return Values:	*tilerdno*	integer	Positive road number on the tile if successful, 0 if not.
	tilename	string	Name of the tile.
See also:	**World_LookupRoadNumber**		

5.2.9.13 World_GetRoadToPlateVector

Translation between global road number and plate index.

Arguments:	*roads_max*	integer	highest road number in current world
	plates vector	vector	plates vector of **World_EnumTiles**
Return Values:	*index vector*	vector	translation vector between global road number and plate index
See also:	**World_LookupRoadTile**		

5.2.9.14 World_SetRoadMode

Set surface quality of all roads.

Arguments:	*roadmode*	integer	see appendix A.14 for possible values
Return Values:	none		

5.2.9.15 World_SetNorth

Set the sky orientation relative to standard orientation.

Arguments:	*angle*	float	the angle in degree
Return Values:	none		
Remarks:	Call this function after setting the weather.		

5.2.9.16 World_SetNewLap

Set surface new lap.

Arguments:

justreset	int, opt.	If this is set to 1, accumulated data for the current lap is cleared while neither incrementing the lap counter nor emitting any measurement errors.

Return Values: none

5.2.9.17 World_GetJunctionCenter

Return the road coordinates of the center of the next junction as seen from the given road position.

Arguments:

inrdno	integer	
indist	float	
inangle	float	
intrack	float, opt.	ignored
inworld	integer, opt.	

Return Values:

outrdno	integer	in case of function failure, this is a (negative) error code.
outdist	float	
outangle	float	
outtrack	float	
outworld	integer	

Remarks: The absolute position corresponding to the returned road position faces towards the input road position. On junctions, there is no unique correspondence between road and absolute positions. The function only guarantees that the returned road position represents the junction's center point and the direction facing towards the input road position. No particular relation between input and output road number is implied.

Do not rely on the particular choice made by the functions, because it might change in future software versions.

5.2.9.18 World_GetWorldName

Return the name of a world given a world identifier.

Arguments:

id	integer	the world identifier, c.f. appendix A.1.

Return Values:

name	string	world name, empty string for invalid id.

5.2.9.19 World_FindWorldByName

Return the id of a world identified by its name.

Arguments:	*name*	string	world name
Return Values:	*id*	integer	a negative error code or the positive world identifier, c.f. appendix A.1.

5.2.9.20 World_GetLanes

Return an array of lane widths at the given road position. The function ignores the position track and only extracts the direction from the angle. It then returns the lane values to the right of the center line. For invalid positions, the array will be empty.

Arguments:	*flags*	integer	reserved, must be set to 0.
	inrdno	integer	
	indist	float	
	inangle	float	only direction matters
	intrack	float, opt.	ignored
	inworld	integer, opt.	
Return Values:	*lanetype*	integer	c.f. appendix A.44, LANETYPE_NONE designates the end of the array.
	lanewidth	float	width of the lane in m.
	...		

5.2.9.21 World_EnumTiles

Enumerates all tiles in the world.

Arguments:	*id*	integer, opt.	world id
Return Values:	*count*	integer	the positive number of tiles or a negative error code.
	tiletype1	string	the type of the first plate.
	tilelookup1	string	the lookup name of the first plate.
	...		

Remarks: The return vector contains $count$ pairs of strings. Its end is marked by an empty $tiletype$.

5.2.9.22 World_EnumSignSchemes

Enumerates the signing schemes defined for a world.

Arguments:	*id*	integer, opt.	world id, default is current world
Return Values:	*count*	integer	the positive number of signing schemes or a negative error code.
	scheme1	string	the name of the first scheme.
	...		
Remarks:	For a valid world, at least the "default" scheme is always present.		

5.2.9.23 World_GetTileSignVariant

Query the sign variant for a given plate according to a given signing scheme.

Arguments:	*tilelookup*	string	the plate's lookup, c.f. **World_EnumTiles**
	scheme	string, opt.	the signing scheme, c.f. **World_EnumSignSchemes**, default is "default"
	world	integer, opt.	world id, default is current world
Return Values:	*variant*	integer	the positive variant number or a negative error code.
Remarks:	The function may return 0 even if the plate lookup does not exist.		

5.2.9.24 World_IsRoadOnInvisiblePlate

Query whether a given road road position is on an invisible plate.

Arguments:	*roadno*	integer	The road about which info is requested.
	distance	float	The distance on the road.
	angle	float, opt.	
	track	float, opt.	
	world	integer, opt.	
Return Values:	*isinvis*	integer	The boolean result (0 or 1) or a negative error code.
Remarks:	The $world$ parameter is ignored.		

5.2.9.25 World_GetInvisibleSourcePosition

Find a position on an invisible plate (reverse from the given position).

Arguments:	*dirwish*	integer	A hint in which direction to search, see remarks.
Arguments:	*roadno*	integer	The road to which a route source should be found.
	distance	float	The distance on the road.
	angle	float	
	track	float, opt.	
	world	integer, opt.	
Return Values:	*srcroadno*	integer	The road number of the found source position or a negative error code.
	srcdistance	float	
	srcangle	float	Either 0 or 180.
	srctrack	float	Always 0 (i.e. default).
	srcworld	int	In case of success, always the current world number.

Remarks: The $dirwish$ parameter has the following meaning: 0 means random direction, 2 means straight on (if possible). 1 means take the first junction right (the route will bend off left) and 3 means take the first junction left.

5.2.9.26 World_GetRoadCount

Get the highest valid road number for a given world.

Arguments:	*worldno*	integer	The world to get information about, pass 0 for the current world.
Return Values:	*roadno*	integer	The (positive) highest valid road number or a negative error code.

5.2.9.27 World_GetNextRoad

Get information about the road following a given road in the current world.

Arguments:	*dirwish*	integer	If multiple next roads exist, the result is chosen using this flag. Specify 1 for left, 2 for straight on or 3 for right. Negative values also include forbidden roads.
	curroadno	integer	The current road number.
	curdist	float	The distance value on the current road.
	curangle	float	The angle on the current road in degrees. Only the direction information is used.
	curtrack	float	The track on the current road. Currently ignored, but may be used to determine available directions in the future.
Return Values:	*nxtroadno*	integer	The (positive) next road number or a negative error code. In the latter case, the subsequent return values are not set.
	nxtangle	integer	The angle on the result road (0 or 180).
	nxtlength	float	The length of the result road in m.
	nxtjunc	integer	A stricly positive integer if the result is on a junction, 0 otherwise.

5.2.9.28 World_IsRoadOnJunction

Query wheather the road belongs to a junction or a roundabout.

Arguments:	*roadno*	integer	Road to query.
Return Values:	*junc*	integer	Junction number. 0 if not on junction.
	roundabout	integer	Roundabout number. 0 if not on roundabout.

5.2.9.29 World_GetRoadLength

Query the length of a road.

Arguments:	*roadno*	integer	Road to query.
Return Values:	*length*	float	Length of the road. ≤ 0 if road does not exist.

5.2.9.30 World_GetRoadCurvature

Get the road's curvature at the given position.

	roadno	integer	Road number.
	distance	float	Distance along the road (measured at chord line).
	angle	float	Only used for direction: 0 means in standard road direction, 180 in the opposite direction.
	track	float, opt.	Curvature becomes larger towards the center of curvature.
	world	integer, opt.	Ignored.
Return Values:	*curv*	float	The curvature (inverse radius).

5.2.9.31 World_LookupCreate

Create a "lookup by road position" structure. Only available in a scenario.

Arguments: none

Return Values:	*id*	integer	A handle to reference the lookup object or a negative error code.

5.2.9.32 World_LookupLoad

Initialize a "lookup by road position" structure from a vector previously created using **World_LookupSave**. Only available in a scenario.

Arguments:	*id*	integer	The id of the lookup object.
	values	vector	The (opaque) data vector.
Return Values:	none		

5.2.9.33 World_LookupSave

Save a "lookup by road position" structure into a to restore it later using **World_LookupLoad**. Only available in a scenario.

Arguments:	*id*	integer	The id of the lookup object.
Return Values:	*err*	integer	SYSERR_OK if successful, error code if failed.
	data	vector	The result vector if successful.

5.2.9.34 World_Lookup

Query a "lookup by road position" for values recorded earlier using **World_LookupRecord**. Only available in a scenario. For road positions between recorded values, the result is a linear interpolation.

Arguments:	*id*	integer	The id of the lookup object.
	road	integer	The road number.
	distance	float	The distance value on the road.
	angle	float	The angle value on the road. Only directional information is used.
Return Values:	*err*	integer	SYSERR_OK if successful, error code if failed.
	values	vector	The data if successful.

Remarks: You must anticipate this function to fail, because recordings are rarely complete. Always implement a fallback strategy.

5.2.9.35 World_LookupClear

Reset a "lookup by road position" object. Clears all previously recorded data. Only available in a scenario.

Arguments:	*id*	integer	The id of the lookup object.
Return Values:	none		

5.2.9.36 World_LookupRecord

Store values in a "lookup by road position" object. Only available in a scenario.

Arguments:	*id*	integer	The id of the lookup object.
	road	integer	The road number. If 0, the current driver position is used ignoring $distance$ and $angle$.
	distance	float	The distance value on the road.
	angle	float	The angle value on the road. Only directional information is used.
	data	vector	The data to store. Must consist of float values (i.e. no strings).
Return Values:	none		

5.2.9.37 World_LookupSimplify

Simplify values in a "lookup by road position" object in order to reduce memory usage. Only available in a scenario.

Arguments:

id	integer	The id of the lookup object.
gran	float	Reduce to averages over road pieces of approx. length $gran\ m$.
tolerance	vector, opt.	Tolerance vector. The data is further reduced by omitting stored values, for which linear interpolation does not produce errors larger than tolerance. The default tolerance (used when specifying 0) is 0.001.

Return Values: none

Chapter 6

Complex Statements

6.1 The STMT Directive

Using the **STMT** directive, you can combine function calls, procedure calls, assignments and calculations into a single line. You can simply use STMT as a replacement for **LET** (with '='), **FUNC**, **CALC** and **CALL**, but you can also build more complex statements.

The possible syntax variants are numerous, whence we specify them in a much more abstract way than for the other directives:

Listing 6.1: STMT Syntax

```
[STMT] [leftlist =] rightlist
```

The **STMT** directive itself and the assignment target *leftlist* are optional. If you specify an assignment target *leftlist*, it must be compatible with *rightlist*, i.e. both must specify the same number of scalar items and the single items must be compatible. The latter means, that you cannot assign e.g. a constant numeric value to a string variable.

6.2 Left Lists

The assignment target list can be in any of the following forms:

Listing 6.2: leftlist variants

```
leftlist := leftfinite
leftlist := leftinfinite
leftinfinite := leftfinite, leftinfinite
leftfinite := leftfinite1, [..., ] leftfinite_n
leftinfinite := vecname
leftinfinite := vecname[infiniteselector]
leftfinite := varname
leftfinite := stringname
leftfinite := vecname.suffix
leftfinite := vecname[rightfinite]
leftfinite := -
```

As for the **FUNC** directive, the last form of $leftfinite$ assigns a scalar component to nothing.

6.3 Vector Selectors

For both left and right lists, square brackets have an extended syntax as follows:

Listing 6.3: vector selector syntax

```
vecname[rightfinite]
vecname[rightinfinite]
vecname[rightfinite,...]
vecname[...,-,...]
```

$rightfinite$ and $rightinfinite$ are finite/infinite lists of scalar items (see below for right list syntax). Each item selects a vector element of $vecname$. Explicitly invalid elements are not allowed (like e.g. $vecname[-1]$), while implicitly invalid indices (e.g. by variable value) result in a void target for left lists or 0 / empty string for right lists.

The dots in the third form are to be understood verbatim and select all vector elements following the last element in $rightfinite$. If the latter was invalid, they specify an infinite sequence of invalid indices. In any case, the dots make the whole expression infinite.

The fourth form is only allowed in left lists. The dots are not meant verbatim, but the hyphen specifies an explicit invalid index and hence an explicit void target at the corresponding position.

For infinite vector selectors not ending in ..., trailing default elements after the explicitly specified minimal length are not interpreted as 0, but as invalid.

If a vector selector of an assignment target contains the same valid index more than once, the behaviour is unspecified.

6.4 Right Lists

Right lists are finite or infinite lists of scalar items, constant or variable. They are composited from the following forms:

Listing 6.4: rightlist variants

```
rightlist := ( rightlist )
rightlist := rightfinite
rightlist := rightinfinite
rightinfinite := rightfinite, rightinfinite
rightfinite := rightfinite1, [..., ] rightfinite_n
rightinfinite := vecname
rightinfinite := vecname[infiniteselector]
rightfinite := rightscalar
rightscalar := varname
rightscalar := stringname
rightscalar := vecname.suffix
```

```
rightfinite := vecname[rightfinite]
rightscalar := vecname[rightscalar]
rightscalar := number
rightscalar := "string"
rightscalar := rightlist1 CMP rightlist2
rightlist := rightlist1 OP rightlist2
rightlist := funcname(rightlist)
rightlist := procname(rightlist)
rightlist := rightlist[vectorselector]
rightlist := rightlist.suffix
```

OP specifies an arithmetic operator (c.f. **CALC**) or a logical operator ((**AND**, **OR** and **XOR**)). Operations OP can operate on vectors and other lists component-wise. CMP is a comparison operator with the same meaning as for **IF** statements.

When used as arithmetic values, boolean expressions are integers with value 0 for *false* or 1 for *true*. On the other hand, numbers can be used as boolean expressions, where nonzero means *true*.

In composite expressions, the operator precedence for arithmetic operators is as for the C language, while logical operators have no precedence, i.e. they are evaluated from left to right. If in doubt, you should use parantheses.

For scalar expressions, the term right of **AND** and **OR** is only evaluated if necessary, i.e. when the term left of **AND** is *true* or the term left of **OR** is false.

Note, that the **XOR** operator cannot act on infinite list, because $0\ XOR\ 0 = 1$, which would lead to a nonterminating infinite list.

Chapter 7

Advanced Topics

7.1 Namespaces

There is an extended syntax for variable names, labels and procedure names.

Listing 7.1: Extended name syntax

```
identifier
::identifier
::namespace::identifier
```

The interpretation of names depends on the *current namespace*, which can be changed using the **NAMESPACE** and **ENDNAMESPACE** directives. For each **NAMESPACE** there must be a matching **ENDNAMESPACE**. Nesting is allowed for temporarily switching namespaces — the namespaces themselves cannot be nested, i.e. namespace a inside namespace b is the same as a inside c or a at global scope.

For variable and label declarations (**VAR**, **VEC**, **STRING**, **TIMER**, **LOCALVAR**, **LOCALVEC**, **LABEL**, **PROC**), the current namespace is prepended to the name, if the first form is used.

For variable and label usage (e.g. **LET**, **FUNC**, **CALL**, **STMT**, **CALC**) the first form is resolved prepending the current namespace and falling back to global namespace if that does not succeed. An example:

Listing 7.2: Namespace example

```
VAR x = 1       // declare ::x
NAMESPACE a     // now namespace a
VAR x = 2       // declare ::a::x
NAMESPACE b     // now namespace b
VAR x = 3       // declare ::b::x
NAMESPACE       // temporarily global namespace
VAR y = 4       // declare ::y
ENDNAMESPACE    // back to namespace b
    ::x = x     // ::b::x -> ::x
    ::a::x = x  // ::b::x -> ::a::x
    y = x       // ::b::x -> ::y
```

```
ENDNAMESPACE    // back to namespace a
ENDNAMESPACE    // back to global namespace
    DEBUGVAR y // output 3
```

7.2 Libraries

You can put part of your code into library files and include it from other scripts. Library files must be put into the ./SCENARIOS/INCLUDE/ folder. Their structure is very similar to scenario files and they should be stored using the file extension *.SCI. The only syntactical difference to scenario files is, that library files do not contain header directives. Instead, they only contain scenario scripting code between a **BEGIN** and an **END** directive.

Inside the scenario script, you include a library file using the **INCLUDE** directive:

Listing 7.3: Library inclusion syntax

```
INCLUDE "MyLibrary.sci"
```

This includes all the code between **BEGIN** and **END** from the file ./SCENARIOS/INCLUDE/MYLIBRARY.SCI at the location, where you put the **INCLUDE** line.

7.2.1 Library Guidelines

Since the code from a library file is pulled into each scenario, which includes the library, you must take care not to create name clashes. In particular, you must either decorate all of your variables and labels with a prefix specific to your library or use namespaces. For the sake of readability, it is advisable to put the whole library content into a private namespace and only put the external interface into either global namespace or a dedicated interface namespace.

Usually, your library starts with variable declarations. Then, a GOTO statement leads to the end of the library file. After the GOTO statement, you write your library routines. In this way, you can include the library right at the beginning of a scenario file and then use its routines.

A library file should always contain a comment block at its beginning, which describes how to use the library.

Listing 7.4: Example.sci

```
// This is an example library
// It provides a random-like function ExampleRandom, which returns a
// predetermined number sequence.
// INCLUDE the library at the beginning of your script. In your script's
// initialization code, select the number sequence using
//
// CALL ExampleInit(a)
//
// where a is 1,2 or 3 and selects the number sequence.
// Later, get numbers with code like
//
// number = CALL ExampleRandom()
//
BEGIN
VEC Example_Seq1 5,1,2,3,4,5
VEC Example_Seq2 4,2,4,6,8
VEC Example_Seq3 8,1,2,3,5,7,11,13,17
VEC Example_Seq
VAR Example_Idx 0

LET Example_Seq Example_Seq1            // ensure something sensible

GOTO Example_End

PROC ExampleInit
    IF argv[0] <= 1
        LET Example_Seq = Example_Seq1
    ELSIF argv[0] = 2
        LET Example_Seq = Example_Seq2
    ELSE
        LET Example_Seq = Example_Seq3
    ENDIF
    LET Example_Idx = 0
    RETURN

PROC ExampleRandom
    CALC Example_Idx = Example_Idx + 1
    LET retv[0] = Example_Seq[Example_Idx]
    CALC Example_Idx = Example_Idx % Example_Seq[0]
    RETURN

LABEL Example_End
END
```

Listing 7.5: Example.sce

```
...
BEGIN
    INCLUDE "Example.sci"

    VAR rnd
    ... (some script initialization)
    CALL ExampleInit(2)
    ... (further initialization)

LABEL mainloop
    CALL rnd = ExampleRandom()
    ... (do something)
    SLEEP 0
    GOTO mainloop

    ...
END
```

7.3 World Editor

To use custom built worlds on the simulator, there are predefinied procedures which set the traffic signs as defined in the editor. Also the start and end position can be set, if they are defined in the world editor. Here you have an example how to use these predefined procedures to init your own world.

Listing 7.6: WorldEditor.sce

```
...
BEGIN

VEC endpos
VEC startpos
VAR mywnr

VAR worldId

// variables used by DynamicWorlds_LIB.sci
// include it before DynamicWorlds_LIB.sci !
INCLUDE "DynamicWorlds_Common_LIB.sci"

// provides procedures to initialize a custom world
INCLUDE "DynamicWorlds_LIB.sci"


System_InitPhase(1)
    mywnr = Driver_GetID(0)

    // get the world constant for your world
    worldId = World_FindWorldByName("Some world")

    // init world, set traffic signs as specified in the world editor
    DynamicWorlds::SetupWorld(worldId, "default")
    // the below is only necessary, if you use the runtime features.
    // you can specify TRAFFIC_LIGHT or TRAFFIC_NORMAL instead of
    // TRAFFIC_NONE. Then, you MUST also place the runtime call
    // in the mainloop. The resulting traffic density depends on
    // your world construction. Your mileage may vary.
    DynamicWorlds::SetupTraffic(DynamicWorlds::TRAFFIC_NONE, worldId)

    // get start and end position
    startpos = DynamicWorlds::QueryInfo(DynamicWorlds::QUERY_STARTPOS)
    endpos = DynamicWorlds::QueryInfo(DynamicWorlds::QUERY_ROUTEEND)
    IF startpos.road > 0    // if start position is set
        Driver_SetPositionDirect(startpos.road, startpos.dist, startpos.angle, \
                                 startpos.track, WORLD_UNCHANGED)
    ENDIF
        // else the default start position is 1,5,0

    IF endpos.road > 0
        // if end position is set, set the route
        Traffic_SetRoute(mywnr, ENDROUTE_RANDOM, endpos)
    ENDIF
System_InitPhase(0)

LABEL mainloop
    ...
    // The below is only necessary if you want per tile runtime code to be
    // called and/or you want automatic traffic. See SetupTraffic above.
    DynamicWorlds::Runtime()
    ...
GOTO mainloop

...

END
```

7.4 Traffic Lights

If you want to insert traffic lights for a simple 4-way junction, call **Traffic_SetSigns** for each of the four lights:

Listing 7.7: Simple Traffic Lights

```
Traffic_SetSigns(STATIC_TRAFFICLIGHT, 0, pos1)
Traffic_SetSigns(STATIC_TRAFFICLIGHT, 1, pos2)
Traffic_SetSigns(STATIC_TRAFFICLIGHT, 0, pos3)
Traffic_SetSigns(STATIC_TRAFFICLIGHT, 1, pos4)
```

Here, *pos1*, ..., *pos4* specify road positions beside the four roads joined at the junction in e.g. clockwise order.

If this is the only thing you want to do, read no further.

7.4.1 Levels of Functionality

In contrast to other static objects (like speed limit signs), traffic lights are not independent of each other. You specify their relationship to other traffic lights with the second argument of **Traffic_SetSigns**.

With this argument, a traffic light is bound to a so-called **logical traffic light**, which specifies the state the traffic light is in (red, green, ...). Many visible traffic lights can be bound to the same logical traffic light (two in example 7.7).

Groups of logical traffic lights are associated to **traffic light programs**. The most simple traffic light program only knows a green phase, a red phase and intermediate amber phases. The two logical traffic lights 0 and 1 are bound to this program, where logical traffic light 1 has its green and red phase almost reversed with respect to logical traffic light 0. Almost, because the red phases are a little bit longer to accomodate for the other traffic light's amber phases.

Depending on your country settings, the amber phase following the red phase may not be shown. This just means that it is displayed as part of the red phase. It is present nevertheless.

The example in listing 7.7 thus creates four visible traffic lights, which are all associated to the same traffic light program – two of them to logical traffic light 0 and the other two to logical traffic light 1.

If you used an analogous sequence of function calls to place traffic lights around a second junction, these would also be bound to the same logical traffic lights and thereby to the same traffic light program. In particular, manipulating a single traffic light using **Surround_SetLight** affects all eight traffic lights.

7.4.2 Advanced Positioning

A single traffic light as created with

Listing 7.8: Standard Traffic Light Insertion

```
Traffic_SetSigns(STATIC_TRAFFICLIGHT, logtl, pos)
```

serves two main purposes. Firstly, it creates a graphical object which displays the logical traffic light *logtl*'s state. Secondly, it is a traffic regulating object, which is

regarded by simulated vehicles and triggers a driver's fault when not obeyed by the human driver.

Both functions are equally important. What now, if you wanted the traffic light to be displayed at another location than beside the road right before the associated junction – e.g. behind the junction?

Settings like this can be accomplished by splitting the traffic light's functionality. You insert a regulating traffic light (visible or not) at the standard location and an additional traffic light display without the regulating function whereever you like it.

Invisible traffic lights can be created by associating them to negative logical traffic lights according to the formula $lt' = -1-lt$. Traffic lights without regulating function can be inserted using the sign types **STATIC_TRAFFICLIGHTSLAVE** and **STATIC_TRAFFICLIGHTSLAVE_NOPOST**.

Consider the situation described above. Let *pos1* denote the standard position before the junction and *pos2* the desired location behind the junction. For simplicity, we want the traffic light to be bound to logical traffic light 1. Then, the wanted setting can be created by the following sequence of function calls:

Listing 7.9: Traffic Light Behind Junction

```
Traffic_SetSigns(STATIC_TRAFFICLIGHT, -2, pos1)     // invisible
Traffic_SetSigns(STATIC_TRAFFICLIGHTSLAVE, 1, pos2) // display only
```

7.4.3 Traffic Light Programs

This section lists the available traffic light programs. In the list below, LTR denotes the logical traffic light's number. All programs can be temporarily replaced by an additional program displaying yellow blinking lights. When this program is selected, the traffic lights are not regarded by simulated traffic anymore. In order to select the yellow blinking program, use **Surround_SetLight** to set the phase of an arbitrary traffic light bound to the program under consideration to

- **TLST_YELLOW_BLINK**.

The original program is reactivated by setting one of the phases

- **TLST_RED**,
- **TLST_GREEN**,
- **TLST_YELLOW** or
- **TLST_REDYELLOW**.

The phase lengths in the list are default lengths. You can use **Surround_SetLightDurations** to set the phase lengths of the individual traffic light program belonging to a given logical traffic light.

Please note, that the *Separate Left-Turning Lane Program* produces correctly working traffic lights, but neither the automatic traffic nor the fault recognition system can cope with this complex traffic light situation. If you want to use it, you must take care of all of this using scripting language code.

LTR	Meaning
	Simple Traffic Light Program (green $= 15s$)
0	direction 1
1	direction 2 (opposite phase of logical traffic light 0)
	Separate Left-Turning Lane Program (roads in clockwise order)
2	left-turning lane on road 1 (green $= 6s$)
3	straight-and-right lane on road 1 (green $= 14.4s$)
4	left-turning lane on road 2
5	straight-and-right lane on road 2
6	left-turning lane on road 3
7	straight-and-right lane on road 3
8	left-turning lane on road 4
9	straight-and-right lane on road 4
	Simple Traffic Light Program 2 (green $= 12s$)
10	direction 1
11	direction 2 (opposite phase of logical traffic light 10)
	Simple Traffic Light Program 3 (green $= 17.5s$, for large junctions)
12	direction 1
13	direction 2 (opposite phase of logical traffic light 12)
	Three Sequential Program (green $= 15s$)
14	direction 1 (1st to show green)
15	direction 2 (2nd to show green)
16	direction 3 (3rd to show green)
	Four Sequential Program (green $= 15s$)
17	direction 1 (1st to show green)
18	direction 2 (2nd to show green)
19	direction 3 (3rd to show green)
20	direction 4 (4th to show green)

Chapter 8

The Network Protocol

8.1 Introduction

You can influence a running simulation from another computer using the network protocol. Put into a nutshell, (almost) all function calls documented for the scripting language are available to be executed remotely.

The network protocol is built on top of the TCP/IP protocol (by default on port 1365), which provides reliant connection oriented data transfer. The simulator acts as a server, to which the controlling program has to establish a connection before any data can be transferred.

The details of the connection process are described in section 8.2. Having established a connection, one can then send commands and retrieve data. The details of this process are the subject of section 8.3.

The **Programming Tool** package also includes the source code (C++) of an example client. Please have a look at the TESTCLIENT folder.

8.2 Connection and Transfer Protocol

We will not discuss how to establish a TCP/IP connection to the server. Please read one of the many standard textbooks on TCP/IP programming for this.

Though the underlying transport protocol is stream oriented, all data transfers are packaged in message containers. The layout of the message container is

Listing 8.1: Network Container Structure

```
typedef struct _MSGHDR {
    BYTE signature;          // must be 0xcc
    WORD length;             // length of the message
    BYTE command;            // command code
} MSGHDR;

typedef struct _DATAMSG {
    MSGHDR hdr;              // Header
    char   data[0x8000];     // the actual data
} DATAMSG;
```

As you might have guessed, only the first **hdr.length** bytes of each **DATAMSG** are transferred over the network. The length field is encoded in little-endian order.

The command field can contain the following valid values:

Command	Remarks
1 = **REGISTER_NAME**	The own name must be written into the data field.
2 = **XFER_DATA**	The data (a string) must be written into the data field.
3 = **REQUEST_SESSION**	The name of the server must be written into the data field.
4 = **SESSION_REQUEST_RESPONSE**	A ”1“ in the data field confirms the connection.
5 = **SESSION_CLOSE**	no parameters
6 = **DEREGISTER_NAME**	no parameters

After a TCP connection has been established, client and server exchange names and the client initiates a session. For details of the session initialization process, please have a look at the TESTCLIENT source code.

Afterwards, both the client’s requests and the server’s answers are packaged as strings inside **XFER_DATA** packets. A terminating zero byte is included in the packet.

8.3 Data Protocol

As mentioned above, both the function call and the server's response are encoded as strings.

The server's response will always be an identifier for the function producing the result followed by a space separated list of return values as documented for the function. The function call itself is encoded with the following general syntax:

Listing 8.2: Network Function Call Syntax

```
name arg1 arg2 arg3 ...
```

where *name* is a function name (c.f. chapter 5) and the function's arguments are appended as a space separated list. Put into a nutshell, in contrast to the scripting language you have to omit the **FUNC** keyword and the brackets. In addition, you replace commata by spaces.

Appendix A

Predefined Constants

A.1 World Constants

Name	Value	Comments
WORLD_UNCHANGED	-2	use this world to keep the same world, without resetting anything
WORLD_DEFAULT	0	some default behaviour, usually current world
WORLD_ROUNDCOURSE	4	$6.2km$ of flat country road, endless
WORLD_MOTORWAY	5	2x2 lane motorway, multiple segments connected by jumps, endless
WORLD_TRAINER	6	map of a small city
WORLD_SEGMENTS	7	several mutually unconnected road segments (motorway, road, crossing)
WORLD_HIGHRESOLUTION	8	small circuit course with crossing, hi-res graphics
WORLD_MOUNTAIN	9	open course in highlands
WORLD_EMPTY	10	empty world for free driving (special purpose, not suitable for scenarios)
WORLD_SMALLTOWN	11	small town with many crossings
WORLD_RACING	12	simple racing course with service lane
WORLD_TOWN	13	city with many crossings and 2 lanes per direction
WORLD_WASTELAND	14	simple circuit with wasteland graphics
WORLD_CITY2	15	larger city prebuilt with tiles. Only use this if you know what you are doing.
WORLD_UNDEFINED	48	on input: current world. on output: invalid position

Functions: Driver_SetWorld

Remarks: Not all worlds are suitable for scenario scripts. Some worlds, for instance, don't support foreign vehicles. Check the comments in the above table before using a world. Worlds not documented in chapter B are experimental and may behave unexpectedly. In addition, they may change in future software revisions or even be removed completely.

A.2 Language Constants

Name	Value	Comments
LANG_GERMAN	1	
LANG_ENGLISH	2	
LANG_DUTCH	3	Netherlands
LANG_DANISH	4	
LANG_SWEDISH	5	
LANG_SPANISH	6	
LANG_GREEK	7	
LANG_FRENCH	8	
LANG_DUTCH_BELGIAN	9	Belgium (Flanders)
LANG_SWIZZ	10	
LANG_FARSI	11	
LANG_ARABIC	12	
LANG_URDU	13	
LANG_ICELAND	14	
LANG_LAST	15	all known languages have smaller numbers

Functions: System_GetLanguage

A.3 Country Constants

Name	Value	Comments
COUNTRY_GERMANY	1	
COUNTRY_GREAT_BRITAIN	2	left-hand-traffic
COUNTRY_NETHERLANDS	3	
COUNTRY_DENMARK	4	
COUNTRY_SWEDEN	5	
COUNTRY_SPAIN	6	
COUNTRY_GREECE	7	
COUNTRY_FRANCE	8	
COUNTRY_BELGIUM	9	
COUNTRY_SWITZERLAND	10	
COUNTRY_IRAN	11	
COUNTRY_UAE	12	
COUNTRY_PAKISTAN	13	left-hand-traffic
COUNTRY_ICELAND	14	
COUNTRY_ALGERIA	15	
COUNTRY_HUNGARY	16	
COUNTRY_SLOVENIA	17	
COUNTRY_ITALY	18	
COUNTRY_USA	19	
COUNTRY_POLAND	20	
COUNTRY_RUSSIA	21	
COUNTRY_AUSTRIA	22	
COUNTRY_ISRAEL	23	
COUNTRY_BULGARIA	24	
COUNTRY_MEXICO	25	
COUNTRY_LATHUNIA	26	

Name	Value	Comments
COUNTRY_SAUDI_ARABIA	27	
COUNTRY_TURKEY	28	
COUNTRY_LAST	29	all known countries have smaller numbers.

Functions: System_GetCountry

A.4 Object Constants

Name	Value	Comments
OBJECT_ALL_CARS	0	random car type
OBJECT_SMALL_CAR	1	
OBJECT_TRUCK	2	
OBJECT_CYCLIST	3	
OBJECT_PARK_CAR	6	
OBJECT_TREE	7	
OBJECT_CHILD	8	
OBJECT_ROADWORK	9	
OBJECT_DEER	10	
OBJECT_PARK_CAR2	11	special car (can open door, currently only one available)
OBJECT_PILLAR	12	
OBJECT_ADULT	13	
OBJECT_BUS	14	
OBJECT_EMERGENCY	15	
OBJECT_DONKEY	16	
OBJECT_CART	17	
OBJECT_CAMEL	18	
OBJECT_MOTORBIKE	19	
OBJECT_JEEP	20	
OBJECT_TAXI	21	
OBJECT_HEAVY_TRUCK	23	big truck (Pakistan version only)
OBJECT_PARK_TRUCK	26	

Functions: Traffic_InsertVehicle

A.5 Object Insertion Flags

Name	Value	Comments
INSFLAG_DEFAULT	0	
INSFLAG_ONCOMING	16	insert facing opposite direction
INSFLAG_USEANGLE	32	use angle verbatim (not corrected to pure direction)
INSFLAG_USETRACK	64	use track component (otherwise default track is used)
INSFLAG_ARGISROUTE	256	vector argument is a route destination, use current driver position as anchor
INSFLAG_ALLOW_ILLEGAL	512	explicitly allow insertion at illegal position (e.g. heading wrong way in one-way-street).
INSFLAG_EXPECT_FAILURE	2048	suppress log messages concerning some expectable failure causes (not enough space, against one-way, on circle, ...). Use this in order not to flood the debug log if you expect the call to often fail.
INSFLAG_USECLEARANCE	4096	the offset parameter does not specify a position offset, but a minimal clearance

Functions: Traffic_InsertVehicle

A.6 Information Request Constants

Name	Value	Comments
EVTI_WNRWITHUSFRONT	0	id of vehicle in front of object, same direction. 2nd value is distance.
EVTI_WNRWITHUSREAR	1	id of vehicle behind object, same direction. 2nd value is distance.
EVTI_WNRAGAINSTUSFRONT	2	id of vehicle in front of object, opposite direction. 2nd value is distance.
EVTI_WNRAGAINSTUSREAR	3	id of vehicle behind object, opposite direction. 2nd value is distance.
EVTI_WNRPEDESTRIANFRONT	4	id of pedestrian in front of object. 2nd value is distance.
EVTI_WNRPEDESTRIANREAR	5	id of pedestrian behind object. 2nd value is distance.
EVTI_WNROBSTACLEFRONT	6	id of obstacle in front of object. 2nd value is distance.
EVTI_WNROBSTACLEREAR	7	id of obstacle behind object. 2nd value is distance.
EVTI_WNRWITHUSLANEFRONT	8	id of vehicle on the same lane in front of object. 2nd value is distance.
EVTI_WNRWITHUSLANEREAR	9	id of vehicle on the same lane behind object. 2nd value is distance.
EVTI_WNRWITHUSLEFTFRONT	10	vehicle left and in front of the object. 2nd value is distance.
EVTI_WNRWITHUSRIGHTFRONT	11	vehicle right and in front of the object. 2nd value is distance.
EVTI_WNRWITHUSLEFTREAR	12	vehicle left and behind of the object. 2nd value is distance.
EVTI_WNRWITHUSRIGHTREAR	13	vehicle right and behind of the object. 2nd value is distance.
EVTI_AKTIV	16	status of the object. 0: not in the world, ready to be used. 1: intermediate state. 2: object is in the world.
EVTI_PHASE	17	behaviour phase of the object, see **AIPHASE_...**
EVTI_DISTANCE	18	driven distance in meter
EVTI_SPEED	19	current speed in m/s.
EVTI_SPEEDMEAN	20	mean speed in m/s.
EVTI_LIGHTSTATE	21	mask of active lights, see **LIGHTMASK_...**
EVTI_VEHICLELENGTH	25	length in m
EVTI_VEHICLEWIDTH	26	width in m
EVTI_DESIREDSPEED	27	desired speed in m/s
EVTI_ACCELERATION	28	acceleration in m/s^2
EVTI_DECELERATION	29	deceleration in m/s^2
EVTI_TRACKUSAGE	35	estimation of the track range covered by the vehicle (sampled at some points and parallel transported along the road to the vehicle center)
EVTI_CURVATURE_AHEAD	36	minimal and maximal curvature of the road ahead in degree/m. Additional parameter is how far to look.

Name	Value	Comments
EVTI_WHEELBASE	37	distance between axles, z of front axle, z of rear axle
EVTI_SPEEDLIMIT	38	current speed limit (without tolerance) in m/s

Functions: Traffic_GetInfo

A.6.1 Bit Values for EVTI_LIGHTSTATE

Name	Value	Comments
LIGHTSTATE_LOWBEAM	1	low-beam
LIGHTSTATE_BRAKE	2	brake lights
LIGHTSTATE_BLINK_LEFT	4	left blinker
LIGHTSTATE_BLINK_RIGHT	8	right blinker
LIGHTSTATE_BLINK_WARN	16	warn blinker (usually both blinkers)
LIGHTSTATE_REVERSE	32	reverse light (white lamp when reversing)
LIGHTSTATE_HIGHBEAM	64	hight-beam
LIGHTSTATE_BLINK_ON	256	blinkers currently visually on (if also activated with above flag)

A.6.2 Traffic Behaviour Phases for EVTI_PHASE

Name	Value	Comments
AIPHASE_VOID	0	not in the world
AIPHASE_ENDLESS	1	vehicle goes on forever (or until orderd otherwise)
AIPHASE_AIM_WAIT	2	waiting before going to a particular place
AIPHASE_AIM_RIDE	3	going to some particular place
AIPHASE_AIM_REACHED	4	particular place reached.
AIPHASE_WAITING	5	waiting
AIPHASE_CROSS_ONCE	6	car drives to the other end of the world
AIPHASE_CROSS_READY	7	after AIPHASE_CROSS_ONCE
AIPHASE_CROSS_STARTLEFT	8	start AIPHASE_CROSS_ONCE from the left
AIPHASE_CROSS_STARTRIGHT	9	start AIPHASE_CROSS_ONCE from the right
AIPHASE_PROGRAMMED	10	vehicle is on a special mission
AIPHASE_SCENARIO	11	vehicle is essential part of a subscenario
AIPHASE_START_PARKING	12	vehicle is in a parking maneouvre
AIPHASE_START_HALTING	13	vehicle is in a halting maneouvre (parking with running engine)
AIPHASE_PARKING	14	parking
AIPHASE_HALTING	15	halting, i.e. parking with running engine
AIPHASE_LEAVE_PARKING	16	reentering traffic after parking or halting
AIPHASE_LEAVE_PARKING_COMPLETE	17	finished reentering traffic (one round only)

A.7 Static Object Types

Name	Value	Comments
STATIC_NONE	0	not a valid object
STATIC_VILLAGE	1	Speed regulating village sign. value: 50=village entry, 100=village exit, 10=children-area 20=children-exit, 30=30-zone, 40=30-zone-end, 80=speedway-end, 90=speedway, 120=Motorway-Exit, 130=Motorway-Entry.
STATIC_SPEEDLIMIT	2	Speed limit. The sign value is the limit in km/h, available are 30,50,60,70,80,100,120 or use 0 for free limit.
STATIC_STOP	3	
STATIC_NORIGHTOFWAY	4	
STATIC_RIGHTOFWAY	5	value 0: white-yellow 'have right of way' sign, 1: end, 2: minor road from the left, 4: minor road from the right, 6: minor road crossing
STATIC_NOOVERTAKE	6	
STATIC_ONEWAY	7	One-way sign (white arrow on blue ground) value: 0=to right, 1=to left.
STATIC_ONEWAYWRONG	8	One-way sign (red ground with white bar)
STATIC_NOENTRY	9	No entry (red border around white)
STATIC_ROUNDABOUT	10	
STATIC_TURNING	11	Directional restriction, for values see **SIGNTURN_LEFT ...**
STATIC_CONTAINER	12	no graphics only for collision
STATIC_BRICKS	13	not implemented
STATIC_TENT	14	traffic works tent
STATIC_PIT	15	not implemented
STATIC_STONE	16	not implemented
STATIC_PLANT	17	not implemented
STATIC_PILLAR	18	street pillar object
STATIC_TREE	19	large tree, use values $1, .., 5$ for different types
STATIC_UNEVEN	20	uneven road surface (covers whole street width)
STATIC_HUMP	21	regular marked street hump (covers whole street width)
STATIC_SPEEDREDUCE	22	no graphics, value is speedlimit in km/h.
STATIC_TRAFFICLIGHT	23	traffic light (visible or logical)
STATIC_TRAFFICLIGHTSLAVE	24	pure display
STATIC_INVISIBLE_BLOCK	27	invisible collision block. The size is xx dm x yy dm x zz dm for value = xxyyzz. xx = 00 is the same as xx = 10. If inserted with offset, the driver will collide, but other vehicles will ignore the block.
STATIC_MANDATORYSIGN	29	Circular signs, usually with red border. Use sign values $1 \leq t \leq 50$ (c.f. appendix A.7.2) to determine the typeï¿½Check ogfx\Land$n$\F10-Mat-Schild-Informational.dds for for the meaning of unnamed mandatory signs. Only named signs are guaranteed to exist in future versions.ï¿½.
STATIC_INFORMATIONALSIGN	30	Rectangular sign, usually blue or green. Use sign values $1 \leq t \leq 50$ (c.f. appendix A.7.3) to determine the typeï¿½Check ogfx\Land$n$\F10-Mat-Schild-Informational.dds for for the meaning of unnamed info signs. Only named signs are guaranteed to exist in future versions.ï¿½.
STATIC_SMALLINFORMATIONALSIGN	31	Smaller rectangular sign (quadratic). Use sign values $1 \leq t \leq 50$ (c.f. appendix A.7.4) to determine the typeï¿½Check ogfx\Land$n$\F10-Mat-Schild-Informational.dds for for the meaning of unnamed info signs. Only named signs are guaranteed to exist in future versions.ï¿½.
STATIC_LARGEINFORMATIONALSIGN	32	Large informational sign with two signposts. Use sign values $1 \leq t \leq 50$ to determine the typeï¿½Check ogfx\Land$n$\BigInfoSign$x$.bmp for the meaning of type xï¿½.
STATIC_WARNINGSIGN	33	Triangular sign, usually with red border. Use sign values $1 \leq t \leq 50$ (c.f. appendix A.7.1).

Name	Value	Comments
STATIC_STOP_LINE	34	Line painted on the ground (or just pretended), where cars should stop. The value determines the type of line. See **STOPLINE_PRETENDED ...**. The visual effect depends on the country.
STATIC_PYLON	35	Pylon
STATIC_PYLON_EXTENDED	36	
STATIC_CROSSWALK	38	Pedestrian crossing marks, painted on the ground. The value determines the type of line. See **CROSSWALK_NORMAL ...**. The visual effect depends on the country.
STATIC_TUNNELMARKER	39	Mark a tunnel. Position at center of road. The value is a mask with the following bit meanings: 0000000srrrrrrrrgggggggbbbbbbbb, where s means "tunnel starts here", and r,g,b are the bits of an illumination value inside the tunnel.
STATIC_HDRMARKER	40	Mark a HDR influenced region eg. a Tunnel or a region with deep set sun that infuence HDR Exposure: 0000000saaaaaaaaddddddddxxxxxxxx, where s as above, a means due to ambient light, d means direction dependent light portion.
STATIC_ROADHOLE	41	Roadhole
STATIC_BILLBOARD	42	The value determines the type of billboard. See **BILLBOARD_STANDING ...**. An added number (1…10) determines the picture on the billboard.
STATIC_SOLIDLINE	43	logical solid line (only marker, no graphics)
STATIC_COW	44	Animated cow. The value is a mask with the following bit meanings: ooooooaaaaaaaaaatttt, where t is the type of the cow, a the animation sequence (see **ANISEQ_...**) and o the offset in seconds in the sequence where the animation starts.
STATIC_PEDESTRIAN	45	Animated pedestrian. The value is a mask with the following bit meanings: ooooooaaaaaaaaaatttt, where t is the type of the pedestrian, a the animation sequence (see **ANISEQ_...**) and o the offset in seconds in the sequence where the animation starts.
STATIC_INDEPENDENT_LANE	46	next junction handled as normal road, also inhibits lane change on multilane roads
STATIC_ROADTYPE	47	marks the whole road (in both directions) as being of the specified type, c.f. **ROADTYPE_...**
STATIC_ROADSPEED	48	specifies default speed limit on the whole road. This is a hint for newly inserted vehicles, which have not seen a speed limit sign yet.
STATIC_MOTORWAYSIGN	49	signs at motorway, see **MOTORWAYSIGN_...**
STATIC_SWAPCONTAINER	50	swap container
STATIC_COUNTDOWN_MARKER	51	countdown marker on motorway
STATIC_RACELIGHTSIGNALS	52	racing lignts, 5x 4 light traffic lights with strange sorting (value 1,...,5) and two service lane lights (values 6,7). Controlled via **SIGNPROPERTY_LIGHTMASK**. with bits 1=green, 2=yellow, 4=red 1, ..., 64=red 5, 128 = service blue, 256 = service red, 512 = service green
STATIC_RACELIGHTSIGNALS_NOPOST	53	racing lignts without sign posts, see above for details
STATIC_TRAFFICLIGHTSLAVE_NOPOST	54	pure display, no sign post
STATIC_ROADMARK_ARROW	55	direction arrow on the ground. See **ROADMARKARROW_L ...** for the arrow type.
STATIC_ROADMARK_BUSLANE	56	bus lane marker on the ground (not the sign)
STATIC_STORAGE_RACK	57	a storage rack that is used for physics simulation and can be equipped with boxes, see **STORAGERACK_...**
STATIC_WAREHOUSE_PARTS	58	parts for warehouse building, see **WAREHOUSE_...**
STATIC_HEAVYTRUCK_OPEN	59	a static truck with opened compartment that can be loaded (driven into), use value=1 for a damaged version
STATIC_LOADINGRAMP	60	a loading ramp to allow a vehicle to drive over a gap. Use value=1 for the active version
STATIC_BLOCKAHEAD	61	a virtual sign blocking the road ahead vor $value$ meters if some other vehicle is there (used for serpentines). Acts like a traffic light turning red iff there is someone on the relevant road piece.

Name	Value	Comments
STATIC_WATERPLANE	62	a water surface - for realistic results you must ensure that the borders are below surface level. The value is xz where the width is $10(x+1)m$ and the length $10(z+1)m$. The standard height is $10cm$ above road level, but can be adjusted with graphical offset.
STATIC_BEACON	63	traffic beacons: values 1, 2 and 3 have different light patterns. 4, 5 and 6 have the form of a **STATIC_PYLON** with a lamp on top.
STATIC_SMALLINFORMATIONALADDON	64	Like **STATIC_SMALLINFORMATIONALSIGN**, but without the sign post and lower (in order to be mounted underneath a **STATIC_WARNINGSIGN**). Use sign values $1 \leq t \leq 50$ (c.f. appendix A.7.4) to determine the typeï¿½Check ogfx\Land$n$\F10-Mat-Schild-Informational.dds for for the meaning of unnamed info signs. Only named signs are guaranteed to exist in future versions.ï¿½.
STATIC_BEGIN_TRANSACTION	66	Inserting this begins a transaction, i.e. a block of sign insertions. Grouping sign insertions into transactions is much faster than single insertions.
STATIC_END_TRANSACTION	67	Finishes a transaction. Don't forget to insert this at the end of the transaction, or strange things might happen.
STATIC_BEGIN_SPEEDUP	68	Inserting this begins a general transaction, i.e. more commands are allowed per round than would otherwise be.
STATIC_END_SPEEDUP	69	Finishes a general transaction. Don't forget to insert this at the end of the general transaction, or strange things might happen.

Functions: Traffic_SetSigns

A.7.1 Values for STATIC_WARNINGSIGN

Name	Value
WARNINGSIGN_ONCOMING_TRAFFIC	1
WARNINGSIGN_TWO_WAY_TRAFFIC_CROSSING	2
WARNINGSIGN_CHILDREN_CROSSING	3
WARNINGSIGN_DRAWBRIDGE	4
WARNINGSIGN_QUAI	5
WARNINGSIGN_DOUBLE_BEND_RIGHT	6
WARNINGSIGN_LOOSE_GRAVEL	7
WARNINGSIGN_CYCLISTS	8
WARNINGSIGN_CATTLE	9
WARNINGSIGN_LIGHT_SIGNALS	10
WARNINGSIGN_AIR_FIELD	11
WARNINGSIGN_STRONG_CROSS_WIND	12
WARNINGSIGN_EQUAL_RIGHTS_Y_JUNCTION	13
WARNINGSIGN_MINOR_CROSS_ROAD	14
WARNINGSIGN_MINOR_ROAD_LEFT	15
WARNINGSIGN_MINOR_ROAD_RIGHT	16
WARNINGSIGN_MINOR_ROADS_RIGHT_LEFT	17
WARNINGSIGN_MINOR_ROADS_LEFT_RIGHT	18
WARNINGSIGN_MINOR_ROAD_JOINS_RIGHT	19
WARNINGSIGN_MINOR_ROAD_JOINS_LEFT	20
WARNINGSIGN_MINOR_ROAD_BRANCHES_RIGHT	21
WARNINGSIGN_MINOR_ROAD_BRANCHES_LEFT	22
WARNINGSIGN_MAJOR_T_AHEAD	23
WARNINGSIGN_MINOR_ROAD_LEFT_IN_T	24
WARNINGSING_NARROW_PASSAGE	25
WARNINGSIGN_MINOR_ROAD_LEFT_IN_RIGHT_BEND	26
WARNINGSIGN_TRAFFIC_LIGHTS_AHEAD	27
WARNINGSIGN_BRIDGE	28

Name	Value
WARNINGSIGN_RISK_OF_GROUNDING	29
WARNINGSIGN_TUNNEL	30
WARNINGSIGN_HIGH_VOLTAGE_LINE	31
WARNINGSIGN_NO_SHOULDER	32
WARNINGSIGN_RISK_OF_ICE	33
WARNINGSIGN_LEFT_BEND	34
WARNINGSIGN_RIGHT_BEND	35
WARNINGSIGN_STEEP_DESCENT	36
WARNINGSIGN_UNEVEN_ROAD	37
WARNINGSIGN_STEEP_ASCENT	38
WARNINGSIGN_DOUBLE_BEND_LEFT	39
WARNINGSIGN_SLIPPERY_ROADS	40
WARNINGSIGN_GENERAL_HAZARD	41
WARNINGSIGN_YIELD_RIGHT	42
WARNINGSIGN_WILD_ANIMAL_CROSSING	43
WARNINGSIGN_ZEBRA_CROSSING	44
WARNINGSIGN_OTHER_DANGERS	45
WARNINGSIGN_ROADWORKS	46
WARNINGSIGN_RAILWAY	47
WARNINGSIGN_NARROWING_ROAD	48
WARNINGSIGN_NARROWING_ROAD_ONESIDED	49

A.7.2 Values for STATIC_MANDATORYSIGN

Name	Value
MANDATORYSIGN_NO_U_TURN	6
MANDATORYSIGN_MAXHEIGHT_2_5_M	18
MANDATORYSIGN_MAXHEIGHT_3_8_M	20
MANDATORYSIGN_ZONE_50_BEGIN	40
MANDATORYSIGN_ZONE_50_END	41

A.7.3 Values for STATIC_INFORMATIONALSIGN

Name	Value
INFORMATIONSIGN_GASSTATION	9
INFORMATIONSIGN_PARKING_RIGHT	13
INFORMATIONSIGN_PARKING_LEFT	14
INFORMATIONSIGN_TUNNEL	18
INFORMATIONSIGN_ADVANCE_DIRECTION_SIGN	20
INFORMATIONSIGN_PARTIAL_SIDEWALK_PARKING_LEFT	22
INFORMATIONSIGN_PARTIAL_SIDEWALK_PARKING_RIGHT	23
INFORMATIONSIGN_FULL_SIDEWALK_PARKING_LEFT	24
INFORMATIONSIGN_FULL_SIDEWALK_PARKING_RIGHT	25

A.7.4 STATIC_SMALLINFORMATIONALSIGN Values

Name	Value	Comments
SMALLINFOSIGN_PARKING	1	
SMALLINFO_ONEWAY	8	one-way street straight ahead, for countries where sideways pointing signs are unusual (e.g. Switzerland)
SMALLINFO_FLOOD	10	text addon for flooded roads
SMALLINFOSIGN_NO_TURN_ON_RED	19	only exists in countries, where turning right at traffic lights is generally allowed (e.g. USA)

A.7.5 Values for STATIC_TURNING

Name	Value
SIGNTURN_LEFT	2
SIGNTURN_RIGHT	8
SIGNTURN_LEFTRIGHT	10
SIGNTURN_STRAIGHT	4
SIGNTURN_STRAIGHT_LEFT	6
SIGNTURN_STRAIGHT_RIGHT	12

A.7.6 Values for STATIC_STOP_LINE

Name	Value	Comments
STOPLINE_PRETENDED	0	pretended stop line (Sichtlinie)
STOPLINE_YIELD	1024	stop line as found with yield signs
STOPLINE_STOP	2048	stop line as found with stop signs
STOPLINE_TRAFFIC_LIGHT	4096	stop line as found with traffic lights
STOPLINE_ROUNDABOUT	8192	stop line as found at roundabout entries

A.7.7 Values for STATIC_ROADMARK_ARROW

Name	Value	Comments
ROADMARKARROW_L	1	left only
ROADMARKARROW_SL	2	left and straight on
ROADMARKARROW_S	3	straight on
ROADMARKARROW_SR	4	right and straight on
ROADMARKARROW_R	5	right only
ROADMARKARROW_HL	6	half left (presort mark)
ROADMARKARROW_HR	7	half right (presort mark)

A.7.8 Values for STATIC_CROSSWALK

Name	Value	Comments
CROSSWALK_SIMPLE	0	old implementation with single width *deprecated*
CROSSWALK_NORMAL	4096	standard pedestrian crosswalk without traffic lights
CROSSWALK_TRAFFIC_LIGHT	8192	pedestrian crossing with traffic lights (two dash lines)

A.7.9 Values for STATIC_BILLBOARD

Name	Value	Comments
BILLBOARD_STANDING	64	Check ogfx\Common\BB_stand_x.jpg for the meaning of the added number x.
BILLBOARD_WALL	128	Check ogfx\Common\BB_wall_x.jpg for the meaning of the added number x.
BILLBOARD_MEGALIGHT	256	Check ogfx\Common\BB_megalight_x.jpg for the meaning of the added number x.
BILLBOARD_PILLAR	512	Check ogfx\Common\pillar_gs_x.jpg for the meaning of the added number x.
BILLBOARD_CITYLIGHTPILLAR	1024	Check ogfx\Common\pillar_cls_x.jpg for the meaning of the added number x.
BILLBOARD_CITYLIGHTPOSTER	2048	Check ogfx\Common\BB_clp_x.jpg for the meaning of the added number x.
BILLBOARD_BUSSTOP	4096	Check ogfx\Common\BB_Bus_x.jpg for the meaning of the added number x.
BILLBOARD_MEGALIGHT_ANIMATED	8192	Check ogfx\Common\BB_megalight_anim_x.jpg for the meaning of the added number x.
BILLBOARD_CITYLIGHTPILLAR_ROUND	16384	Check ogfx\Common\pillar_cls_rd_x.jpg for the meaning of the added number x.
BILLBOARD_TOPTRONIC	32768	Check ogfx\Common\bb_top_x.jpg for the meaning of the added number x.
BILLBOARD_STANDING_330	65536	Check ogfx\Common\BB_stand_330_x.jpg for the meaning of the added number x.

Name	Value	Comments
BILLBOARD_MEGALIGHT_320	131072	Check ogfx\Common\BB_megalight_320_x.jpg for the meaning of the added number x.
BILLBOARD_CITYLIGHTPOSTER_0985	262144	Check ogfx\Common\BB_clp_0985_x.jpg for the meaning of the added number x.
BILLBOARD_STANDING_330_ANIMATED	524288	Check ogfx\Common\BB_stand_330_anim_x.jpg for the meaning of the added number x.
BILLBOARD_MEGALIGHT_320_ANIMATED	1048576	Check ogfx\Common\BB_megalight_320_anim_x.jpg for the meaning of the added number x.
BILLBOARD_CITYLIGHTPOSTER_0985_ANIMATED	2097152	Check ogfx\Common\BB_clp_0985_anim_x.jpg for the meaning of the added number x.
BILLBOARD_BLOWUP	4194304	Check ogfx\Common\BB_blowup_x.jpg for the meaning of the added number x.
BILLBOARD_LANDSCAPE_12x4	8388608	Check ogfx\Common\BB_FL_12x4_x.jpg for the meaning of the added number x.
BILLBOARD_LANDSCAPE_6x3	16777216	Check ogfx\Common\BB_FL_6x3_x.jpg for the meaning of the added number x.
BILLBOARD_LANDSCAPE_12	33554432	Check ogfx\Common\BB_FL_12_x.jpg for the meaning of the added number x.

A.7.10 Values for SIGNPROPERTY

Name	Value	Comments
SIGNPROPERTY_UV_ANIMATION	1	parameters are: uv-coordinates of upper left corner $[0..1, 0..1]$, uv-coordinates of lower right corner $[0..1, 0..1]$, time for transition in seconds
SIGNPROPERTY_SIZE_ANIMATION	2	parameters are: scaling factor in x-direction, scaling factor in y-direction, time for transition in seconds
SIGNPROPERTY_LIGHTMASK	3	bit mask for active lights on the specified sign (e.g. racing signal lights), may also affact other signs/different values

Functions: Traffic_SetSignProperty

A.7.11 Values for STATIC_MOTORWAYSIGN

Name	Value	Comments
MOTORWAYSIGN_DIRECTION_2LANES	1	overhead gantry sign with direction information for two lanes
MOTORWAYSIGN_DIRECTION_3LANES	2	overhead gantry sign with direction information for three lanes
MOTORWAYSIGN_DIRECTION_2LANES_PARKING	3	overhead gantry sign with direction information for two lanes, motorway station at next exit

A.7.12 Values for STATIC_COUNTDOWN_MARKER

Name	Value
COUNTDOWN_MOTORWAY_100	0
COUNTDOWN_MOTORWAY_200	1
COUNTDOWN_MOTORWAY_300	2

A.7.13 Values for STATIC_ROADTYPE

Name	Value	Comments
ROADTYPE_UNDEFINED	0	
ROADTYPE_RURAL	1	road type: rural road
ROADTYPE_TOWN	2	road type: inside city borders
ROADTYPE_MOTORWAY	3	road type: motorway
ROADTYPE_PATH	4	road type: small road, usually not tarmaced and just a single lane
ROADTYPE_ONEWAY	5	road type: one-way street (special form of inner city road)
ROADTYPE_MOTORWAY_LIKE	6	road type: at least two lanes per direction, barrier between directions
ROADTYPE_TYPE_MASK	15	AND with the type to strip off flags
ROADTYPE_FLAG_NO_PASSING	16	flag: ban on passing (e.g. continuous line)
ROADTYPE_FLAG_GRAVEL	32	flag: ground type is gravel
ROADTYPE_FLAG_PARKING_LANE	64	flag: the road has a parking lane
ROADTYPE_FLAG_SIDEWALK	128	flag: the road has a sidwalk

A.8 Speed Setting Modes

Name	Value	Comments
SETSPEED_WISH	1	Set a speed wish. Signs will be ignored.
SETSPEED_IMMEDIATE	2	Sets the speed immediately. Signs are ignored.
SETSPEED_WISH_RELATIVE	3	Sets the speed wish relative to current speed. Signs will be ignored.
SETSPEED_IMMEDIATE_RELATIVE	4	Immediately changes the speed. Signs will be ignored.
SETSPEED_FLAG_OBEY_LIMITS	4	Add this to the above to (continue to) obey speed limit signs (also c.f. convenience defines below)
SETSPEED_FLAG_IMMEDIATE_LOAD_CHANGE	8	Add this to allow immediate load changes (beware of braking light flicker)
SETSPEED_FLAG_FORCE_PARKING	16	Add this to cause a car with speed 0 to park (even when on a driving lane)
SETSPEED_FLAG_FORCE_HALTING	32	Add this to cause a car with speed 0 not to park (even when on a driving lane)
SETSPEED_WISH_OBEY	5	Like **SETSPEED_WISH**, but adhere to speed limits
SETSPEED_IMMEDIATE_OBEY	6	Like **SETSPEED_IMMEDIATE**, but adhere to speed limits. The current speed is altered even if the result is illegal.
SETSPEED_WISH_RELATIVE_OBEY	7	Like **SETSPEED_WISH_OBEY**, but speed relative to current.
SETSPEED_IMMEDIATE_RELATIVE_OBEY	8	Like **SETSPEED_IMMEDIATE_OBEY**, but speed relative to current.

Functions: Traffic_SetSpeed

A.9 Track Setting Modes

Name	Value	Comments
SETTRACK_IMMEDIATE	0	change track immediately (jump)
SETTRACK_WISH	1	change lane to reach target track
SETTRACK_NOBLINK	2	same as **SETTRACK_WISH**, but without blinker
SETTRACK_NEITHER_STRETCH_NOR_BLINK	3	same as **SETTRACK_NOBLINK**, but no speed dependent lane change stretch

Functions: Traffic_SetTrackDeviation

A.10 Enhanced Reality Constants

Name	Value	Comments
ENRE_DISTANCE_ARROW	0	an arrow visualizing the distance to the next vehicle ahead of us.
ENRE_ECO_SHELL_DIAG	1	a shell diagram with moving cursor demarking the current motor working point.
ENRE_VIRTUAL_CRATE	2	a freely movable wooden crate.
ENRE_DIRT_GLASS	3	a visual overlay on the driver windshield glass.
ENRE_PARKMARKER_1	4	a bar to mark parking places
ENRE_PARKMARKER_2	5	same as above, different default color
ENRE_ALLOW_HELICOPTER	6	allow switching to an above helicopter perspective during ride.
ENRE_TANK_SCHEME	7	a schematic view of the tanker truck with fill level and slosh visualization
ENRE_FORCES_DISPLAY	8	a schematic view of the vehicle with longitudinal and transversal forces
ENRE_REDGREEN_BAR	9	a bar with red/green color distribution and moving arrow
ENRE_MARKER_BALL	10	a virtual ball (by default signalling orange).
ENRE_MARKER_BALL_1	10	
ENRE_MARKER_BALL_2	11	
ENRE_MARKER_BALL_3	12	
ENRE_MARKER_BALL_4	13	
ENRE_MARKER_BALL_5	14	
ENRE_MARKER_BALL_LAST	14	
ENRE_MARKER_CIRCLE	15	
ENRE_MARKER_CIRCLE_1	15	
ENRE_MARKER_CIRCLE_2	16	
ENRE_MARKER_CIRCLE_3	17	
ENRE_MARKER_CIRCLE_4	18	
ENRE_MARKER_CIRCLE_5	19	
ENRE_MARKER_CIRCLE_LAST	19	
ENRE_PIP_1	20	a Picture-in-Picture overlay, use ENREPARM_PIPCAM, ENREPARM_SCALE and ENREPARM_PIPPAN
ENRE_PIP_2	21	
ENRE_PIP_3	22	
ENRE_PIP_4	23	
ENRE_PIP_5	24	
ENRE_PIP_6	25	
ENRE_PIP_7	26	
ENRE_PIP_8	27	
ENRE_PIP_9	28	

Name	Value	Comments
ENRE_PIP_10	29	
ENRE_PIP_11	30	
ENRE_PIP_LEFTVIEW	20	
ENRE_PIP_RIGHTVIEW	21	
ENRE_PIP_REARCAM	22	
ENRE_PIP_LEFTMIRROR	23	
ENRE_PIP_RIGHTMIRROR	24	
ENRE_PIP_LEFTMIRROR_AMPLIFICATION	25	
ENRE_PIP_RIGHTMIRROR_AMPLIFICATION	26	
ENRE_PIP_LEFTMIRROR2	27	
ENRE_PIP_RIGHTMIRROR2	28	
ENRE_PIP_USER	29	
ENRE_PIP_LAST	30	
ENRE_REARVIEW_CAM	31	
ENRE_AUTOSIDEVIEW	32	
ENRE_USER_CAM	33	allows control of the CAMERA_USER position

Functions: System_EnhancedReality,System_EnhancedRealityParameter

A.10.1 Enhanced Reality Parameters

Name	Value	Comments
ENREPARM_POSITION	0	the road position, at which the feature is shown.
ENREPARM_COLOUR	1	three values: red, green, blue in [0 . . . 1].
ENREPARM_SCALE	2	a magnification scale
ENREPARM_TRANSPARENCY	3	a transparency value (0=solid, 1=invisible)
ENREPARM_OFFSET	4	a cartesian position offset from ground position set with **ENREPARM_POSITION**.
ENREPARM_VEHICLE_BINDING	5	continously update **ENREPARM_POSITION** from given vehicle. Bind to 0 or explicitly set position to unbind.
ENREPARM_DIAGRAMTYPE	6	switch ENRE_ECO_SHELL_DIAG between several diagrams, c.f. **DIAGRAMTYPE_...** below
ENREPARM_SCREENPOS	7	sets on-screen position (left, top, width), each in percent of screen width/height. The red/green bar takes height as fourth argument
ENREPARM_REDGREEN_LIMITS	8	set limits in percent of height (in bottom-up or left-right direction bad until, good from, good to, bad from)
ENREPARM_REDGREEN_POINTER	9	set pointer in percent of height (in bottom-up or left-right direction)
ENREPARM_REDGREEN_SCALE	10	set scale values: start percent, start value, step percent, step value, end percent, unit(string)
ENREPARM_PIPCAM	11	e.g. **CAMERA_FRONT**
ENREPARM_PIPPAN	12	horizontal and vertical offsets to a centered camera image
ENREPARM_HELICOPTER_KEYS	13	which keys toggle helicopter mode? 0 means enter, otherwise list of **INSTR_...** values

A.10.2 Diagram Types

Name	Value	Comments
DIAGRAMTYPE_NONE	0	
DIAGRAMTYPE_ECO	1	diagram collection related to economic driving, including shell diagram with engine operating point
DIAGRAMTYPE_SAFETY	2	diagram collection related to driving safety

A.11 Event Constants

Name	Value	Comments
EVENT_TRUCK	1	truck blocks the path (with warning lights), slowly drives on later
EVENT_PEDESTRIAN_CROSSING	2	pedestrian crosses the street after having stood behind a pillar
EVENT_CHILD_CROSSING	3	child crosses the street after having stood behind a pillar
EVENT_DEER_RIGHT	4	deer jumps onto the road from behind a tree, leaves it again
EVENT_PARK_DOOR	5	a parking car opens its door
EVENT_CHILD_BALL_PILLAR	6	a child stands behind a pillar and follows its ball onto the street
EVENT_CHILD_BALL	7	a child stands behind a car and follows its ball onto the street
EVENT_PARK_DOOR_WITHHINT	8	a parking car opens its door after having flashed its braking lights
EVENT_PARK_START	9	a parking car suddenly enters the road
EVENT_JOGGERRUN	11	a jogger enters the road and leaves it later on
EVENT_CYCLIST	12	a cyclist enters the road and leaves it later on
EVENT_CAR	13	a car enters the road and leaves it later on
EVENT_DEER	15	deer jumps onto the road, leaves it again (no tree)
EVENT_ROADWORKMACHINE	17	a roadwork machine blocks the path
EVENT_DEER_LEFT	18	deer, same as **EVENT_DEER_RIGHT**, but from the left
EVENT_CHILD_BALL_LEFT	19	child with ball, same as **EVENT_CHILD_BALL**, but from the left
EVENT_AUDIOSIGNAL	20	a sound is played, the driver is supposed to press enter
EVENT_STOPSIGN	21	a stop sign appears, driver is supposed to brake
EVENT_DONKEY	22	a donkey appears, driver is supposed to brake
EVENT_CAMEL	23	a camel appears, driver is supposed to brake
EVENT_DONKEYCROSSING	24	a donkey crosses the road
EVENT_BROKEN_RADIATOR	25	the temperature rises until the motor overheats, driver is supposed to stop the car.
EVENT_OIL_LEAKAGE	26	the oil lamp fires. driver is supposed to stop the car. after max. 90s the motor is broken.
EVENT_TIRE_RUPTURE	27	the steering gets a sudden offset like from ruptured tyre. the driver should halt without accident.

Functions: Traffic_SetEvent

A.11.1 Event Triggering Modes

Name	Value	Comments
EVENTMODE_CONST_DISTANCE	1	use specified distance from driver to event
EVENTMODE_CONST_REACTION	2	adjust distance based on current speed to yield $1.5s$ reaction time
EVENTMODE_CONST_SPEED	3	same, but based on speed limit
EVENTMODE_CONST_REACTION_VARIABLE	4	same, but offset is reaction time in s (0.1 ... 4.0)

A.11.2 Event Option Bits

Name	Value	Comments
EVENTOPT_NORMAL	0	
EVENTOPT_NO_BRAKING_LIGHTS	1	braking lights hidden
EVENTOPT_NO_BLINKING_LIGHTS	2	blinker lights hidden
EVENTOPT_EARLY_WARN	4	extra warning by early blinker activity
EVENTOPT_PARK_AGAIN	8	let the object run and then park again
EVENTOPT_EARLY_RELEASE	16	stop the event control early, let objects go on uncontrolled
EVENTOPT_NO_MEASUREMENT	256	neither display nor log the event's reaction measurement

Remarks: EVENTOPT_ bitmask values only work for some event types.

A.12 Weather Constants

Name	Value	Comments
WEATHER_NICE	0	Daytime, slight cloud coverage
WEATHER_FOG	1	Daytime, adjustable fog density
WEATHER_RAIN	2	Daytime, cloudy, slight fog and precipitation
WEATHER_SNOW	3	Like WEATHER_NICE, but in winter
WEATHER_NIGHT	4	Night with low rest visibility
WEATHER_TWILIGHT	5	
WEATHER_LOWSUN	6	
WEATHER_SUNNY	7	Daytime, clear sky
WEATHER_SNOWING	8	in contrast to WEATHER_SNOW, it is actually snowing

Functions: World_GetWeather, World_SetWeather

A.13 Weather Parameter Constants

Name	Value	Comments
WEATHERPARM_INVALID	0	
WEATHERPARM_FOG_SIGHT	1	single float: sight distance in m
WEATHERPARM_SANDSTORM_STRENGTH	2	three floats and a flag: wind speed (beaufort), deviation (beaufort), direction angle (degrees), immediate?
WEATHERPARM_ROADS_CLEARED	3	1 for cleared roads, 0 else
WEATHERPARM_RAIN_IN_WINTER	4	1 means WEATHER_SNOWING has rain, not snowfall
WEATHERPARM_THUNDERSTORM_STRENGTH	5	two floats and a flag: strength (0-10), direction angle (degrees), immediate?

Functions: World_GetWeatherParameter, World_SetWeatherParameter

A.14 Road Surface Modes

Name	Value	Comments
ROADMODE_AUTO	-1	automatic from road type and config
ROADMODE_SMOOTH	0	no humps at all
ROADMODE_MOTORWAY_SMOOTH	1	
ROADMODE_ROAD_SMOOTH	2	
ROADMODE_MOTORWAY_UNEVEN	3	
ROADMODE_ROAD_SLIGHTLY_UNEVEN	4	
ROADMODE_ROAD_VERY_UNEVEN	5	
ROADMODE_BUMPY_ROAD	6	very badly tended road

Functions: World_SetRoadMode

A.15 Road Distance Constants

Name	Value	Comments
ROADDIST_LEGALONLY	1	regard signs when searching shortest route
ROADDIST_ENDDIRECTION	2	route must end with same direction as specified
ROADDIST_NOSTARTDIRECTION	4	route can start with different direction than specified

Functions: World_GetRoadDistance

A.16 System Error Constants

Name	Value	Comments
SYSERR_OK	-1	no error
SYSERR_ERROR	-2	generic error
SYSERR_NO_OBJECT	-3	no matching object found
SYSERR_PARTIAL_FUNCTION	-4	the requested function is only partially available
— errors while setting positions —		
SYSERR_ROAD_POSITION	-5	faulty road position
SYSERR_WORLD_POSITION	-6	faulty/inappropriate cartesian position
— errors while inserting vehicles —		
SYSERR_NO_CAR_LEFT	-7	no unused car of the specified type left
SYSERR_NO_SPACE	-8	no space at the specified position
SYSERR_NOT_IMPLEMENTED	-9	unsupported function
SYSERR_NO_SLOT_LEFT	-10	too many objects of involved type active
SYSERR_REFERENCE_ERROR	-11	no or invalid reference object
— errors for event (c.f. ***Traffic_SetEvent****) —*		
SYSERR_WRONG_TYPE	-12	event type unknown or not supported
SYSERR_STILL_RUNNING	-13	other running event prevents creation
SYSERR_DRIVER_NOT_READY	-14	the driver stands still or is braking
SYSERR_INIT_ERROR	-15	event setup failed (e.g. object allocation failure)
— errors for info signs —		
SYSERR_INFOSIGN_INVALID_ID	-16	invalid identifier
SYSERR_INFOSIGN_OUT_OF_RANGE	-17	out of range
— graphics/printer errors —		
SYSERR_GRAPHICERROR	-18	graphics operation failed
— errors concerning the replay subsystem —		
SYSERR_REPLAY_NOTACTIVE	-19	command failed: replay has not been activated
SYSERR_REPLAY_NODATA	-20	command failed due to missing replay data
SYSERR_REPLAY_FILENOTFOUND	-21	couldn't open replay file
SYSERR_REPLAY_NOTAFILM	-22	specified file is not a replay file (wrong header)
SYSERR_REPLAY_READFAILED	-23	crippled replay file
SYSERR_NAVIGATION_INVALID_ID	-24	unknown NAVI instrument
SYSERR_NAVIGATION_OUT_OF_RANGE	-25	value out of range
SYSERR_NAVIGATION_INVALID_DIRECTION	-26	direction unknown
SYSERR_NAVIGATION_INVALID_DIRECTION_ID	-27	direction id unknown
SYSERR_NAVIGATION_INVALID_DISTANCE	-28	distance out of range
SYSERR_DIAGRAM_INVALID_ID	-29	invalid identifier
SYSERR_DIAGRAM_OUT_OF_RANGE	-30	out of range

A.17 Traffic Error Constants

Name	Value	Comments
ERR_OK	0	no error
ERR_FRONTAL	1	frontal collision (float arg: relative velocity in m/s) *NotInUse*
ERR_COLLISION	2	collision from behind (float arg: relative velocity in m/s) *NotInUse*
ERR_ACCIDENT	3	general accident (neither **ERR_FRONTAL** nor **ERR_COLLISION**, float arg: relative velocity in m/s)
ERR_ROAD_BORDER	4	outer street border hit (arg=-1 left, +1 right). Only used on old courses
ERR_RED	5	red traffic light disregarded
ERR_YELLOW	6	yellow traffic light disregarded
ERR_TOO_FAST	7	too fast (according to sign or weather conditions)
ERR_RIGHTOFWAY	8	right of way disregarded
ERR_WITHOUT_LIGHT	9	headlights not turned on (by night or in rain or fog)
ERR_HIGHBEAM	10	highbeam during daytime (not directly blinding)
ERR_BLIND	11	highbeam blinds oncoming traffic
ERR_BLINKER_NOT_SET	12	blinker not used while changing lane (arg=-1 left blinker, +1 right blinker)
ERR_BELT_NOT_FASTENED	13	seatbelt not fastened
ERR_ROAD_BORDER_ACCIDENT	14	accident with road border object (float arg: relative velocity in m/s)
ERR_ANIMAL_ACCIDENT	15	accident with deer or other wildlife (float arg: relative velocity in m/s)
ERR_GEARBOX_SWITCH	16	forbidden change of gear (e.g. reverse chosen while moving forward)
ERR_GEARBOX_DESTROYED	17	gearbox destroyed
ERR_DEVIATION	18	driving with large track deviation for prolonged amounts of time
ERR_ABORT	19	simulation aborted
ERR_TURN_OVER	20	vehicle overturned
ERR_OVERTAKING_RIGHT	21	overtaking on forbidden lane
ERR_SPEED_LIMIT	22	speed limit exceeded (signs only)
ERR_OVERTAKING	23	illegal overtaking (e.g. inside city limits)
ERR_HEADWAY	24	forward clearance too small (half velocity rule, float arg: distance in m)
ERR_TOO_SLOW	25	permanently driving too slow (obstructing traffic)
ERR_LANE_WRONG	26	using wrong lane (lane for faster traffic or traffic headed elsewhere)
ERR_SIDE_WRONG	27	using wrong lane (lane for oncoming traffic)
ERR_SLIDING	28	car drifts (float arg: centrifugal acceleration in m/s^2)
ERR_SLINGING	29	car is skidding
ERR_OVERTAKING_DANGEROUS	30	overtaken without sufficient overview
ERR_CROSSING_TOO_FAST	31	too fast while crossing junction
ERR_AQUAPLANING	32	aquaplaning – tires lost contact
ERR_SLIP	33	car slides
ERR_LATERAL_FORCE	34	centrifugal force too high (arg: would-be acceleration according to wheel position and velocity in g)
ERR_REVOLUTIONS_HIGH	35	motor RPM too high (motor RPM)
ERR_BRAKE_HARD	36	unnecessarily abrupt braking (arg: brake pedal value [0…1])
ERR_ENGINE_STALLED	37	engine stalled
ERR_NO_GRIP	38	wheels slipping (e.g. on snow)
ERR_WHEEL_BLOCK	39	wheels block (e.g. braking on snow)

Name	Value	Comments
ERR_HANDBRAKE_NOT_RELEASED	40	hand brake not released
ERR_LEFT_BRAKE	41	brake pedal hit with left foot (i.e. while still on accelerator pedal, error in reaction time measurement)
ERR_STOP_TOO_EARLY	42	brake hit too early (error in reaction time measurement)
ERR_NOREACTION	43	no reaction at all (error in reaction time measurement)
ERR_RUMBLE	44	car drives beside the road
ERR_NEARBY_ACCIDENT	45	other vehicles were endangered (almost accident)
ERR_WINKER_CROSSING	46	blinker not or incorrectly used at junction
ERR_RIGHT_EDGE_DRIVING	47	driving on lane not meant for traffic (e.g. hard-shoulder on motorway)
ERR_SWITCH_SEQUENCE	48	switched gear out of sequence
ERR_PERFORMANCE_RATING	49	log a fitness number
ERR_CLUTCH_HARD	50	clutch pedal released too fast
ERR_GAP_TOO_LONG	51	gap acceptance scenario: not sufficiently decisive
ERR_GAP_TOO_SHORT	52	gap acceptance scenario: taking too much risk
ERR_WRONG_WAY	53	disregarded exercise directives
ERR_ACCEL_LONGITUDAL_HIGH	54	longitudinal acceleration too high
ERR_SWITCH_TIME	55	took too much time to shift gears
ERR_LATERAL_FORCE_LOW	56	accepted centrifugal force too low (training lesson for curves)
ERR_TIME_REACTION	57	reaction time measurement (arg: time in s)
ERR_REACTION_STOP_DISTANCE	58	measurement of stopping distance (arg: stopping distance in m)
ERR_NO_HIGHBEAM	59	driver could have used high beam, but made no use of it
ERR_SMART_BRAKE	60	driver should step on brake to enter driving gear
ERR_START_GEAR_NOT_NEUTRAL	61	starting impossible because of active gear (not neutral)
ERR_LAP_TIME	62	lap duration (arg: time in s)
ERR_LAP_LENGTH	63	length of a lap
ERR_TRACK_DEVIATION	64	track deviation (0 - 100, scaled like performance rating)
ERR_LINE_MIDDLE	65	solid line crossed
ERR_ABS_ACTIVE	66	ABS activated (instead of blocking)
ERR_STOP_DISTANCE	67	measurement of braking distance (arg: distance in m)
ERR_CONSUMPTION	68	fuel consumption
ERR_DURATION	69	duration of simulation
ERR_LANE_LINE	70	solid line disregarded – drive right!
ERR_TURN_FORBIDDEN	71	forbidden turning manoever (leads to exercise abortion).
ERR_STOPDIST_WITH_ASS	72	braking distance announcement with special text
ERR_STOPDIST_WITHOUT_ASS	73	braking distance announcement with special text
ERR_STOP_SIGN	74	stop sign disregarded
ERR_WRONG_DIRECTION_GHOST	75	heading wrong direction (e.g. on motorway)
ERR_SIDEWALK	76	touched the pavement
ERR_TIME_TO_LINE	77	line touched too fast (time < x seconds, arg time in s)
ERR_TIME_TO_COLLISION	78	extrapolated time to collision with foreign vehicle (arg: time in s)
ERR_OVERTAKE_HEADWAY	79	dropped below security distance while overtaking (overtaking scenarios only, arg: min. time to collision in s)
ERR_ACCELERATOR_RELEASE	80	time until accelerator pedal was released (arg: time in s)
ERR_FUEL_CONSUMPTION_HIGH	81	mean fuel consumtion too high (arg: consumption in $l/100km$)

Name	Value	Comments
ERR_TIME_EXCEEDED	82	time to exercise completion exceeded (arg: time in s)
ERR_SCORE	83	score display
ERR_STOP_FORBIDDEN	84	stopping forbidden (motorway)
ERR_PRESSURE_BRAKE	85	reaction test (arg: max. braking pressure $[0..100]$)
ERR_CONTROL_UNIT	86	marker set with control unit (arg: key pressed: mode, select, enter)
ERR_ROUTE_LOST	87	driver left planned route. route controller turned off.
ERR_ROUTE_ENDS	88	planned destination reached. route controller turned off.
ERR_ROUTE_ERROR	89	route controller turned off due to error
ERR_ROUTE_LONG_WAY	90	driver chose large detour (set by some scenarios)
ERR_ACTIVITY	91	driver distracted (e.g. read the road map while driving)
ERR_TIME_TO_COLLISION_DRIVER	92	oncoming traffic: time to collision < x s. (arg: time in s).
ERR_WRONG_TURN	93	wrong turn taken at junction (ignored sign)
ERR_WRONG_DIRECTION_ONEWAY	94	wrong direction in one-way street
ERR_DESTINATIONREACHED	95	destination reached
ERR_SCENARIOFINISHED	96	scenario ended
ERR_SPEED_MEASURED	97	velocity measurement for braking distance scenarios
ERR_RECOMMENDED_SPEED	98	recommended speed exceeded ($130km/h$ on German autobahn)
ERR_BACKWARDS_FORBIDDEN	99	wrong direction (e.g. reverse gear driving on motorway)
ERR_REVOLUTIONS_TOO_LOW	100	motor RPM too low – danger of choking the motor
ERR_SWITCH_WO_CLUTCH	101	clutch not fully disengaged to switch gear
ERR_MODIFIED_BY_RUNTIME	102	the current scenario was modified by an external (TCP) command (unuseable for statistics)
ERR_END_OF_RIDE	103	the current scenario ends
ERR_EVENT_STARTED	104	an event was started (not necessarily its action)
ERR_EVENT_FINISHED	105	an event finished (usually after a reaction measurement)
ERR_MOTOR_OVERHEATED	106	the motor is boiling (driver didn't react to rising temperature or oil display)
ERR_FAILED_REACTIONMEASURE	107	a reaction measurement failed
ERR_DOOR_NOT_CLOSED	108	(bus) door not closed while driving
ERR_PASSENGER_NOT_PICKED_UP	109	(bus) passenger not picked up
ERR_GAP_TO_ROAD_TOO_WIDE	110	(bus) gap to roadside too wide for boarding
ERR_TRAILER_ACCIDENT	111	collision with own trailer
ERR_TRAILER_TOWBAR_LIMIT	112	touched trailer's towing bar (angle too large)
ERR_ROAD_CONTACT_LOST	113	more than one wheel is off the ground.
ERR_SPEED_RADAR	114	you drove too fast and got a speeding ticket (arg: overspeed in km/h)
ERR_SCENARIOSPECIFIC_1	115	with-values-once: string arg is the error message
ERR_SCENARIOSPECIFIC_2	116	with-values-once: string arg is the error message
ERR_SCENARIOSPECIFIC_3	117	with-values-once: string arg is the error message
ERR_MIRROR_NOT_ADJUSTED	118	mirror was not adjusted properly.
ERR_RPM_HIGH_WRONG_GEAR	119	motor RPM too high due to wrong gear choice (100*old gear + new gear)
ERR_RPM_LOW_WRONG_GEAR	120	motor RPM too low due to wrong gear choice (100*old gear + new gear)
ERR_ENGINE_STALLED_WRONG_GEAR	121	engine stalled due to wrong gear choice (100*old gear + new gear)
ERR_PARKING_NOT_FIRST_GEAR	122	not inserted the 1. gear when parking the car
ERR_PARKING_NO_HANDBRAKE	123	handbrake not activated when parking the car
ERR_PARKING_LIGHT_ON	124	light still switched on when parking the car
ERR_SCENARIOSPECIFIC_4	125	counted error: string arg is the error message

Name	Value	Comments
ERR_SCENARIOSPECIFIC_5	126	counted error: string arg is the error message
ERR_SCENARIOSPECIFIC_6	127	counted error: string arg is the error message
ERR_PYLONHIT	128	counted error: a traffic pylon was hit.
ERR_AVERAGE_RPM	129	average motor RPM
ERR_NUMBER_SHIFTINGS	130	number of shifting operations
ERR_AVERAGE_SPEED	131	average speed [km/h]
ERR_TOTAL_CONSUMPTION	132	total fuel consumption [$liter$]
ERR_ECONOMIC_FIGURE	133	economic figure
ERR_SCENARIOSPECIFIC_7	134	with-values: string arg is the error message
ERR_SCENARIOSPECIFIC_8	135	with-values: string arg is the error message
ERR_SCENARIOSPECIFIC_9	136	with-values: string arg is the error message
ERR_WIPER_NOT_ACTIVATED	137	wiper not used in heavy rain
ERR_ESP_ACTIVE	138	ESC is active
ERR_MEAN_TRACK_DEVIATION	139	mean track deviation in m (no 0 corridor, averaged over distance)
ERR_STDDEV_TRACK_DEVIATION	140	std. deviation of above
ERR_MEAN_TRACK	141	mean track in m (not deviation, measured from road centerline, averaged over distance)
ERR_STDDEV_TRACK	142	std. deviation of above
ERR_RUMBLE_DURATION	143	car has driven beside the road, value is duration in s
ERR_UNCLEAN_TRACK	144	the car is not on a single driving lane
ERR_UNCLEAN_TRACK_DURATION	145	the car was not on a single driving lane (value is duration in s)
ERR_MARKER	146	used to mark times in error logs
ERR_EVENT_TRACK_DEVIATION	147	same as ERR_TRACK_DEVIATION, but for event measurements
ERR_EVENT_MEAN_TRACK_DEVIATION	148	same as ERR_MEAN_TRACK_DEVIATION, but for event measurements
ERR_EVENT_STDDEV_TRACK_DEVIATION	149	same as ERR_STDDEV_TRACK_DEVIATION, but for event measurements
ERR_EVENT_MEAN_TRACK	150	same as ERR_MEAN_TRACK, but for event measurements
ERR_EVENT_STDDEV_TRACK	151	same as ERR_STDDEV_TRACK, but for event measurements
ERR_EVENT_REACTION_WRONG	152	wrong reaction on event
ERR_NUMBER_STOPS	153	number of full stops
ERR_NUMBER_BRAKINGS	154	number of significant brakings (50% brake)
ERR_START_CLUTCH	155	driver should step on clutch before pressing start button
ERR_RUMBLE_COUNT	204	how often the road was left outside event measurements
ERR_RUMBLE_MEAN_DURATION	205	average time for which the road was left
ERR_RUMBLE_STDDEV_DURATION	206	standard deviation of above
ERR_RUMBLE_MIN_DURATION	207	min. time for which the road was left
ERR_RUMBLE_MAX_DURATION	208	max. time for which the road was left
ERR_EVENT_RUMBLE_COUNT	209	same as ERR_RUMBLE_COUNT, but for event measurements
ERR_EVENT_RUMBLE_MEAN_DURATION	210	same as ERR_RUMBLE_MEAN_DURATION, but for event measurements
ERR_EVENT_RUMBLE_STDDEV_DURATION	211	same as ERR_RUMBLE_STDDEV_DURATION, but for event measurements
ERR_EVENT_RUMBLE_MIN_DURATION	212	same as ERR_RUMBLE_MIN_DURATION, but for event measurements
ERR_EVENT_RUMBLE_MAX_DURATION	213	same as ERR_RUMBLE_MAX_DURATION, but for event measurements
ERR_UNCLEAN_TRACK_COUNT	214	how often the road was left outside event measurements
ERR_UNCLEAN_TRACK_MEAN_DURATION	215	average time for which the road was left
ERR_UNCLEAN_TRACK_STDDEV_DURATION	216	standard deviation of above

Name	Value	Comments
ERR_UNCLEAN_TRACK_MIN_DURATION	217	min. time for which the road was left
ERR_UNCLEAN_TRACK_MAX_DURATION	218	max. time for which the road was left
ERR_EVENT_UNCLEAN_TRACK_COUNT	219	same as ERR_UNCLEAN_TRACK_COUNT, but for event measurements
ERR_EVENT_UNCLEAN_TRACK_MEAN_DURATION	220	same as ERR_UNCLEAN_TRACK_MEAN_-DURATION, but for event measurements
ERR_EVENT_UNCLEAN_TRACK_STDDEV_DURATION	221	same as ERR_UNCLEAN_TRACK_STDDEV_-DURATION, but for event measurements
ERR_EVENT_UNCLEAN_TRACK_MIN_DURATION	222	same as ERR_UNCLEAN_TRACK_MIN_DURATION, but for event measurements
ERR_EVENT_UNCLEAN_TRACK_MAX_DURATION	223	same as ERR_UNCLEAN_TRACK_MAX_DURATION, but for event measurements
ERR_NO_SHOULDER_CHECK	224	no shoulder check while changing lane or turning (arg=-1 to the left, +1 to the right). During the ride, this is replaced by either ERR_NO_SHOULDER_CHECK_RIGHT or ERR_NO_SHOULDER_CHECK_LEFT.
ERR_NO_SIDE_GLANCE	225	head not turned to check for traffic (while entering road in a right angle)
ERR_NEITHER_BLINKER_NOR_SHOULDER	226	combination of ERR_NO_SHOULDER_CHECK and ERR_BLINKER_NOT_SET (display only)
ERR_NEITHER_BLINKER_NOR_SIDEGLANCE	227	combination of ERR_NO_SIDE_GLANCE and ERR_BLINKER_NOT_SET (display only)
ERR_NEITHER_XBLINKER_NOR_SHOULDER	228	combination of ERR_NO_SHOULDER_CHECK and ERR_WINKER_CROSSING (display only)
ERR_CROSS_TRAFFIC_CHECK	229	didn't look into side street to check for cross traffic
ERR_NEITHER_XBLINKER_NOR_XCHECK	230	combination of ERR_CROSS_TRAFFIC_CHECK and ERR_WINKER_CROSSING (display only)
ERR_NEITHER_SHOULDER_NOR_XCHECK	231	combination of ERR_CROSS_TRAFFIC_CHECK and ERR_NO_SHOULDER_CHECK (display only)
ERR_NEITHER_XBLINKER_NOR_LOOK	232	combination of ERR_CROSS_TRAFFIC_CHECK, ERR_NO_SHOULDER_CHECK and ERR_WINKER_CROSSING (display only)
ERR_START_WITHOUT_BLINKER	233	blinker not used when starting. Only detected at first start. Use Traffic_SetAI(AI_PARKING) on the driver to reactivate.
ERR_START_WITHOUT_SHOULDER_CHECK	234	no shoulder check while starting, c.f. ERR_START_WITHOUT_BLINKER
ERR_START_NO_BLINKER_NOR_SHOULDER	235	combination of ERR_START_WITHOUT_BLINKER and ERR_-START_WITHOUT_SHOULDER_CHECK (display only)
ERR_NO_SHOULDER_CHECK_LEFT	236	no shoulder check to the left before junction (arg=-1). Display only, counted as ERR_NO_SHOULDER_CHECK
ERR_NO_SHOULDER_CHECK_RIGHT	237	no shoulder check to the right before junction (arg=+1). Display only, counted as ERR_NO_SHOULDER_CHECK
ERR_PILLARHIT	238	counted error: a traffic pillar was knocked over (only when not configured to provoke an accident)
ERR_ENGINE_STALLED_BAD_TIMING	239	scenario specific variant of ERR_ENGINE_STALLED. Use for display only.
ERR_ENGINE_STALLED_STOP_NO_CLUTCH	240	scenario specific variant of ERR_ENGINE_STALLED. Use for display only.
ERR_ENGINE_STALLED_BAD_GEAR_DRIVING	241	scenario specific variant of ERR_ENGINE_STALLED. Use for display only.
ERR_ENGINE_STALLED_BAD_GEAR_START	242	scenario specific variant of ERR_ENGINE_STALLED. Use for display only.

Name	Value	Comments
ERR_ENGINE_STALLED_FAST_CLUTCH	243	scenario specific variant of ERR_ENGINE_STALLED. Use for display only.
ERR_ENGINE_STALLED_BRAKE	244	scenario specific variant of ERR_ENGINE_STALLED. Use for display only.
ERR_ENGINE_STALLED_HANDBRAKE	245	scenario specific variant of ERR_ENGINE_STALLED. Use for display only.
ERR_FAST_PRESSURE_LOADING	246	air was filled at idle gear above 1000 rpm
ERR_COOLDOWN_WARNING	247	a warning to let diesel engines cool down after heavy use.
ERR_SUNKEN	248	car drowned in deep water
ERR_OVERTAKING_LANE	249	too far to the left, message tells to choose the right lane. Also c.f. ERR_LANE_WRONG.

Functions: Error_GetLastError, Error_SetError

A.18 Error Mode Constants

Name	Value	Comments
ERRMODE_IGNORE	0	completely ignore
ERRMODE_DRIVING	1	show during ride
ERRMODE_RESULT	2	show in on-screen evaluation
ERRMODE_PRINT	4	show on printouts
ERRMODE_STATISTIC	8	use for statistics (i.e. logfiles)
ERRMODE_NORMAL	15	**ERRMODE_DRIVING | ERRMODE_RESULT | ERRMODE_PRINT | ERRMODE_STATISTIC**
ERRMODE_KEEPMODE	128	don't change the error mode (ignore above)

Functions: Error_SetErrorMode

A.19 Vehicle Light Constants

Name	Value	Comments
LIGHTCLASS_OFF	2	headlights off and release
LIGHTCLASS_ON	3	force headlights on
LIGHTCLASS_BRAKE_OFF	4	braking lights off and release
LIGHTCLASS_BRAKE_ON	5	force braking lights on
LIGHTCLASS_REAR_OFF	6	rear light off and release
LIGHTCLASS_REAR_ON	7	force rear light on
LIGHTCLASS_BLINK_LEFT_OFF	8	left blinker off and release
LIGHTCLASS_BLINK_LEFT_ON	9	force left blinker on
LIGHTCLASS_BLINK_RIGHT_OFF	10	right blinker off and release
LIGHTCLASS_BLINK_RIGHT_ON	11	force right blinker on
LIGHTCLASS_BACKDRIVE_OFF	12	reverse lights off and release
LIGHTCLASS_BACKDRIVE_ON	13	force reverse lights white
LIGHTCLASS_CONE_OFF	14	light cone off and release
LIGHTCLASS_CONE_PLAIN	15	force light cone on (a geometry near the ground)
LIGHTCLASS_OFF_ALL	17	turn all lights off and release

Name	Value	Comments
LIGHTCLASS_OFF_GEO_ALL	18	turn all lights off, release them and immediately set the geometries.
LIGHTCLASS_OFF_WARN_OFF	19	warning lights off and release
LIGHTCLASS_OFF_WARN_ON	20	force warning lights on
LIGHTCLASS_FORCEOFF	21	force headlights off
LIGHTCLASS_BRAKE_FORCEOFF	22	force braking lights off
LIGHTCLASS_BACK_FORCEOFF	23	force back light off
LIGHTCLASS_BLINK_LEFT_FORCEOFF	24	force left blinker off
LIGHTCLASS_BLINK_RIGHT_FORCEOFF	25	force right blinker off
LIGHTCLASS_BACKDRIVE_FORCEOFF	26	force reverse lights off
LIGHTCLASS_CONE_FORCEOFF	27	force light cone off
LIGHTCLASS_FORCEOFF_ALL	28	force all lights off

Functions: Traffic_SetLights

A.20 Windscreen Wiper Constants

Name	Value
WW_OFF	0
WW_INTERVAL	1
WW_ON	2
WW_FAST	3

Functions: Driver_GetWiper

A.21 Gear Shift Constants

Name	Value	Comments
GS_AUTOMATIC	1	automatic gearbox
GS_HAND_START	2	hand gear shift with automatic start
GS_HAND	3	hand gear shift
GS_SOFTTIP	4	SMART softtip
GS_SOFTTOUCH	5	SMART softtouch
GS_TIPTRONIC	6	Joystick gear box
GS_GASLEFT	7	automatic for disabled: accelerator via the clutch pedal (i.e. left foot)

Functions: Driver_SetGearShift

A.22 Instrument Constants

Name	Value	Comments
INSTR_ACCELERATOR	0	accelerator pedal
INSTR_BRAKE	1	brake pedal
INSTR_CLUTCH	2	clutch pedal
INSTR_HANDBRAKE	3	hand brake
INSTR_STEERING	4	steering wheel
INSTR_TEMPERATURE	8	temperature display
INSTR_OILWARN	9	oil warn lamp (display may be identical with battery warn lamp)
INSTR_WIPER	10	wiper (control)
INSTR_GEAR	11	gear: 0 is idle, -1 reverse
INSTR_GEARTYPE	12	type of gear shift, c.f. appendix A.21
INSTR_STEERINGOFFSET	13	offset to the steering wheel (dynamic, see config)
INSTR_BATTERYWARN	14	battery warn lamp (display may be identical with oil warn lamp)
INSTR_LINEBUTTON	16	control unit “Line”
INSTR_MODEBUTTON	17	control unit “Mode”
INSTR_HORN	18	horn button
INSTR_ENTERBUTTON	19	“Enter” button
INSTR_LOOKRIGHTBUTTON	20	right shift view button (F10PT)
INSTR_LOOKLEFTBUTTON	21	left shift view button (F10PT)
INSTR_AUXINPUT	22	external input
INSTR_USERBUTTON1	23	multi-purpose button on e.g. truck panel. Setting sets the label on the LCD panel (see User Labels **INSTRVAL_ULB_...** below).
INSTR_USERBUTTON2	24	multi-purpose button on e.g. truck panel
INSTR_USERBUTTON3	25	multi-purpose button on e.g. truck panel
INSTR_USERBUTTON4	26	multi-purpose button on e.g. truck panel.
INSTR_USERBUTTON5	27	multi-purpose button on e.g. truck panel
INSTR_USERBUTTON6	28	multi-purpose button on e.g. truck panel
INSTR_MIRRORSWITCH	29	Set to ensure, that a particular mirror is controllableï¿½You have to be prepared to receive **SYSERR_PARTIAL_FUNCTION**. This is not an error, but means that the user can only control the vertical mirror position.ï¿½. Use **INSTRVAL_MIRROR_...**.
INSTR_MIRROR_CENTER_V	30	Vertical angle of center mirror. Taking over this instrument only sets the angle and leaves control with the driver.
INSTR_MIRROR_CENTER_H	31	Horizontal angle of center mirror. See **INSTR_MIRROR_CENTER_H**.
INSTR_MIRROR_LEFT_V	32	See **INSTR_MIRROR_CENTER_H**.
INSTR_MIRROR_LEFT_H	33	See **INSTR_MIRROR_CENTER_H**.
INSTR_MIRROR_RIGHT_V	34	See **INSTR_MIRROR_CENTER_H**.
INSTR_MIRROR_RIGHT_H	35	See **INSTR_MIRROR_CENTER_H**.
INSTR_SAFETYBELT	36	Safety belt fastened? (read only)
INSTR_FUEL_CONSUMPTION	37	consumption in l/100km. Setting this to 0 resets the complete consumption measurement system.
INSTR_FUEL_TOTAL	38	consumption in l since last reset. Setting this to 0 resets the complete consumption measurement system.
INSTR_FUEL_AVERAGE	39	average consumption since last reset. Setting this to 0 resets the complete consumption measurement system.
INSTR_FUEL_SPECIFIC	40	specific consumption in l/kwH. Setting this to 0 resets the complete consumption measurement system.
INSTR_LIGHTSWITCH	41	combination of bits from **INSTRVAL_LIGHT_...**
INSTR_IGNITION	42	bit mask: 1=ignition, 2=starter
INSTR_AVERAGE_RPM	44	
INSTR_NUMBER_SHIFTINGS	45	
INSTR_ECONOMIC_FIGURE	46	(read only)
INSTR_FLASHINGBLUELIGHT	47	flashing blue light (for fire truck only)
INSTR_FUEL_DRIVEN_DISTANCE	50	distance in m driven while active fuel measurement was running. Setting this to 0 resets the complete consumption measurement system.

Name	Value	Comments
INSTR_FUEL_DRIVEN_TIME	51	time in s for which the active fuel measurement is running. Setting this to 0 resets the complete consumption measurement system.
INSTR_FUEL_AVERAGE_SPEED	52	average speed in km/h while active fuel measurement was running. Setting this to 0 resets the complete consumption measurement system.
INSTR_EMERGENCY_LIGHT_SWITCH	53	switch for the emergency lights (used alternatively to user buttons)
INSTR_EMERGENCY_HORN_SWITCH	54	switch for the emergency horn (used alternatively to user buttons)
INSTR_EMERGENCY_LIGHT_BUTTON	55	button for the emergency lights (like user buttons, but fixed function)
INSTR_EMERGENCY_HORN_BUTTON	56	button for the emergency horn (like user buttons, but fixed function)
INSTR_HELPBUTTON	57	button for driver help text/video/audio
INSTR_FORKLIFT_MAST_TILT	58	for tilting entire mast (pitch in Degree -6ï¿½(fore) to +12ï¿½(back), 0ï¿½ is vertical and default).
INSTR_FORKLIFT_FORK_HEIGHT	59	for vertical movement of the forkgroup (in meter, -0.1 to 2.15, default is 0).
INSTR_FORKLIFT_FORK_CENTER	60	for center position of both forks (in meter, -0.5 to 0.5, default/center is 0).
INSTR_FORKLIFT_FORK_WIDTH	61	for width between both forks (in meter).
INSTR_FORKLEVER_TILT	62	Lever for tilting the forklift-truck mast, range [-1...1].
INSTR_FORKLEVER_LIFT	63	Lever for lifting the forklift-truck forks, range [-1...1].
INSTR_FORKLEVER_SHIFT	64	Lever for horizontally shifting the forklift-truck forks, range [-1...1].
INSTR_FORKLEVER_WIDEN	65	Lever for widening the horizontal distance of the forklift-truck forks, range [-1...1].
INSTR_LINEREVERSEBUTTON	66	control unit inverse “Line”
INSTR_MODEREVERSEBUTTON	67	control unit inverse “Mode”
INSTR_FOGLAMP_INDICATOR	68	dashboard indicator rear fog lamp
INSTR_BLINKER	69	bit mask, 1=right, 2=left
INSTR_RIDESTART	70	combination of enter button, starter and accelerator for Realos
INSTR_USERBUTTONA	71	A button
INSTR_USERBUTTONB	72	B button
INSTR_USERBUTTON_MAPPED	73	only for querying: returns true if any user button is pressed, which has been mapped to any of up to 4 given ULB values.

Functions: Driver_GetInstrument, Driver_SetInstrument

A.22.1 Instument Value Constants

Name	Value	Comments
INSTRVAL_MIRROR_CENTER	0	use with **INSTR_MIRRORSWITCH**
INSTRVAL_MIRROR_LEFT	1	use with **INSTR_MIRRORSWITCH**
INSTRVAL_MIRROR_RIGHT	2	use with **INSTR_MIRRORSWITCH**
INSTRVAL_ULB_DEFAULT	0	default function for **INSTR_USERBUTTONx**
INSTRVAL_ULB_ENTER	1	**INSTR_USERBUTTONx** works as enter button
INSTRVAL_ULB_ACC_OFF	0	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_TEMPOMAT_INACTIVE	2	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_ACC_INACTIVE	3	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_TEMPOMAT_ACTIVE	4	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_ACC_ACTIVE	5	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_LGS_OFF	6	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_LGS_ON	7	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_VIEW_LEFT	8	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_VIEW_RIGHT	9	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_USERDEFINED	10	use with **INSTR_USERBUTTONx** to use this button scenario specific
INSTRVAL_ULB_BLUELIGHT_OFF	11	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_BLUELIGHT_ON	12	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_HORN_OFF	13	use with **INSTR_USERBUTTONx**

Name	Value	Comments
INSTRVAL_ULB_HORN_ON	14	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_HELP	15	use with **INSTR_USERBUTTONx** icon in virtpanel only
INSTRVAL_ULB_BUSDOOR_CLOSE	16	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_BUSDOOR_OPEN	17	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_TRUCK_UP	18	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_TRUCK_DOWN	19	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_TRAILER_UP	20	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_TRAILER_DOWN	21	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_SPEEDBUTTON	22	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_RETARDER_BUTTON	23	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_RETARDER_UP	24	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_RETARDER_DOWN	25	use with **INSTR_USERBUTTONx**
INSTRVAL_ULB_NOTHING	26	use with **INSTR_USERBUTTONx**
INSTRVAL_LIGHT_HEADLIGHTS	1	use with **INSTR_LIGHTSWITCH**
INSTRVAL_LIGHT_HIGHBEAM	2	use with **INSTR_LIGHTSWITCH**
INSTRVAL_LIGHT_PARKLIGHT	4	use with **INSTR_LIGHTSWITCH**
INSTRVAL_LIGHT_FOGLAMP	8	use with **INSTR_LIGHTSWITCH**

A.23 Behaviour Profile Constants

Name	Value
CONTROLLER_DEFAULT	0
CONTROLLER_PROFILE_1	1
CONTROLLER_PROFILE_2	2
CONTROLLER_PROFILE_3	3
CONTROLLER_PROFILE_4	4
CONTROLLER_PROFILE_5	5
CONTROLLER_LAST	6

Functions: Traffic_SetProfile

A.24 Behaviour Constants

Name	Value	Comments
BEHAVE_DUMB	0H	absolutely stupid driver
BEHAVE_DRIVER	1H	regards driver's vehicle (as possible obstacle in order not to collide)
BEHAVE_VEHICLES	2H	regards other vehicles (as possible obstacle in order not to collide)
BEHAVE_GIVE_WAY	4H	regards right of way rules
BEHAVE_LIGHTS	8H	regards traffic lights
BEHAVE_SIGNS	10H	regards traffic signs
BEHAVE_SPEED	20H	regards speed limits
BEHAVE_EARLY_BRAKE	100H	brakes early to leave enough space beween own vehicle and vehicle ahead
BEHAVE_JUNCTION_LOGIC	200H	regards objects on junctions (otherwise does not enter nonempty junctions)
BEHAVE_OVERTAKE	400H	able to overtake
BEHAVE_TRACKCHANGE	800H	able to change lane
BEHAVE_WALKER	1000H	regards pedestrians
BEHAVE_OBSTACLES	2000H	regards obstacles
BEHAVE_ZIGZAG	4000H	does drive light zig-zag (more natural)
BEHAVE_END_REVERSE	8000H	object is turned around at the end of the street (otherwise: object is deleted)
BEHAVE_NO_RIGHT_OVERTAKING	10000H	does not overtake other vehicles while on the right lane
BEHAVE_USE_LANES	20000H	use lanes properly
BEHAVE_USE_TIS	40000H	use Traffic-Information-System and look for most ununsed roads (only if not told to use main road)
BEHAVE_TURN_SLOWLY	80000H	drive slowly during turns on crossing and on roundabouts
BEHAVE_ENTER_JUNCTION	100000H	(1L«20) ignore driver's right of way after 10s of total blocking.
BEHAVE_LANE_CHANGE_CARE	200000H	(1L«21) be carefule when doing a lane change and defer operation if necessary.
BEHAVE_LANE_CHANGECARE_RESET	400000H	(1L«22) reset-value for **BEHAVE_LANE_CHANGE_CARE**: 0=not careful, 1 careful next time.
BEHAVE_EMERGENCY_DRIVER_STOP	800000H	(1L«23) Emergency Driver Stop: Stoppt beim Fahrer bevor er auf ihn auffi¿½hrt.
BEHAVE_KEEP_LEGALLANE	1000000H	(1L«24) Driver tries to stick to a legal lane (changes if not on one)
BEHAVE_DODGE_OTHERS	2000000H	(1L«25) look for oncoming vehicles and avoid them by driving to the side of the own lane
BEHAVE_GO_FORWARD	4000000H	(1L«26) unset this bit to go backwards
BEHAVE_ALLOW_LEFT_TURN	8000000H	(1L«27) unset this bit to disallow left turns
BEHAVE_CURVATURE	10000000H	(1L«28) unset this bit to ignore road curvature
BEHAVE_CLEVER	1FFFFFFFH	absolutely "clever" driver, default value

Functions: Traffic_SetBehaviour

A.25 Artificial Intelligence Constants

Name	Value	Comments
AI_REGARD_DRIVER	1	tries not to collide with driver
AI_REGARD_OTHERS	2	tries not to collide with other vehicles
AI_GIVE_WAY	3	regards rights of way
AI_REGARD_LIGHTS	4	regards traffic lights
AI_REGARD_SIGNS	5	regard traffic signs
AI_REGARD_SPEEDLIMIT	6	respect speed limits
AI_SMOOTH_SPEEDCHANGE	7	accelerate and brake smoothly
AI_CAREFUL_SPEEDCHANGE	8	accelerate and brake carefully
AI_EARLY_BRAKE	9	brake early to leave enough space between own vehicle and vehicle ahead
AI_XROADS_OVERVIEW	10	watch more than one other vehicle at junctions
AI_OVERTAKE	11	able to overtake
AI_LANECHANGE	12	able to change lanes
AI_REGARD_PEDESTRIANS	13	regards pedestrians
AI_REGARD_OBSTACLES	14	regards obstacles
AI_ZIGZAG	15	does drive light zig-zag (more natural)
AI_TURN_AT_END	16	turn around when reaching end of street (otherwise: disintegrate)
AI_NO_RIGHT_OVERTAKE	17	refrain from overtaking on the right lane
AI_USE_LANES	18	make proper lane changes instead of just driving around others
AI_AVOID_JAM	19	try to use roads with few traffic (only if neither bound to main road nor having a route)
AI_TURN_SLOWLY	20	decelerate before taking narrow turns
AI_GO_ON_CROSS	21	Enter Junctions (even without priority), only when using **AI_XROADS_OVERVIEW**.
AI_LANE_CHANGE_CARE	22	be careful when doing a lane change and defer operation if necessary.
AI_LANE_CHANGECARE_RESET	23	reset-value for **AI_LANE_CHANGE_CARE**: 0=not careful, 1 careful next time.
AI_EMERGENCY_DRIVER_STOP	24	Emergency Driver Stop: Stoppt beim Fahrer bevor er auf ihn auffï¿½hrt.
AI_KEEP_LEGALLANE	25	Driver tries to stick to a legal lane (changes if not on one)
AI_DODGE_OTHERS	26	look for oncoming vehicles and avoid them by driving to the side of the own lane
AI_GO_BACKWARDS	-27	go backwards (only effective when manual steering has been activated)
AI_ALLOW_LEFT_TURN	28	allow all turns - disable to forbid left turns
AI_REGARD_CURVATURE	29	slow down in narrow curves
AI_BEHAVE_CLEVER	1000	all of the above
AI_BEHAVE_DUMB	-1000	behave dumb
AI_BEHAVE_DUMB_ON_LANE	-999	behave dumb, but don't modify behaviour regarding keeping on lanes.
AI_NO_ZIGZAG	-15	don't drive in small zig-zag lines
AI_EMERGENCY	1001	stop and turn warning lights on. drive on after driver has passed.
AI_OPENDOOR	1002	open car door (for special parking car only)
AI_LEAVEPARKING	1003	parking car is ordered to leave parking and enter the road (cannot be turned off, 1 additional parameter: acceleration in m/s^2)
AI_WALKING	1004	object starts (or stops) walking. Only for pedestrians or similar obects.
AI_STOPPEDANIMATION	1005	set the animation associated to zero velocity (value **ANISEQ_LIE**, ...)
AI_SLOWANIMATION	1006	s.o., but for slow speed
AI_FASTANIMATION	1007	s.o., but for fast speed
AI_TRAFFICLIGHT	1010	Order an object to display traffic light statesï¿½For each green phase it should display, give three extra arguments: A logical traffic light (c.f. **Traffic_SetSigns**), a flag integer and a road number. The green phase is meant for cars coming from the given road (flag=0) or from the given road and the opposite direction (flag=1).ï¿½.
AI_COUPLED	1020	Couple a trailer to a pulling vehicle. Pass the pulling vehicle's id as argument. 0 decouples the trailer.
AI_MOUNTING	1025	Mount a person on a bus. Pass the vehicle to mount id as argument. 0 unmounts the person.
AI_MOUNTPOSITION	1026	Set local position of a mounted person to x,y,z,yaw,pitch,roll

Name	Value	Comments
AI_WANTSPEED	1030	one parameter: speed, the vehicle wants to drive (if nothing hinders it), in km/h
AI_ACCELERATION	1031	one parameter: acceleration for speeding up to desired speed, in m/s
AI_DECELERATION	1032	one parameter: acceleration for slowing down in normal traffic flow (not emergency braking), in m/s
AI_DISTANCEFACTOR	1033	normally, vehicles always keep a distance allowing them to brake within **AI_DECELERATION** limits. The parameter is is a factor on this distance.
AI_MINDISTANCE	1034	one parameter: vehicles try never to drive closer to others than this (in m).
AI_MINDLC	1035	two parameters: the minimal distance to line crossing kept by the vehicle (if possible). first parameter is lateral distance towards oncoming traffic, second to own side of road.
AI_KEEPTTC	1036	one parameter: minimum TTC to keep
AI_REACTIONTIME	1037	one parameter: reaction time.
AI_SETZIGZAG	1038	two parameters: amplitude and offset (in m) of zig-zag-motion.
AI_MANUALSTEERING	1039	one parameter: the manual steering angle (max. 80 degree plus (right) or minus (left)).
AI_PARKING	1040	object is ordered to park immediately (cannot be turned off, 1 additional parameter: track in m)
AI_HALTING	1042	like **AI_PARKING**, but lights stay on

Functions: Traffic_SetAI

A.25.1 Animation Constants

Name	Value	Comments
ANISEQ_NONE	0	uninitialized, don't set
ANISEQ_WALK	1	slowly (or normally) moving/walking animation
ANISEQ_STOP	2	standard stopped animation
ANISEQ_RUN	3	animation for faster movement (running)
ANISEQ_STAND	4	upright standing position
ANISEQ_SIT	5	sitting or relaxed position
ANISEQ_LIE	6	position close to the ground
ANISEQ_POLICE_WAVE	30	Policeman waving traffic over junction
ANISEQ_POLICE_STOP	31	Policeman stopping traffic in front
ANISEQ_POLICE_WHISTLE	32	Policeman whistling for attention
ANISEQ_POLICE_ATTEND1	33	Policeman right arm upright, left arm outstretched
ANISEQ_POLICE_ATTEND2	34	Policeman right arm angled up, left arm down
ANISEQ_POLICE_ATTEND3	35	Policeman right arm down, left arm outstretched
ANISEQ_POLICE_ATTEND4	36	Policeman left and right arm outstreched
ANISEQ_SPECIAL1	50	special animation (meaning depends on the animated object)
ANISEQ_SPECIAL2	51	
ANISEQ_SPECIAL3	52	
ANISEQ_SPECIAL4	53	
ANISEQ_SPECIAL5	54	
ANISEQ_SPECIAL6	55	
ANISEQ_SPECIAL7	56	
ANISEQ_SPECIAL8	57	
ANISEQ_SPECIAL9	58	
ANISEQ_INTERNAL1	90	internal (intermediate) animation sequences, don't use externally
ANISEQ_INTERNAL2	91	
ANISEQ_INTERNAL3	92	
ANISEQ_INTERNAL4	93	
ANISEQ_INTERNAL5	94	
ANISEQ_INTERNAL6	95	

Name	Value	Comments
ANISEQ_INTERNAL7	96	
ANISEQ_INTERNAL8	97	
ANISEQ_INTERNAL9	98	
ANISEQ_INTERNAL10	99	
ANISEQ_INTERNAL11	100	
ANISEQ_INTERNAL12	101	
ANISEQ_INTERNAL13	102	
ANISEQ_INTERNAL14	103	
ANISEQ_INTERNAL15	104	
ANISEQ_INTERNAL16	105	
ANISEQ_INTERNAL17	106	
ANISEQ_INTERNAL18	107	
ANISEQ_INTERNAL19	108	
ANISEQ_INTERNAL20	109	
ANISEQ_INTERNAL21	110	
ANISEQ_INTERNAL22	111	
ANISEQ_INTERNAL23	112	
ANISEQ_INTERNAL24	113	
ANISEQ_INTERNAL25	114	
ANISEQ_INTERNAL26	115	
ANISEQ_INTERNAL27	116	
ANISEQ_INTERNAL28	117	
ANISEQ_INTERNAL29	118	
ANISEQ_INTERNAL30	119	
ANISEQ_HANDS2HIP	130	for **OBJECTPOINTER_WALKER...**
ANISEQ_FOLDARMS	131	for **OBJECTPOINTER_WALKER...**
ANISEQ_WAVE	132	for **OBJECTPOINTER_WALKER...**
ANISEQ_VIEWLEFT	133	for **OBJECTPOINTER_WALKER...**
ANISEQ_VIEWRIGHT	134	for **OBJECTPOINTER_WALKER...**
ANISEQ_VIEWCLOCK	135	for **OBJECTPOINTER_WALKER...**
ANISEQ_COW_BEHAVE_A	136	cow behaviour A for **STATIC_COW**
ANISEQ_COW_BEHAVE_B	137	cow behaviour B for **STATIC_COW**
ANISEQ_COW_BEHAVE_C	138	cow behaviour C for **STATIC_COW**
ANISEQ_CHARACTER_BEHAVE_A	139	character behaviour A for **STATIC_PEDESTRIAN**
ANISEQ_CHARACTER_BEHAVE_B	140	character behaviour B for **STATIC_PEDESTRIAN**
ANISEQ_CHARACTER_BEHAVE_C	141	character behaviour C for **STATIC_PEDESTRIAN**
ANISEQ_OPENBUSDOORS	142	opening doors for **OBJECTPOINTER_URBANBUS**
ANISEQ_CLOSEBUSDOORS	143	closing doors for **OBJECTPOINTER_URBANBUS**
ANISEQ_HANDS2HIP_SIT	144	for **OBJECTPOINTER_WALKER...**
ANISEQ_FOLDARMS_SIT	145	for **OBJECTPOINTER_WALKER...**
ANISEQ_WAVE_SIT	146	for **OBJECTPOINTER_WALKER...**
ANISEQ_VIEWLEFT_SIT	147	for **OBJECTPOINTER_WALKER...**
ANISEQ_VIEWRIGHT_SIT	148	for **OBJECTPOINTER_WALKER...**
ANISEQ_VIEWCLOCK_SIT	149	for **OBJECTPOINTER_WALKER...**
ANISEQ_SITTING	150	for **OBJECTPOINTER_WALKER...**
ANISEQ_RETRACT_SWAPBODYPILLAR	151	for **OBJECTPOINTER_SWAPBODY**
ANISEQ_SWINGOUT_SWAPBODYPILLAR	152	for **OBJECTPOINTER_SWAPBODY**
ANISEQ_LAST	153	last valid + 1

A.26 Sound Constants

Name	Value	Comments
SOUND_MOTOR	0	
SOUND_STARTER	1	
SOUND_CRASH	2	
SOUND_CLINK	3	
SOUND_HORN	4	
SOUND_SCREAM	5	
SOUND_WIND	6	
SOUND_SCRATCH	7	
SOUND_SPRAY	8	
SOUND_RAIN	9	rain during driving
SOUND_SIGNAL	10	signalling sound
SOUND_RUMBLE	11	
SOUND_SINE	12	
SOUND_TURNOVER_SHORT	15	
SOUND_TURNOVER_LONG	16	
SOUND_MBUFFER	17	no sound
SOUND_RAIN_STANDING	18	rain while not driving
SOUND_DOPPLER_ONCOMING	19	
SOUND_DOPPLER_AWAY	20	
SOUND_POWER	21	
SOUND_FOREIGN_HORN	22	
SOUND_LEFT	23	NAVI: turn left
SOUND_RIGHT	24	NAVI: turn right
SOUND_STRAIGHT	25	NAVI: driv straight on
SOUND_SOON_RIGHT	26	NAVI: turn right soon
SOUND_SOON_LEFT	27	NAVI: turn left soon
SOUND_ABS	28	Anti Brake locking System
SOUND_BLINK1	29	
SOUND_BLINK2	30	
SOUND_HUMPUP	31	
SOUND_HUMPDOWN	32	
SOUND_TRUCKBRAKEON	33	
SOUND_TRUCKBRAKERELEASE	34	
SOUND_TRUCKGEARSHIFT	35	
SOUND_TRUCKMOTOR	36	
SOUND_MTRUCKBUFFER	37	no sound
SOUND_TRUCKMOTOR2	38	
SOUND_MTRUCKBUFFER2	39	no sound
SOUND_SIGNAL_NOPE	40	
SOUND_SIGNAL_YEP	41	
SOUND_BUSMOTOR	42	
SOUND_MBUSBUFFER	43	no sound
SOUND_IDLE_BUS	44	
SOUND_GEARBOX_ACCEPT	45	
SOUND_GEARBOX_DECLINE	46	
SOUND_RUMBLESTRIP_LEFT	47	
SOUND_RUMBLESTRIP_RIGHT	48	
SOUND_ACC_DISTANCE_ALERT	49	
SOUND_ACC_COLLISION_WARNING	50	
SOUND_CIRCLE_1ST	51	NAVI: take first exit in roundabout
SOUND_CIRCLE_2ND	52	NAVI: take second exit in roundabout
SOUND_CIRCLE_3RD	53	NAVI: take third exit in roundabout
SOUND_CIRCLE_4TH	54	NAVI: take fourth exit in roundabout

Name	Value	Comments
SOUND_SOON_CIRCLE_1ST	55	NAVI: take first exit in coming roundabout
SOUND_SOON_CIRCLE_2ND	56	NAVI: take second exit in coming roundabout
SOUND_SOON_CIRCLE_3RD	57	NAVI: take third exit in coming roundabout
SOUND_SOON_CIRCLE_4TH	58	NAVI: take fourth exit in coming roundabout
SOUND_TYREBLOW	59	
SOUND_FELGE	60	
SOUND_DOPPLER_FIRST	61	replaces SOUND_DOPPLER_... above for 3D audio
SOUND_DOPPLER_LAST	70	replaces SOUND_DOPPLER_... above for 3D audio
SOUND_FOREIGN_HORN_FIRST	71	replaces SOUND_FOREIGN_HORN above for 3D audio
SOUND_FOREIGN_HORN_LAST	75	replaces SOUND_FOREIGN_HORN above for 3D audio
SOUND_MY_MICROPHONE	76	special sound: you can set its volume to tune REMOTE playback of simulator's microphone
SOUND_REMOTE_MICROPHONE	77	special sound: you can set volume and pan of a remote speaker
SOUND_STORM	78	like wind, but for jerky external wind sound
SOUND_LIFTING	79	forklift fork movement loop
SOUND_LIFTING_END	80	forklift fork end of loop
SOUND_RUMBLEFORK	81	forklift fork slides on ground (looped)
SOUND_BONK	82	forklift bumping with object
SOUND_BLOB	83	forklift load contact sound
SOUND_BING	84	forklift fork-floor contact sound
SOUND_PRESSUREWARN	85	warning sound air pressure level too low
SOUND_REVERSEBEEPER	86	warning sound for others: reverse gear engaged
SOUND_THUNDERBASE	87	background noises in thunderstorm
SOUND_THUNDERSTRUCK	88	foreground noise in thunderstorm

Functions: System_PlaySample

A.27 Traffic Light Constants

Name	Value	Comments
TLST_NONE	-1	don't set phase, just measure distance
TLST_OFF	0	switched off
TLST_RED	1	red
TLST_REDYELLOW	2	red-yellow (displayed as red for some countries)
TLST_GREEN	3	green
TLST_GREEN_OFF	4	green, but currently off (NOT a seperate phase animation-wise)
TLST_YELLOW	5	yellow
TLST_YELLOW_BLINK	6	disabled, warning
TLST_YELLOW_BLINK_OFF	7	same (just currently off)

Functions: Surround_SetLight

A.28 End of Route Behaviour

Name	Value	Comments
ENDROUTE_RANDOM	0	objekt drives on in a random way (default),
ENDROUTE_REMOVE	1	object is removed at its destination
ENDROUTE_STOP	2	object stops (parks) at end of route
ENDROUTE_PARKING	4	object parks at end of road, track selectable
ENDROUTE_LOCK	8	object is locked during the operation (cannot be reassigned)
ENDROUTE_UNLOCK	16	unlock object when route is finished
ENDROUTE_IGNORE_SIGNS	32	ignore traffic signs for determination of route
ENDROUTE_REPEAT_ROUTE	64	object will repeat same route when finished
ENDROUTE_CONTINUE	128	continue a previously set route instead of calculating a new one
ENDROUTE_STAY_ON	256	together with **ENDROUTE_STOP** or **ENDROUTE_PARKING**: don't turn off the car after stopping

Functions: Traffic_SetRoute

A.29 Numeric Constants

Name	Value
NULL	0
TRUE	1
FALSE	0
PI	3.141592

A.30 Unit Conversion Constants

Name	Value	Comments
KILOMETER_PER_MILE	1.609347	
FOOT_PER_METER	3.281	
CENTIMETER_PER_INCH	2.54	
LITER_PER_GALLON_US	3.7854118	US gallon
LITER_PER_GALLON_CDA	4.546	Canadian gallon
KG_PER_POUND	0.4536	
KG_PER_TON_US	907.2	
PSI_PER_BAR	14.22	bar or hPa conversion to pounds per square inch

A.31 User Data Item Constants

Name	Value	Comments
USERDATA_FILENAME	0	string uniquely identifying the driver, useable as a file name. Only use ASCII letters, digits and underscores.
USERDATA_NAME	1	
USERDATA_GENDER	2	
USERDATA_DATEOFBIRTH	3	
USERDATA_COMMENT	4	
USERDATA_DIAGNOSIS	5	
USERDATA_NICKNAME	6	
USERDATA_PASSWORD	7	
USERDATA_CUSTOMDATA	8	
USERDATA_ORIGINAL_ID	9	same as FILENAME, except for chip cards continuing a given user

Functions: System_GetUserData, System_SetUserData

A.32 Report-Menu Line Attribute Bits

Name	Value	Comments
MENUMODIFIER_NEUTRAL	0	normaler Text
MENUMODIFIER_PRESELECTED	1	Text auf dem gerade der cursor steht
MENUMODIFIER_INACTIVE	6	Text wird in Auswahl ï¿½bersprungen, aber sichtbar (grau)
MENUMODIFIER_HIDDEN	10	Text wird in Auswahl ï¿½bersprungen, nicht sichtbar (Leerzeile)
MENUMODIFIER_MESSAGE	258	Text wird in Auswahl ï¿½bersprungen, normal sichtbar, z.B. Hilfszeile
MENUMODIFIER_STATUS	514	Text wird in Auswahl ï¿½bersprungen, normal sichtbar, z.B. Hilfszeile
MENUMODIFIER_WRONG	16	Text ist falsch
MENUMODIFIER_WARNING	48	Text ist falsch, schrï¿½nkt Funktion ein
MENUMODIFIER_EDITLINE	64	Aktive Edit-Zeile, muï¿½ mit "<Label>:" starten
MENUMODIFIER_HIGHLIGHTED	128	Text ist hervorgehoben (gelb)
MENUMODIFIER_CENTERED	512	Text ist Zentriert (nicht links-oder rechtsbï¿½ndig)
MENUMODIFIER_BIT_CURSOR	1	
MENUMODIFIER_BIT_DISABLED	2	
MENUMODIFIER_BIT_GRAYED	4	
MENUMODIFIER_BIT_INVISIBLE	8	
MENUMODIFIER_BIT_WRONG	16	
MENUMODIFIER_BIT_BLOCKER	32	
MENUMODIFIER_BIT_EDITLINE	64	the editable part starts after the first ':', the cursor is always at the logical string end.
MENUMODIFIER_BIT_HIGHLIGHTED	128	
MENUMODIFIER_BIT_LIGHTGRAY	256	
MENUMODIFIER_BIT_CENTERED	512	
MENUMODIFIER_BIT_NEWBLOCK	1024	visually distinguish this text from preceding text (e.g. new box further down)
MENUMODIFIER_BIT_VCENTERBLOCK	2048	really center the center block (else: below top block)
MENUMODIFIER_BIT_ONLY_IN_REPLAY	4096	treat this block as not existent if not in replay

Functions: System_SetReport, System_GetReport

A.33 Report-Menu useage types

Name	Value	Comments
REPORTMENU_OFF	0	empty and inactive menu
REPORTMENU_RESULT	1	standard error report after a ride
REPORTMENU_EVENT	2	information display after reaction events

Functions: System_SetReport, System_GetReport

A.34 Menu State Bits

Name	Value	Comments
MENU_ON	1	menu is displayed
MENU_FORCED	2	menu always displayed
MENU_FORBIDDEN	4	menu locked (overrides the first two)
MENU_RIDEFORBIDDEN	8	the user cannot start a ride him-/herself
MENU_BELATEDFORBIDDENMSG	16	the message about forbidden start of ride is shown slightly belated (in order to give external controller a chance to react)
MENU_NOFORBIDDENMSG	32	the message about forbidden start of ride is not shown
MENU_ALLOWREPLAY	64	if menu is forbidden, nevertheless allow the replay after ride
MENU_NO_EVALTABLE	128	

Functions: System_Menu

A.35 System States

Name	Value	Comments
SYSSTATE_OFF	0	not even initializing
SYSSTATE_INITIALIZING	1	system initializing
SYSSTATE_WAITING	2	waiting for user input (before ride or in menu)
SYSSTATE_STARTING	3	scenario starting up (transitional state)
SYSSTATE_RUNNING	4	scenario is running
SYSSTATE_WILLEND	5	scenario almost finished, about to end
SYSSTATE_END	6	scenario finished (transitional state), is buffered until queried
SYSSTATE_EVALUATION	7	evaluation phase started
SYSSTATE_EVALWAIT	8	from here on evaluation screen can be cancelled
SYSSTATE_EVALEND	9	evaluation finished, back to **SYSSTATE_WAITING** (transitional)
SYSSTATE_EXIT	10	simulator shutting down
SYSSTATE_STATECOUNT	11	number of valid system states (without flags)
SYSSTATE_STATEMASK	31	use to mask out flag bits
SYSSTATE_FLAG_MENUON	32	flag: the menu is visible
SYSSTATE_FLAG_MENUALLOWED	64	flag: is the user allowed to use the menu?
SYSSTATE_FLAG_REPLAY	128	flag: a replay is running
SYSSTATE_FLAG_EXTREPLAY	256	flag: an externally triggered replay is running (control desk)
SYSSTATE_FLAG_EVALUATION	512	flag: the evaluation screen is visible
SYSSTATE_FLAG_BUSY	1024	flag: the program is busy and not able to react fast

Name	Value	Comments
SYSSTATE_FLAG_INACCIDENT	2048	flag: an accident procedure is running
SYSSTATE_LEASE_EXPIRED	4096	flag: the user's time budget is used up and he won't be able to start another ride.

Functions: System_GetStatus

A.36 Camera Perspectives

Name	Value	Comments
CAMERA_DEFAULT	-1	the default view for the given camera (as set by configuration file)
CAMERA_OFF	0	no view selected
CAMERA_FRONT	1	normal driver perspective
CAMERA_FRONT2	2	driver perspective, wide angle
CAMERA_BACK	3	rear view
CAMERA_BACK2	4	rear view, wide angle
CAMERA_LEFT	5	left view in 3 screen setups
CAMERA_RIGHT	6	right view in 3 screen setups
CAMERA_LEFT2	7	left view
CAMERA_RIGHT2	8	right view
CAMERA_FREE	9	adjustable camera position
CAMERA_REPLAY	10	replay camera
CAMERA_REPLAY_NOOVL	11	replay camera without the front overlay (in particular without the usage hints)
CAMERA_TOP	12	top view camera
CAMERA_TOP2	13	2nd top view camera
CAMERA_BLACK	14	black screen
CAMERA_SAVER1	15	screensaver view 1
CAMERA_SAVER2	16	screensaver view 2
CAMERA_SAVER3	17	screensaver view 3
CAMERA_SAVER4	18	screensaver view 4
CAMERA_SAVER5	19	screensaver view 5
CAMERA_MIRROR_MID	20	main mirror, middle position
CAMERA_MIRROR_MIDR	21	main mirror, right position
CAMERA_MIRROR_LEFT	22	left mirror
CAMERA_MIRROR_RIGHT	23	right mirror
CAMERA_MIRROR_SIDE	24	side view
CAMERA_COCKPIT_FRONT	25	cockpit front view
CAMERA_COCKPIT_LEFT	26	cockpit left view
CAMERA_COCKPIT_RIGHT	27	cockpit right view
CAMERA_COCKPIT_LEFTMOST	28	cockpit far left
CAMERA_COCKPIT_RIGHTMOST	29	cockpit far right
CAMERA_OVERLAY_FRONT	30	overlay front
CAMERA_OVERLAY_LEFT	31	overlay left
CAMERA_OVERLAY_RIGHT	32	overlay right
CAMERA_OVERLAY_LEFTMOST	33	overlay far left
CAMERA_OVERLAY_RIGHTMOST	34	overlay far right
CAMERA_TOP_VIEW	35	top view without perspective
CAMERA_DIAGRAM	36	full screen diagram on black background
CAMERA_MIRROR_LEFT2	37	2nd left mirror
CAMERA_MIRROR_RIGHT2	38	2nd right mirror
CAMERA_MIRROR_RIGHT3	39	3rd right mirror (downwards)

Name	Value	Comments
CAMERA_BLACK_LEFT	40	black screen with left overlay
CAMERA_BLACK_RIGHT	41	black screen with right overlay
CAMERA_USER	42	user-defined camera that can be controlled by **ENRE_USER_CAM**
CAMERA_BACK_LEFT	43	rear view side
CAMERA_BACK_RIGHT	44	rear view side
CAMERA_BACK_LEFT2	45	rear view side
CAMERA_BACK_RIGHT2	46	rear view side
CAMERA_LIVE_VIDEO	47	full-screen live video
CAMERA_TOP_VIEW_NOOVL	48	same as CAMERA_TOP_VIEW, but not showing the front screen overlay
CAMERA_REAR_VIEW	49	video camera for backwards view, in particular for euro trains
CAMERA_MIRROR_LEFT_AMPLIFICATION	50	
CAMERA_MIRROR_RIGHT_AMPLIFICATION	51	
CAMERA_USER_NOOVL	52	user-defined camera without the front overlay
CAMERA_HMD	53	camera for head mounted displays
CAMERA_BLACK_FRONT	54	black screen with front overlay (in particular: menu without 3d graphics)
CAMERA_VISUAL_LEFT	128	maps to **CAMERA_LEFT** or **CAMERA_RIGHT**
CAMERA_VISUAL_RIGHT	129	maps to **CAMERA_RIGHT** or **CAMERA_LEFT**
CAMERA_VISUAL_LEFT2	130	maps to **CAMERA_LEFT2** or **CAMERA_RIGHT2**
CAMERA_VISUAL_RIGHT2	131	maps to **CAMERA_RIGHT2** or **CAMERA_LEFT2**
CAMERA_VISUAL_BACK_LEFT	132	maps to **CAMERA_BACK_LEFT** or **CAMERA_BACK_RIGHT**
CAMERA_VISUAL_BACK_RIGHT	133	maps to **CAMERA_BACK_RIGHT** or **CAMERA_BACK_LEFT**
CAMERA_VISUAL_BACK_LEFT2	134	maps to **CAMERA_BACK_LEFT2** or **CAMERA_BACK_RIGHT2**
CAMERA_VISUAL_BACK_RIGHT2	135	maps to **CAMERA_BACK_RIGHT2** or **CAMERA_BACK_LEFT2**

Functions: System_Perspective

A.36.1 Top View Modes

Name	Value	Comments
TOPVIEW_FOLLOW	1	the camera is centered over the driver's vehicle. x and y are ignored.
TOPVIEW_ABSOLUTE	2	the camera is fixed at the position given by the x and y coordinates.

Functions: System_SetTopViewMode

A.37 Physical Vehicle Parameters

Name	Value	Comments
PHYS_MASS	1	Total vehicle mass in kg (]0 . . . 50000]) excluding tank contents
PHYS_SPRING_CONSTANT	2	factor for spring stiffness, > 1 means harder (]0 . . . 10])
PHYS_SPRING_DAMPING	3	factor for spring damping, 1 means critical, < 1 means oscillating ([0.05 . . . 5])
PHYS_TANK_COMPARTMENTS	4	number of transport tank compartments ([1 . . . 100])
PHYS_TANK_FILLLEVEL	5	fill level of a transport tank ([0 . . . 1])
PHYS_ROLL_RESISTANCE	6	factor for roll resistance, set to > 1 for partially flat tires (]0 . . . 10])
PHYS_WHEELBASE	7	wheelbase of vehicle (cannot be set)
PHYS_CENTEROFMASS_HEIGHT	8	height of center of mass im m (]0 . . . 4])
PHYS_CENTEROFMASS_LONGITUDINAL	9	offset of center of mass from default in m (front is positive direction), com has to stay between axles and has to be at least 1m away from them
PHYS_BRAKESTRENGTH_FRONT	10	factor: 1 means full braking on dry road will block exactly at maximum braking (theoretic value; dynamic load shifting slightly alters this) ([0 . . . 10])
PHYS_BRAKESTRENGTH_REAR	11	factor: 1 means full braking on dry road will block exactly at maximum braking (theoretic value; dynamic load shifting slightly alters this) ([0 . . . 10])
PHYS_TRAILBAR_LENGTH	12	trailbar length in m
PHYS_RETRACTABLE_AXLE_LIFT	13	flag: is the rectractable axle lifted? (0 or 1)
PHYS_MOTORPOWER	14	factor for motor power (]0 . . . 4])
PHYS_GEARFACTOR	15	factor for gear ratios (]0 . . . 10]), larger values make the gear ratio shorter
PHYS_FRONT_DRIVEN	16	flag: are the front wheels driven? (0 or 1)
PHYS_DIFFERENTIAL_LOCK	17	strength in [0 . . . 1], 0 means differential, 1 means no differential
PHYS_WHEELWIDTH_FRONT	18	wheel width in m
PHYS_WHEELWIDTH_REAR	19	wheel width in m
PHYS_TANK_LOADTYPE	20	type of loaded liquid, c.f. **TANKLOAD_**
PHYS_LENGTH	24	total length of the object. note, that objects' origins are centered therein.
PHYS_WIDTH	25	total width of the object. note, that objects' origins are centered therein.
PHYS_FRONTWHEELDIST	26	distance of front wheels to vehicle center, positive means to the front
PHYS_REARWHEELDIST	27	distance of rear wheels to vehicle center, positive means to the rear
PHYS_TRAINFRONTDIST	28	distance to train front, for simple vehicles half **PHYS_LENGTH**
PHYS_TRAINTAILDIST	29	distance to train's tail, for simple vehicles half **PHYS_LENGTH**
PHYS_AIRPRESSURE	31	brake system air pressure in bar (1000hPa)
PHYS_MOTORBRAKE	32	exhaust brake while dragging, c.f. **MOTORBRAKE_**
PHYS_REVERSE_BEEPER	34	1 or 0 for a beeper when ignition is on and reverse gear is engaged

Functions: Traffic_SetPhysicalParameter Traffic_GetPhysicalParameter

A.37.1 Motor Brake Modes

Name	Value	Comments
MOTORBRAKE_NEVER	0	
MOTORBRAKE_ALWAYS	1	
MOTORBRAKE_REPLACES_RETARDER	2	retarder not available, returder + lever activates motor brake
MOTORBRAKE_HIGH_RETARDER	3	motor brake active when highes retarder level chosen
MOTORBRAKE_INVALID	4	last valid + 1

A.38 Image Positioning Constants

Name	Value	Comments
SETIMG_ANCHOR_CENTER	0	use image center as anchor
SETIMG_ANCHOR_LEFT	1	use left image border as anchor
SETIMG_ANCHOR_RIGHT	2	use right image border as anchor
SETIMG_ANCHOR_TOP	4	use top image border as anchor
SETIMG_ANCHOR_BOTTOM	8	use bottom image border as anchor
SETIMG_ANCHOR_TL	5	combinations of above
SETIMG_ANCHOR_TR	6	
SETIMG_ANCHOR_BL	9	
SETIMG_ANCHOR_BR	10	
SETIMG_SCREENANCHOR_CENTER	0	settings for the screen reference point
SETIMG_SCREENANCHOR_LEFT	16	
SETIMG_SCREENANCHOR_RIGHT	32	
SETIMG_SCREENANCHOR_TOP	64	
SETIMG_SCREENANCHOR_BOTTOM	128	
SETIMG_SCREENANCHOR_TL	80	
SETIMG_SCREENANCHOR_TR	96	
SETIMG_SCREENANCHOR_BL	144	
SETIMG_SCREENANCHOR_BR	160	
SETIMG_TEXT_CENTERED	256	
SETIMG_TEXT_RIGHTALIGN	512	
SETIMG_TEXT_RTLREADING	1024	
SETIMG_TEXT_ALPHABLUR	2048	smooth black border around text
SETIMG_TEXT_BOLD	4096	
SETIMG_TEXT_ITALIC	8192	
SETIMG_NOBORDER	16384	don't display a border around the image
SETIMG_NEEDS_ANAMORPHY_CORRECTION	32768	

Functions: System_SetImageText

A.39 Text and Image Slot Constants

Name	Value	Comments
BSLOT_0	0	default slot, used for old scenario implementations
BSLOT_AAC	10	internally used for assistance systems overlay icons
BSLOT_NO_PER_LAYER	16	number of slots per layer
BSLOT_EVENT	176	==BSLOT_NO_PER_LAYER*11, // internally used slots for events are mapped to LAYER_MENU
BSLOT_LAST	192	total number of available slots (maximum index+1)

Note that slots address different layers which are used for sorting and optimising the output of overlay elements (2D-Bitmaps and Text). Each layer is handled as one group and updated as soon as one slot is changed. For optimal performance, each layer should cover small areas and be updated as rarely as possible.
Functions: System_SetImage, System_SetImageFile, System_SetImageText

A.40 System Information

Name	Value	Comments
SYSINFO_FRAMERATE	1	current framerate
SYSINFO_FROZENBYSCENARIO	2	is simulation frozen by scenario?
SYSINFO_SPOKENMESSAGESTATE	3	state of spoken messages. In: textnr or intl. string, Out: state (2: running, 1: queued, 0: neither)
SYSINFO_FRAMENUMBER	4	current frame number
SYSINFO_REQUIRE_CHIPCARD	6	is a chip card currently required?
SYSINFO_ADMINCARD	7	is an admin card currently inserted?
SYSINFO_HAS_STARTER_BUTTON	8	system has a starter button? (in contrast to ignition key).
SYSINFO_HAS_CLUTCH	9	system has hardware clutch pedal?
SYSINFO_MIRRORSWITCH_LEVEL	10	level of hardware mirror switch support: 0=none, 1=up/down only, 2=four-way, 3=four-way with hardware mirror selection (center may only support up/down), 4=full dedicated mirror support (either switched or seperate controls)
SYSINFO_HAS_ABBUTTONS	11	simulator has extra buttons 2 and 3 marked as A+B, sim. press acts as cancel.
SYSINFO_HAS_BELT	12	simulator has a safety belt? 0 = no, 2 = yes, 1 = yes, but not detectable.
SYSINFO_HAS_BLINKER_BUTTONS	13	the blinker (lever) is not turned off by restoring the central position, but by shortly activating the opposite direction.
SYSINFO_NAMEDDRIVER	14	either in chipcard mode or a driver id has been entered
SYSINFO_HAS_FOGLIGHT_SWITCH	15	type of hardware switch for the rear fog light? 0 = none, 1 = pull light switch, 2 = labelled button
SYSINFO_HAS_HMD	16	sight system is a virtual reality helmet/glasses. 2nd return value is 1 if a vr helmet is present, but not currently used.

Functions: System_GetInfo

A.41 Mathematical Functions

Name	Value	Comments
MATHFUNC_NUMBER	0	numbers are not modified, strings are transformed to their value or 0.
MATHFUNC_SQRT	1	square root sign(x1) * sqrt(x1*sign(x1)), operates element-wise on vectors
MATHFUNC_EXP	2	exponential function exp(x1), single argument only
MATHFUNC_POW	3	power sign(x1) * pow(x1*sign(x1), x2)
MATHFUNC_LN	4	natural logarithm sign(x1) * ln(x1*sign(x1)), single argument only
MATHFUNC_SIN_DEG	5	sine of x1 (in degree), operates element-wise on vectors
MATHFUNC_SIN_RAD	6	sine of x1 (in radians), operates element-wise on vectors
MATHFUNC_COS_DEG	7	cosine of x1 (in degree), single argument only
MATHFUNC_COS_RAD	8	cosine of x1 (in radians), single argument only
MATHFUNC_ABS	9	absolute value of x1, operates element-wise on vectors
MATHFUNC_FLOOR	10	integer value of x1 (round towards -infinity), operates element-wise on vectors
MATHFUNC_ARCSIN_DEG	11	arcsine of x1 (in degree), operates element-wise on vectors
MATHFUNC_ARCSIN_RAD	12	arcsine of x1 (in radians), operates element-wise on vectors
MATHFUNC_ARCCOS_DEG	13	arccosine of x1 (in degree), single argument only
MATHFUNC_ARCCOS_RAD	14	arccosine of x1 (in radians), single argument only
MATHFUNC_TAN_DEG	15	tangens of x1 (in degree), operates element-wise on vectors
MATHFUNC_TAN_RAD	16	tangens of x1 (in radians), operates element-wise on vectors
MATHFUNC_ARCTAN_DEG	17	arctan of x1 (in degree), operates element-wise on vectors
MATHFUNC_ARCTAN_RAD	18	arctan of x1 (in radians), operates element-wise on vectors
MATHFUNC_ARCTAN2_DEG	19	arctan2 of x1,x2 (opposite leg, adjacent leg) (in degree)
MATHFUNC_ARCTAN2_RAD	20	arctan2 of x1,x2 (opposite leg, adjacent leg) (in radians)
MATHFUNC_CEIL	21	integer value of x1 (round towards +infinity), operates element-wise on vectors
MATHFUNC_ROUND_TO_ZERO	22	integer value of x1 (round towards zero), operates element-wise on vectors
MATHFUNC_ROUND	23	integer value of x1 (round to nearest integer), operates element-wise on vectors
MATHFUNC_SUM	24	sum of all vector elements
MATHFUNC_L2NORM	25	sum of squares of all vector elements
MATHFUNC_L1NORM	26	sum of absolute values of all vector elements
MATHFUNC_RECTDIST	27	distance of two rectangles, arguments are w1 l1 x1 z1 a1 w2 l2 [x2 z2 a2], with center positions and angles in degrees
MATHFUNC_SORT	28	vector only: returns x1 s1 ... sn where n=x1 and s1 ... sn is x2 ... x(n+1) sorted.
MATHFUNC_UNIQ	29	vector only: returns m s1 ... sm where s1 ... sm is x2 ... x(x1+1) with subsequent equal elements removed.
MATHFUNC_SORT_UNIQ	30	vector only: same as **MATHFUNC_UNIQ** after **MATHFUNC_SORT**
MATHFUNC_FIND	31	vector only: arguments are y n x1 ... xn, returns lowest i such that x(i+1) = y or -1 if y is not found among the xi
MATHFUNC_FIND_SORTED	32	same as **MATHFUNC_FIND**, but the xi are assumed to be sorted. O(log n) instead of O(n).
MATHFUNC_LOWER_BOUND	33	vector only: arguments are y n x1 ... xn with x1 ... xn sorted. returns the lowest i such that inserting y at i would leave the list sorted.
MATHFUNC_UPPER_BOUND	34	vector only: arguments are y n x1 ... xn with x1 ... xn sorted. returns the lowest i such that inserting y behind i would not leave the list sorted, n if no such i exists.
MATHFUNC_EQUAL_RANGE	35	vector only: returns MATHFUNC_LOWER_BOUND and MATHFUNC_UPPER_BOUND
MATHFUNC_INSERT_SORTED	36	vector only: arguments are y n x1 ... xn with x1 ... xn sorted. returns n x1 ... xn if y is contained in the xi, otherwise n+1 y1 ... y(n+1) where the yi are the sorted union of y and the xi.
MATHFUNC_ERASE_SORTED	37	vector only: arguments are y n x1 ... xn with x1 ... xn sorted. returns n x1 ... xn if y is not contained in the xi, otherwise m y1 ... ym where the yi are the xi with y removed.

Functions: System_Math

A.42 String Transformations

Name	Value	Comments
STRINGTRAFO_IDENTITY	0	just return the input string
STRINGTRAFO_TRIMLEFT	1	remove leading spaces
STRINGTRAFO_TRIMRIGHT	2	remove trailing spaces
STRINGTRAFO_TOUPPER	3	transform to upper case
STRINGTRAFO_TOLOWER	4	transform to lower case
STRINGTRAFO_URLENCODE	5	URL-Encode, an additional second string may specify additional characters to encode
STRINGTRAFO_URLDECODE	6	URL-Decode, inverse of **STRINGTRAFO_URLENCODE**
STRINGTRAFO_ISSTRING	7	query whether the given argument is a string
STRINGTRAFO_TRIMBOTH	8	remove leading and trailing spaces
STRINGTRAFO_SPLITURI	9	returns the decoded URL part followed by decoded arguments
STRINGTRAFO_REGEX_REPLACE	10	arguments are s, ex, fmt, glob. replace matches of ex in s with fmt, globally if $glob = 1$. ECMAScript regular expression rules are used.

Functions: String_Transform

A.43 File Operations

Name	Value	Comments
FILEOP_READ	0	read the given file
FILEOP_WRITE	1	write the given vector into the specified file
FILEOP_FIND	2	search files matching the given mask (use paths without file suffix)
FILEOP_DELETE	3	delete all files matching the given mask (use paths without file suffix)

Functions: System_DataFile

A.44 Lane Types

Name	Value
LANETYPE_NONE	0
LANETYPE_OUTERCRASH	0
LANETYPE_INNERCRASH	1
LANETYPE_GREEN	2
LANETYPE_SIDEWALK	3
LANETYPE_PARKING	4
LANETYPE_DRIVING_FIRST	5
LANETYPE_DRIVING_NORMAL	5
LANETYPE_DRIVING_TURNONLY	6
LANETYPE_DRIVING_LAST	6

Functions: World_GetLanes

A.45 Extra Systems

Name	Value	Comments
EXTRASYSTEM_ABS	1	antilock braking system
EXTRASYSTEM_CRUISECONTROL	2	distance regulation control (ACC)
EXTRASYSTEM_ESP	3	electronic stability program
EXTRASYSTEM_BUSDOORS	4	not implemented yet
EXTRASYSTEM_ASR	5	traction control (part of ESC)
EXTRASYSTEM_AEBS	6	active emergency braking system

Functions: Driver_SetExtraSystem

Appendix B

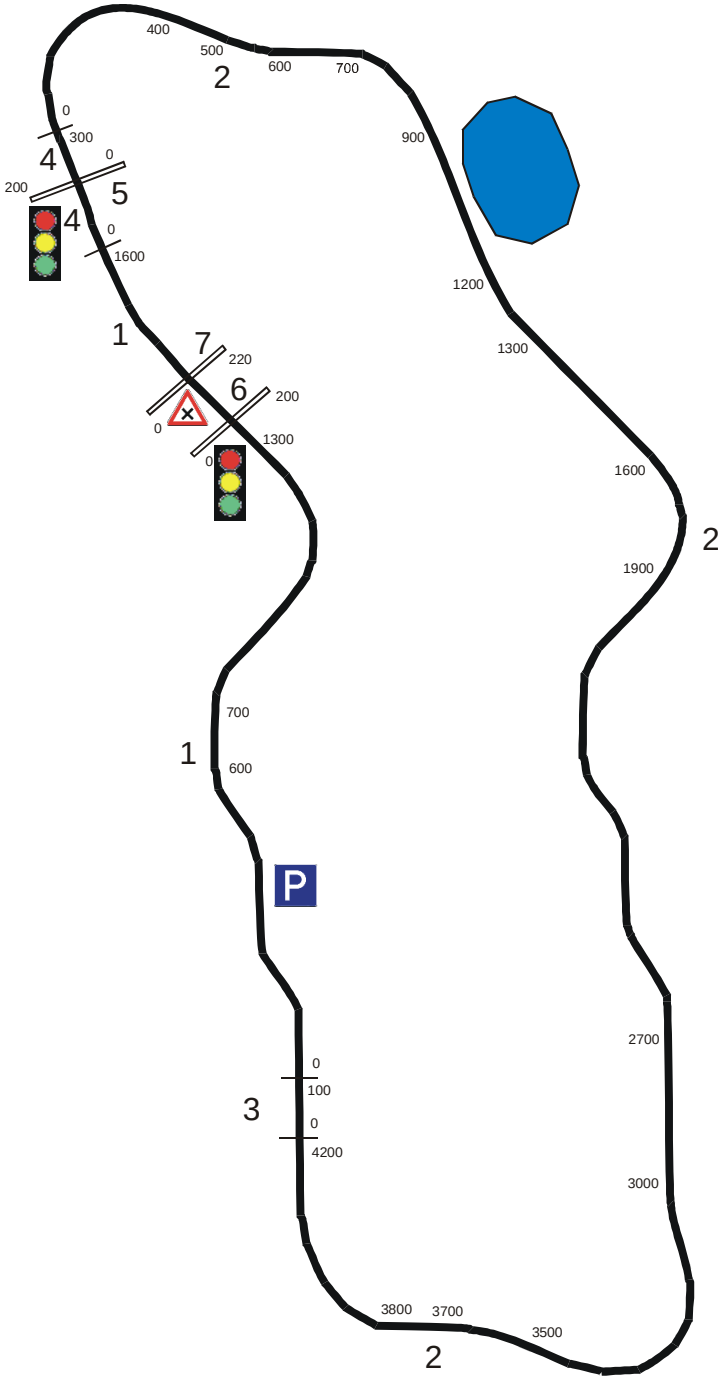
World Maps

The following sections give an overview about the available maps. In the figures showing each map, the road numbers are written as large numbers, while distance values are written as smaller numbers besides the ends of each road segment. The direction from 0 to maximal distance also defines the direction corresponding to angle value 0.
Please note that most maps are not suitable for rides at night.

B.1 Circuit

Constant: **WORLD_ROUNDCOURSE**

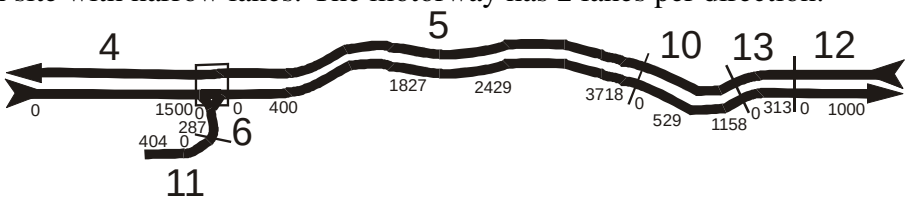
This map represents a circuit. There are some streets crossing the circuit, but the own vehicle must keep the route on the circuit road. Do not insert vehicles at road no 1 and distance 0, because the circuit begins here. The width of a lane is 3 meters. This map is suitable for night rides.

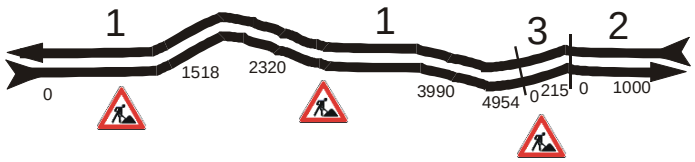


B.2 Motorway

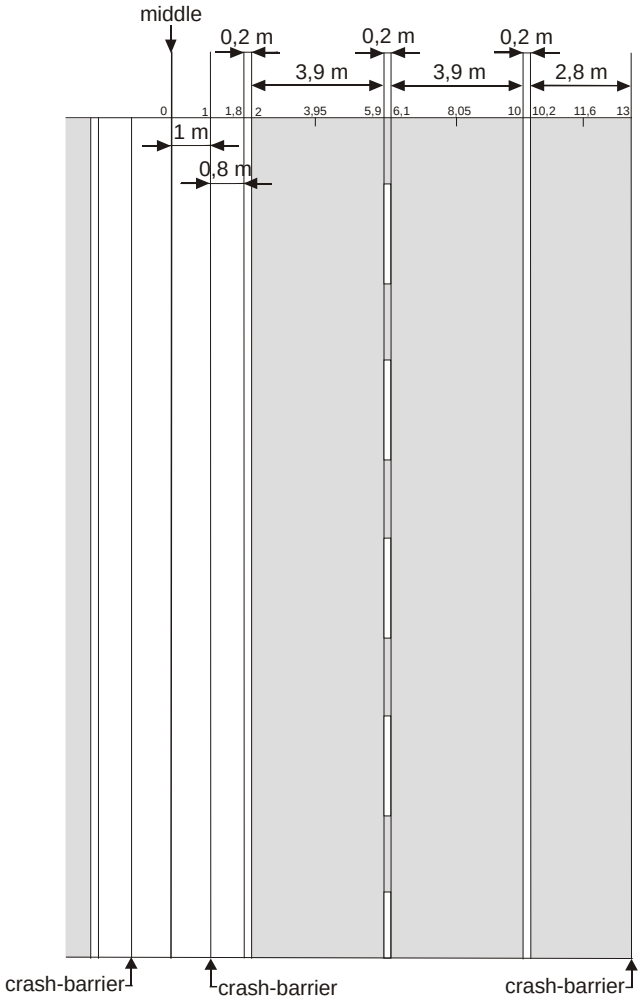
Constant: **WORLD_MOTORWAY**

The map includes two courses. The direction from the left to the right on the map below is endless. That means if a vehicle drives to the end of a course it will be set to the start. One course includes an entry and a exit. The other course includes a construction site with narrow lanes. The motorway has 2 lanes per direction.



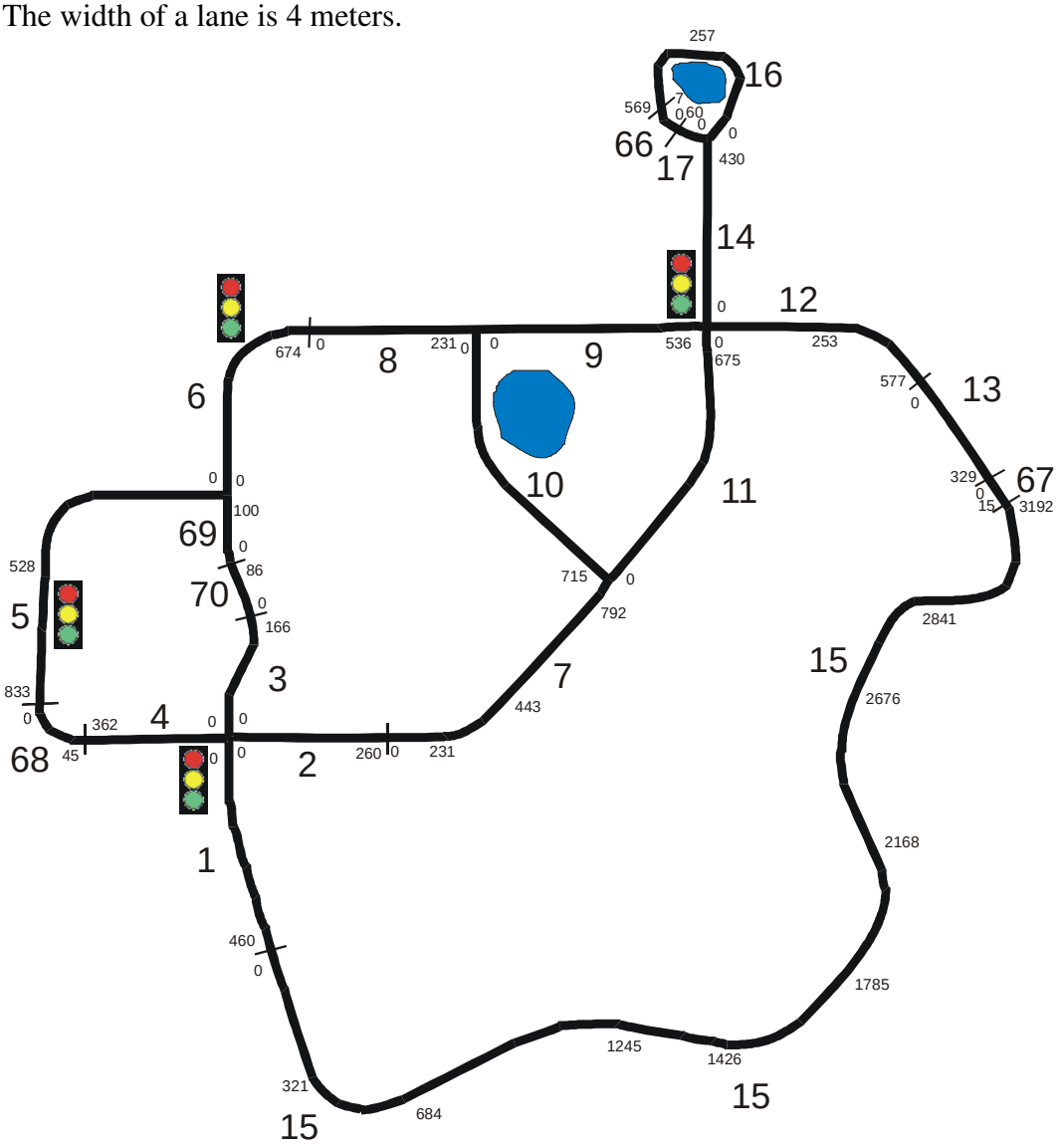


The figure to the right shows the widths of the lanes in the first segment. The small numbers show the distance from the middle of the motorway, e.g. for **Traffic_SetTrackDeviation**.



B.3 Trainer Map

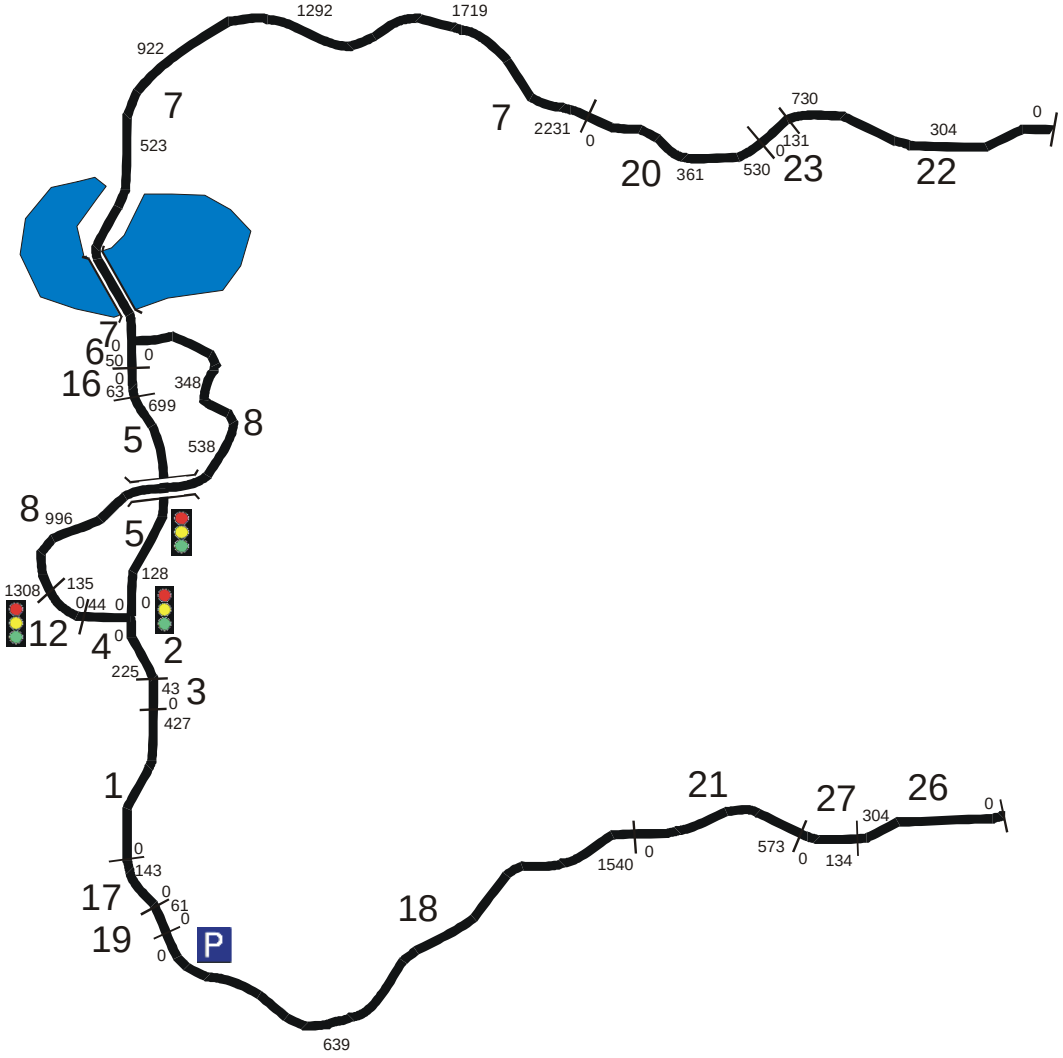
Constant: **WORLD_TRAINER**
The width of a lane is 4 meters.



B.4 Mountain Course

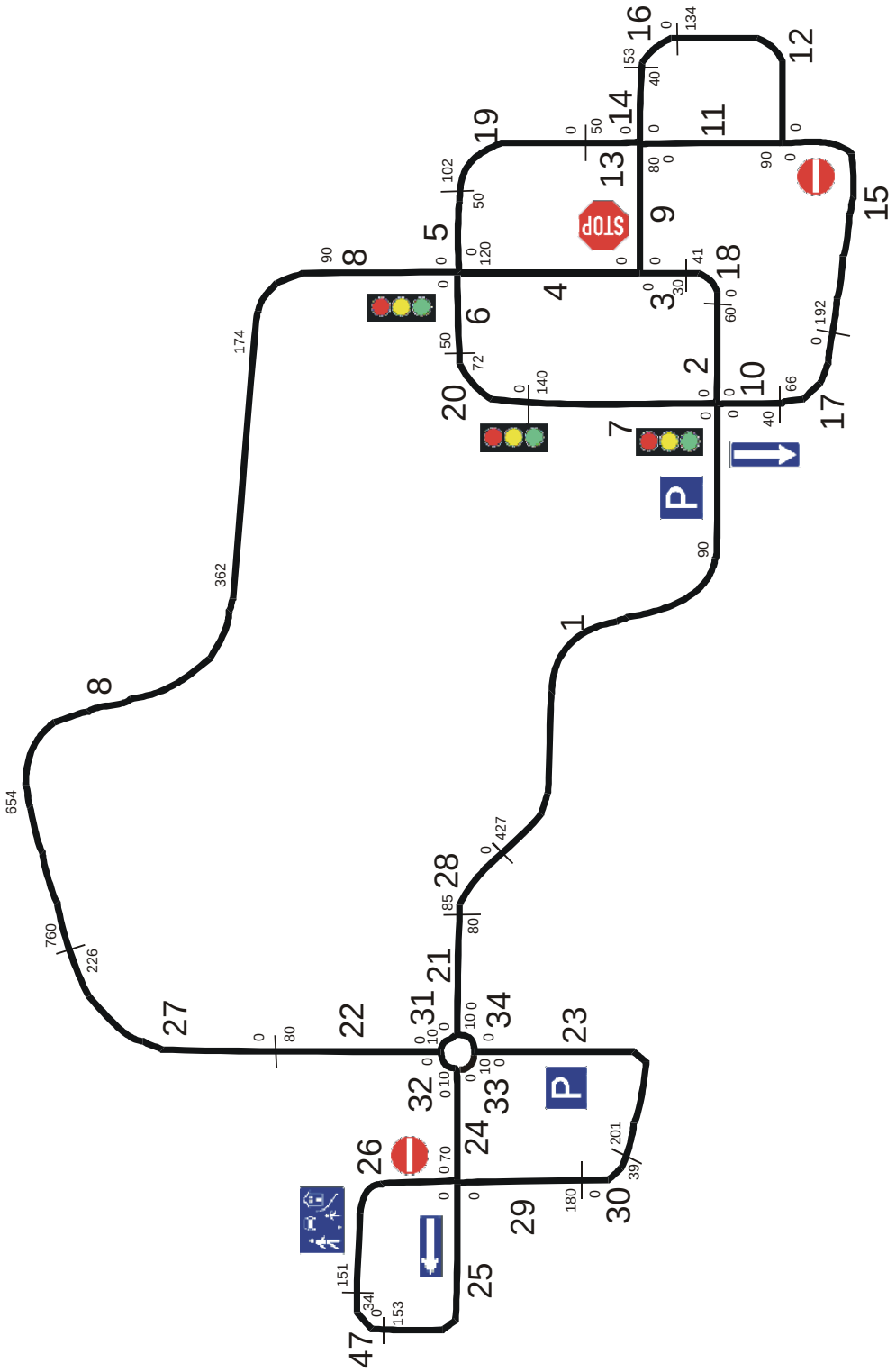
Constant: **WORLD_MOUNTAIN**

Course in mountains with 3D elements. Do not drive near the ends of the course (road no 22 and 26). The width of a lane is 4 meters.



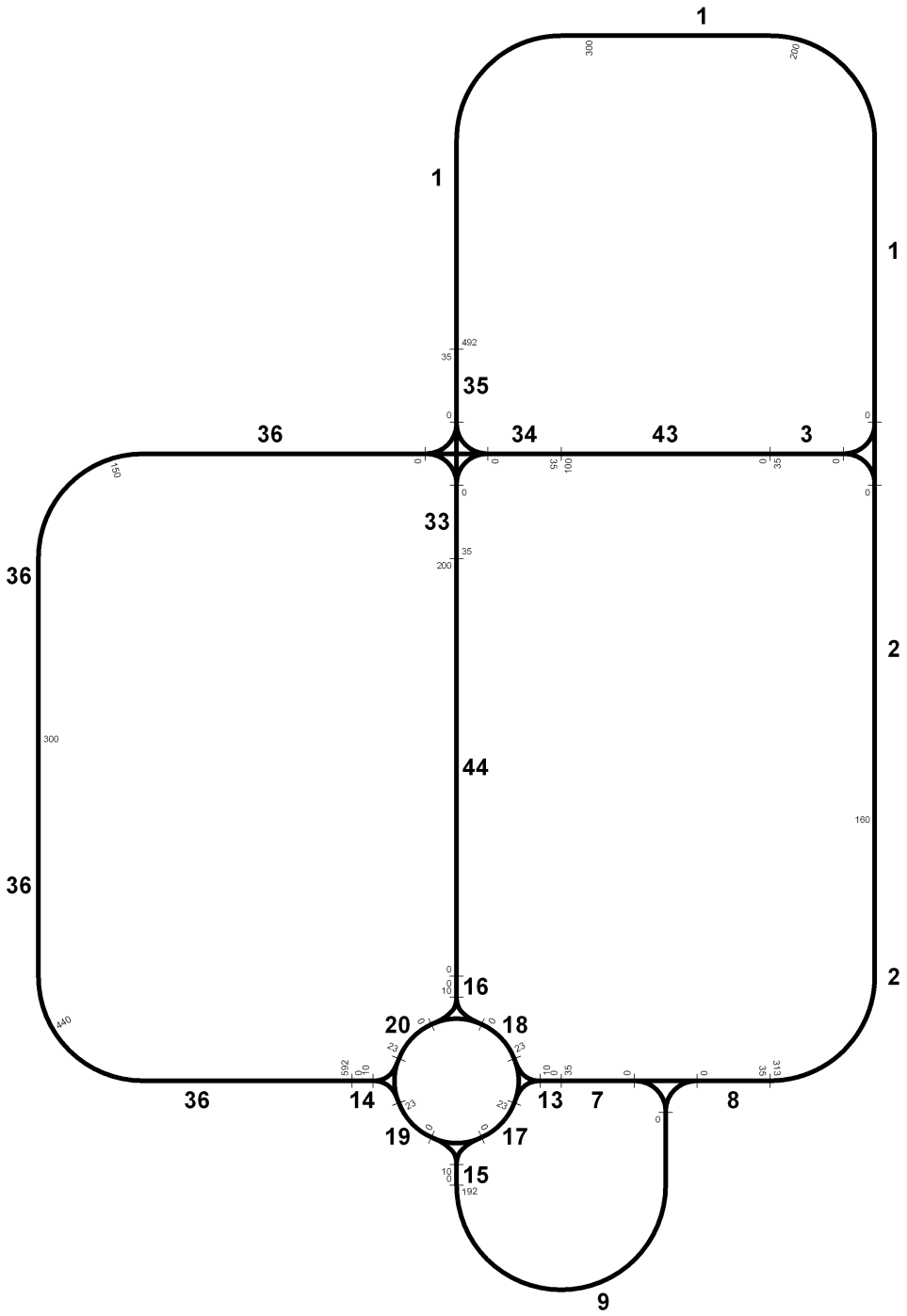
B.5 Small Town

Constant: **WORLD_SMALLTOWN**
The width of a lane is 3 meters.



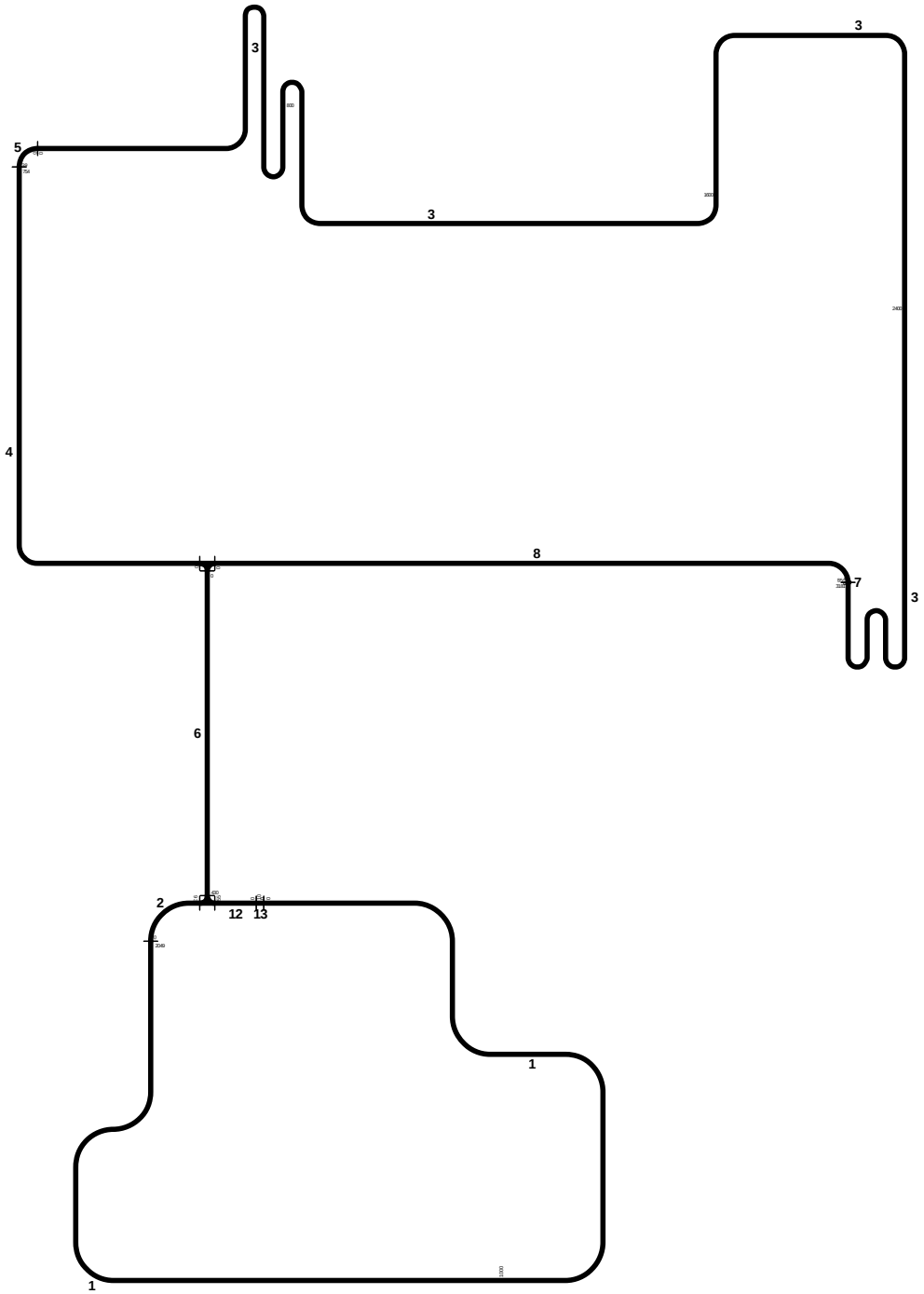
B.6 City

Constant: **WORLD_TOWN**
City map with two lanes per direction. The width of a single lane is 4 meters.



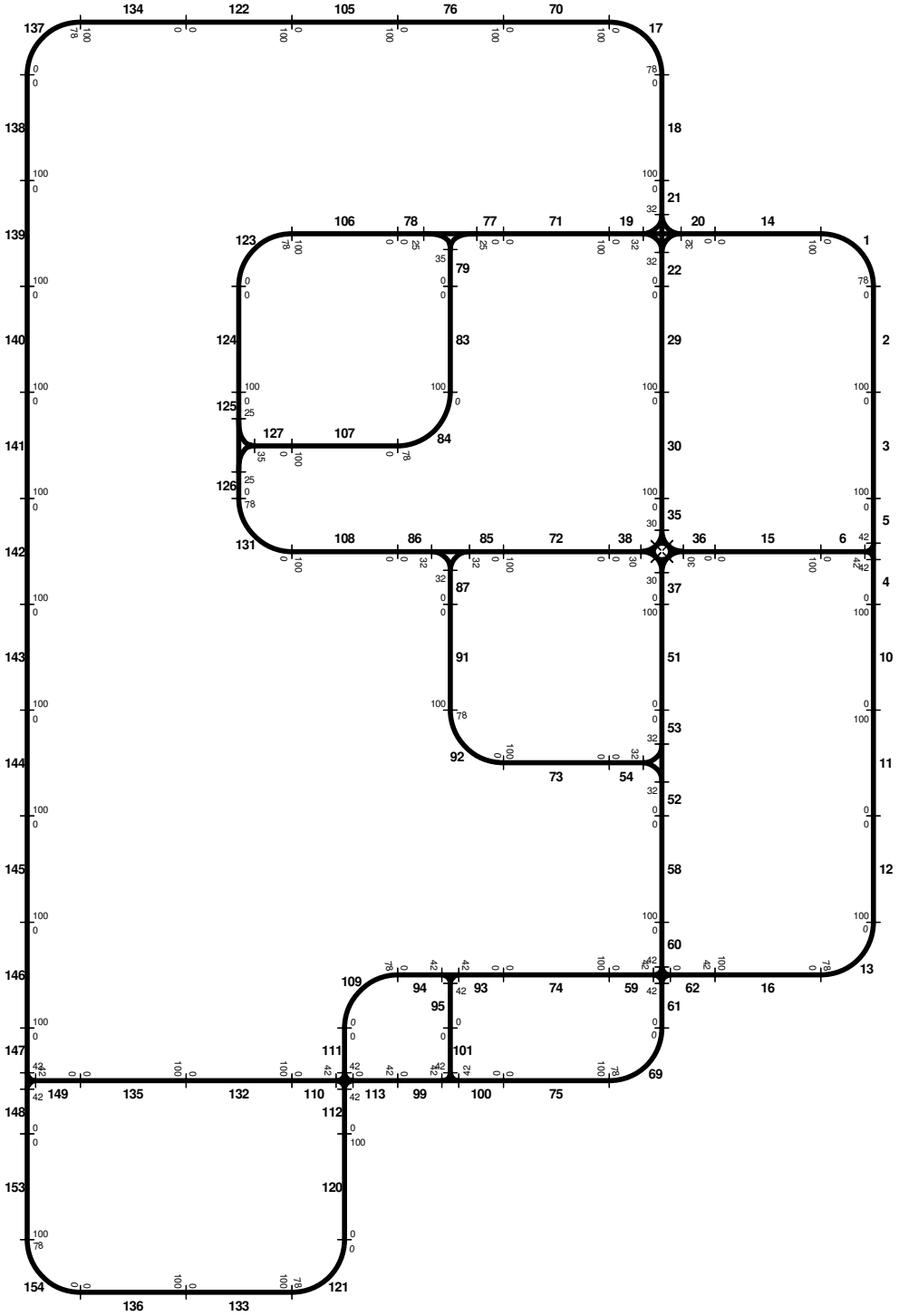
B.7 Desert

Constant: **WORLD_WASTELAND**
Simple round-course through desert area. Sight is restricted by hills to both sides of the road. The width of a single lane is 5 meters.



B.8 City 2

Constant: **WORLD_CITY2**
City map with both two lanes and one lane per direction, many junctions in the inner city area. Also has some sections of rural road outside the city limits.



As the road numbers in the above maps may change in the future (e.g. map extensions), we recommend not to use them directly in scenario scripts. Instead, you should look up the road numbers using **World_LookupRoadNumber** using the data in the table below.

Road	tilename	tilerdno
1	urbancurvesl1	1
2	urbanstraightsl1	1
3	urbanstraightsl2	1
4	urbantcrosssl	1
5	urbantcrosssl	2
6	urbantcrosssl	3
7	urbantcrosssl	4
8	urbantcrosssl	5
9	urbantcrosssl	6
10	urbanstraightsl3	1
11	urbanstraightsl4	1
12	urbanstraightsl5	1
13	urbancurvesl2	1
14	urbantransition1	1
15	urbantransition2	1
16	urbanstraightsl6	1
17	urbancurvedl1	1
18	urbanstraightdl1	1
19	urbancrossdl	1
20	urbancrossdl	2
21	urbancrossdl	3
22	urbancrossdl	4
23	urbancrossdl	5
24	urbancrossdl	6
25	urbancrossdl	7
26	urbancrossdl	8
27	urbancrossdl	9
28	urbancrossdl	10
29	urbanstraightdl2	1
30	urbanparkingslot1	1
31	roundabout	1
32	roundabout	2
33	roundabout	3
34	roundabout	4
35	roundabout	5
36	roundabout	6

Road	tilename	tilerdno
37	roundabout	7
38	roundabout	8
39	roundabout	9
40	roundabout	10
41	roundabout	11
42	roundabout	12
43	roundabout	13
44	roundabout	14
45	roundabout	15
46	roundabout	16
47	roundabout	17
48	roundabout	18
49	roundabout	19
50	roundabout	20
51	urbanstraightdl3	1
52	urbantcrossdl1	1
53	urbantcrossdl1	2
54	urbantcrossdl1	3
55	urbantcrossdl1	4
56	urbantcrossdl1	5
57	urbantcrossdl1	6
58	urbantransition3	1
59	urbancrosssl	1
60	urbancrosssl	2
61	urbancrosssl	3
62	urbancrosssl	4
63	urbancrosssl	5
64	urbancrosssl	6
65	urbancrosssl	7
66	urbancrosssl	8
67	urbancrosssl	9
68	urbancrosssl	10
69	countrycurve1	1
70	industrystraightdl1	1
71	urbanstraightdl4	1
72	urbanstraightdl5	1

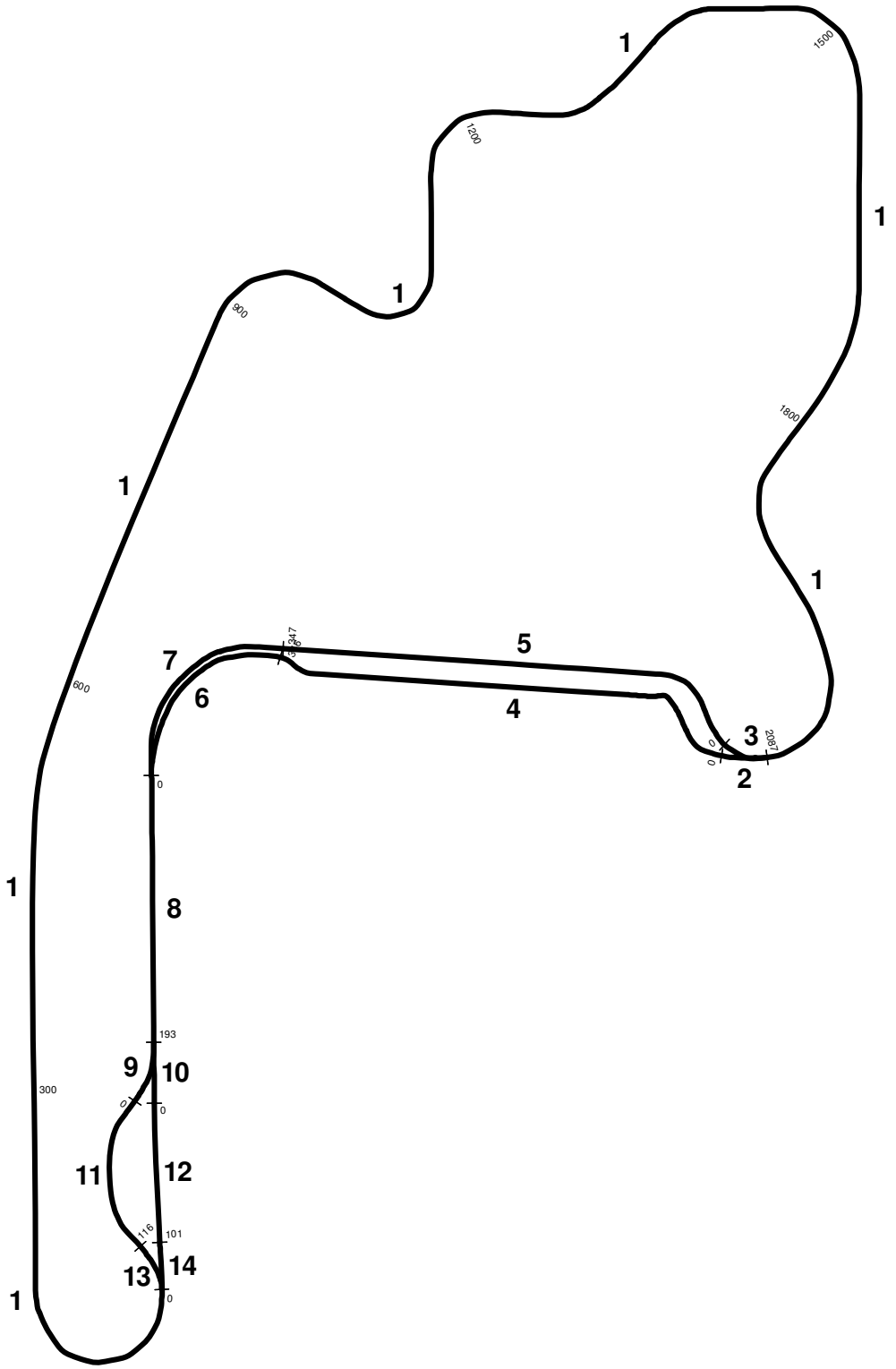
Road	tilename	tilerdno
73	urbanparkingslot2	1
74	countrystraight1	1
75	countrystraight2	1
76	industrystraightdl2	1
77	urbantcrosstrans1	1
78	urbantcrosstrans1	2
79	urbantcrosstrans1	3
80	urbantcrosstrans1	4
81	urbantcrosstrans1	5
82	urbantcrosstrans1	6
83	urbanstraightdl6	1
84	urbancurves13	1
85	urbantcrossdl2	1
86	urbantcrossdl2	2
87	urbantcrossdl2	3
88	urbantcrossdl2	4
89	urbantcrossdl2	5
90	urbantcrossdl2	6
91	urbanstraightdl7	1
92	urbancurvedl2	1
93	countrytcross1	1
94	countrytcross1	2
95	countrytcross1	3
96	countrytcross1	4
97	countrytcross1	5
98	countrytcross1	6
99	countrytcross2	1
100	countrytcross2	2
101	countrytcross2	3
102	countrytcross2	4
103	countrytcross2	5
104	countrytcross2	6
105	industrystraightdl3	1
106	urbanparkingslot3	1
107	urbanparkingplace1	1
108	urbanstraightdl8	1

Road	tilename	tilerdno
109	countrycurve2	1
110	countrycross	1
111	countrycross	2
112	countrycross	3
113	countrycross	4
114	countrycross	5
115	countrycross	6
116	countrycross	7
117	countrycross	8
118	countrycross	9
119	countrycross	10
120	countrystraight3	1
121	countrycurve3	1
122	industrystraightdl4	1
123	urbancurvedl3	1
124	urbanstraightdl9	1
125	urbantcrosstrans2	1
126	urbantcrosstrans2	2
127	urbantcrosstrans2	3
128	urbantcrosstrans2	4
129	urbantcrosstrans2	5
130	urbantcrosstrans2	6
131	urbancurvedl4	1
132	countrystraight4	1
133	bridge	1
134	urbantransition4	1
135	countrystraight5	1
136	countrystraight6	1
137	urbancurvesl4	1
138	longcountrystraight1	1
139	longcountrystraight2	1
140	longcountrystraight3	1
141	longcountrystraight4	1
142	longcountrystraight5	1
143	longcountrystraight6	1

Road	tilename	tilerdno
144	longcountrystraight7	1
145	longcountrystraight8	1
146	longcountrystraight9	1
147	longcountrytcross	1
148	longcountrytcross	2
149	longcountrytcross	3
150	longcountrytcross	4
151	longcountrytcross	5
152	longcountrytcross	6
153	longcountrystraight10	1
154	countrycurve4	1

B.9 Racing Course

Constant: **WORLD_RACING**
A racing course, one direction only. Features narrow curves and rather steep bevels.



Appendix C

Keyboard Shortcuts for F12 Driving Simulators

The simulator program offers a lot of functions using the keyboard. These functions are normally disabled to avoid malfunctioning of the program by accidently pressing keys. They can be useful to test certain simulator functions. Basically the simulator can be operated without external interface using the keyboard only. This list defines all currently available keyboard functions in the driving simulator software.

C.1 Fast Driver Identification Input

This function allows the user to enter the driver identification without choosing and activating the appropriate menu entry. Provided that a menu is shown containing the appropriate line and not another edit function is active, upon pressing a key the menu line is changed and activated and the key is inserted. This function has priority over the Global Name Input but is disabled by unlocking the keyboard (see below).

C.2 Global Name Input

This function allows the user to enter the driver's name during an ongoing ride with the simulator. The name will then be used in the (normally activated) printout after the ride. This entry mode has priority over all other input methods. It can be disabled by unlocking the keyboard (see below).

C.3 Keyboard Lock

Most keyboard functions are locked and must be unlocked before use. To unlock and relock these codes, press ⟨SHIFT+F12⟩. A message on the simulator screen appears and tells if the keyboard is now locked or unlocked. Some functions are available all the time and are shown in boldface in the following table.

Unlocking of the keyboard temporarly disables the global name input and the automatic driver identification input.

C.4 Keyboard Codes

C.4.1 ⟨SHIFT+CTRL+ALT+...⟩

⟨**A**⟩	**Force calibration process.** (May be cancelled using Horn / ⟨F2⟩.)
⟨**F2**⟩	**Switch writing of debug logfile on/off.**
⟨**F4**⟩	**Force simulator computer shutdown including networked computers.**
⟨**F9**⟩	Save the current internal replay film as file **/Video/Replay-Film.rpf**.
⟨**F10**⟩	Reinitialise Screensaver and reload configuration.
⟨**F11**⟩	Force and lock the Demonstration-Screensaver (or unlock it again).

C.4.2 ⟨PLAIN KEYS⟩

⟨ARROW LEFT⟩	Steer left.
⟨ARROW RIGHT⟩	Steer right.
⟨ARROW UP⟩	Accelerate (60%, use ⟨SHIFT⟩ for 100%).
⟨ARROW DOWN⟩	Brake (100%).
⟨**F1**⟩	**Switch language (only between loaded languages).**
⟨**F2**⟩	**Horn/Enter.**
⟨**F3**⟩	**Mode.**
⟨**F4**⟩	**Line.**
⟨F5⟩	Starter (button function).
⟨F6⟩	Ignition ON.
⟨F7⟩	Ignition OFF.
⟨NUM-,⟩	Handbrake ON/OFF (has a timerdelay).
⟨NUM- -⟩	Gearshiftbox: gear down.
⟨NUM- +⟩	Gearshiftbox: gear up.
⟨F9⟩	Print list of best performers. This key needs to be pressed twice and the printer needs to be activated in the service menu. (This function only works on programs with economy-print-option.)
⟨F10⟩	Turn driver camera to the left. (This key also activates the window menu.)
⟨F11⟩	Turn driver camera to normal position.

⟨F12⟩	Turn driver camera to the right.
⟨**1**⟩...⟨**8**⟩	**Choose perspektive 1...8.** (Normal driver's view is ⟨1⟩.)
⟨**9**⟩	**Top view perspektive.** (Zoom using Slider 1 ⟨CTRL+D⟩)
⟨**0**⟩	**High view perspektive.**
⟨C⟩	Horn/Enter.
⟨C⟩	next camera.
⟨D⟩	Display Mode (Wireframe, Flat, Gouraud, Anti-Aliased)
⟨F⟩	Flight Mode of camera. (for free camera only)
⟨O⟩	Orbit Mode of camera. (for free camera only)
⟨H⟩	Throw in car behind us.
⟨L⟩	Throw in car before us.
⟨Y⟩	Wiper modes (TEST).
⟨T⟩	Economy: Part-Bestlist printout. Push twice to activate, Printer ha to be activated (in Service-Menue). (Only if Economy Prinout included)
⟨R⟩	**Scripting langugage:** Reload current Scenario (does not always work)

C.4.3 ⟨SHIFT+...⟩

⟨0⟩	Debug Mode off.
⟨1⟩	Debug Mode 1.0: Pitch angle
⟨2⟩	Debug Mode 2.0: Network data.
⟨3⟩	Debug Mode 3.0: -
⟨4⟩	Debug Mode 1.1: Timing.
⟨5⟩	Debug Mode 1.2: Driving data.
⟨6⟩	Debug Mode 1.3: Track data.
⟨7⟩	Debug Mode 1.4: Collision data.
⟨8⟩	Debug Mode 1.5: Screensaver.
⟨9⟩	Debug Mode 1.6: Framerate (lower right).
⟨D⟩	Onscreen Debug Texts on/off
⟨F1⟩	Steering factor choice and steering reset for ADIODA.
⟨F3⟩	Bonnet on/off.
⟨F4⟩	Title on/off.
⟨F5⟩	Light on/off.
⟨F6⟩	High-beam on/off.
⟨F7⟩	Set camera (Attention: cannot be unset)
⟨F8⟩	Cycle wiper functions
⟨F9⟩	Monitor1 on/off (Rendering)

⟨F10⟩	Monitor2 on/off (Rendering) , Activation/Deactivation of input
⟨F11⟩	Graphic system1 on/off (Mode/Screen.)
⟨**F12**⟩	**Keyboard Lock on/off.**
⟨NUM-0⟩	Gearshiftbox: neutral gear.
⟨NUM-1⟩	Gearshiftbox: 1. gear.
⟨NUM-2⟩	Gearshiftbox: 2. gear.
⟨NUM-3⟩	Gearshiftbox: 3. gear.
⟨NUM-4⟩	Gearshiftbox: 4. gear.
⟨NUM-5⟩	Gearshiftbox: 5. gear.
⟨NUM-6⟩	Gearshiftbox: reverse gear.
⟨X⟩	Gearshift choice (before start of ride)
⟨L⟩	Speed up to 200km/h (on/off).
⟨A⟩	Traffic light program: set all green (only some programs)
⟨R⟩	Reset of traffic objects: remove all, reinsert, and test placement.
⟨N⟩	Force new lap. (only Roundcourse and motorway.)
⟨S⟩	**Scripting langugage:** Start Debug Mode and Singlestep Mode
⟨V⟩	**Scripting langugage:** shows all Float Variables in Debug Mode
⟨I⟩	**Scripting langugage:** shows all Internal Variables in Debug Mode
⟨P⟩	**Scripting langugage:** shows all Position variables in Debug Mode
⟨B⟩	**Scripting langugage:** Set Breakpoint in Debug Mode
⟨G⟩	**Scripting langugage:** Stop Debug Mode (GO)

C.4.4 ⟨CTRL+...⟩

⟨0⟩	Debug Mode off.
⟨1⟩	Debug Mode 1.7: Oncoming Traffic.
⟨2⟩	Debug Mode 1.8: Comoving Traffic.
⟨3⟩	Debug Mode 1.9: Driver's Environment.
⟨4⟩	Debug Mode 3.4: ‚too slow' data
⟨5⟩	Debug Mode 3.5: ‚WayToCollision' data
⟨6⟩	Debug Mode 3.6: Scenario data
⟨7⟩	Debug Mode 3.7: TCP/IP data
⟨L⟩	Automatic Steering, no steering forces.
⟨**F1**⟩	**Diagnostic Display on/off.**
⟨**F2**⟩	**Debug dialog on/off.**
⟨F5⟩	Indicator left

⟨F6⟩	Indicator right
⟨F7⟩	Warning lights
⟨F9⟩	Economy: delete Highscore List. Press twice, 'Economy' has to be installed.
⟨F⟩	Change Front-Beamer type.
⟨D⟩	Change Slider Mode. Set Slider 1 for zoom of top view camera.
⟨P⟩	Make the car before us park.
⟨R⟩	Reverse direction (disabled in some circumstances).
⟨S⟩	Image Generator: Select source, 1 to 5.
⟨T⟩	Change Speed indicator test mode.
⟨V⟩	Change Driver-Vehicle : Mondeo, Lkw, Jeep, . . .
⟨W⟩	Generate screen shot
⟨NUM-7⟩	User defined button 1
⟨NUM-8⟩	User defined button 2
⟨NUM-9⟩	User defined button 3

C.4.5 ⟨ALT+. . . ⟩

⟨**TAB**⟩	**Stop fullscreen mode (reenable with start button).**
⟨**F4**⟩	**Quit program.**

Appendix D

Index

BEGIN, 24
BREAKPOINT, 37

CALC, 30, 54
CALL, 36
CASE, 34
CHECKSUM, 24

DEBUGNEWLINE, 37
DEBUGTEXT, 37
DEBUGVAR, 37
DEFAULT, 34

ELSE, 32
ELSIF, 32
ELSIFONCE, 33
END, 24
ENDIF, 32
ENDNAMESPACE, 127
ENDSWITCH, 34
ENDWHILE, 34

FOR, 35
FUNC, 36
- Driver_GetAbsolutePosition, 58
- Driver_GetAcceleration, 58
- Driver_GetAcceleratorPedal, 58
- Driver_GetAngles, 59
- Driver_GetBlinker, 59
- Driver_GetBrakePedal, 59
- Driver_GetClutchPedal, 60
- Driver_GetDistance, 60
- Driver_GetGear, 60
- Driver_GetGearShift, 60
- Driver_GetHorn, 60
- Driver_GetID, 60
- Driver_GetIgnition, 61
- Driver_GetInstrument, 61
- Driver_GetLateralAcceleration, 61
- Driver_GetLight, 61
- Driver_GetRPM, 61
- Driver_GetSlideAngle, 62
- Driver_GetSpeed, 62
- Driver_GetStarter, 62
- Driver_GetSteeringWheel, 62
- Driver_GetWiper, 62
- Driver_SetAbs, 62
- Driver_SetAbsoluteOrientation, 63
- Driver_SetAbsolutePosition, 63
- Driver_SetAcceleratorPedal, 64
- Driver_SetBrake, 64
- Driver_SetCollision, 68
- Driver_SetDynamicLateral, 64
- Driver_SetDynamicLongitude, 65
- Driver_SetExtraSystem, 63
- Driver_SetGearShift, 65
- Driver_SetInstrument, 65
- Driver_SetMotorSound, 65
- Driver_SetPosition, 67
- Driver_SetPositionDirect, 67
- Driver_SetSpeed, 68
- Driver_SetSteeringWheel, 68
- Driver_SetWorld, 66
- Error_GetErrorCount, 69
- Error_GetErrorMask, 69
- Error_GetLastError, 70
- Error_ResetErrors, 70
- Error_SetError, 70
- Error_SetErrorMode, 70
- Error_SetTriggerValue, 71
- Logfile_AddScriptVariable, 72
- Logfile_Reset, 72
- Logfile_SetRate, 73
- String_GetLength, 74
- String_Part, 74
- String_Transform, 74
- Surround_GetDistanceToLineCrossing, 76
- Surround_GetHeadway, 75
- Surround_GetRoadAhead, 76
- Surround_GetRoadSpeed, 76
- Surround_GetRoadType, 76
- Surround_GetTimeToCollision, 75
- Surround_GetTimeToLineCrossing, 75
- Surround_SetEnvironmentSound, 75
- Surround_SetExternOrientation, 76
- Surround_SetLight, 77
- Surround_SetLightDurations, 77
- Symbol_Activation, 78
- Symbol_SetBrightness, 78
- Symbol_SetPosition, 78
- Symbol_SetSize, 79
- Symbol_SetTransparency, 79
- System_ClearText, 80
- System_ClearUserVariable, 93
- System_CutSubVector, 85
- System_DataFile, 92
- System_EnhancedReality, 80
- System_EnhancedRealityParameter, 80
- System_GetControl, 81
- System_GetCountry, 81
- System_GetInfo, 91
- System_GetLanguage, 81
- System_GetRandom, 81
- System_GetReport, 88
- System_GetSimulation, 82
- System_GetStatus, 82
- System_GetSubVector, 84
- System_GetUserData, 92
- System_GetUserVariable, 93
- System_GetViewSize, 82
- System_InitPhase, 94
- System_InitSampleFile, 82
- System_Math, 91
- System_Menu, 83
- System_Perspective, 83
- System_PlaySample, 83
- System_Reset, 84
- System_Scenario, 84
- System_Screenshot, 88
- System_SetAlcoholValue, 89
- System_SetBlurEffect, 89
- System_SetCurrentPos, 90
- System_SetDisplayOptions, 90
- System_SetImage, 86
- System_SetImageFile, 87
- System_SetImageText, 87
- System_SetReport, 88
- System_SetSubVector, 84
- System_SetText, 90
- System_SetTopViewArea, 91
- System_SetTopViewMode, 91
- System_SetUserData, 92
- System_SetUserVariable, 93
- System_Shutdown, 93
- System_Start, 94
- System_Stop, 94
- System_ToggleAlcoholValue, 89
- System_VectorMath, 92
- Traffic_GetAbsoluteDistance, 95
- Traffic_GetAbsolutePosition, 95
- Traffic_GetAngles, 96
- Traffic_GetDistAndVelocity, 96
- Traffic_GetEventName, 101
- Traffic_GetInfo, 96
- Traffic_GetPhysicalParameter, 108
- Traffic_GetPosition, 97
- Traffic_GetRouteStatus, 97
- Traffic_InsertCarAtNextJunction, 97
- Traffic_InsertVehicle, 98
- Traffic_IsOnInvisiblePlate, 110
- Traffic_ObjectExists, 99

Traffic_Remove, 99
Traffic_ResetSigns, 100
Traffic_SetAbsolutePosition, 103
Traffic_SetAcceleration, 108
Traffic_SetAI, 102
Traffic_SetBehaviour, 102
Traffic_SetDistance, 108
Traffic_SetDoor, 109
Traffic_SetEvent, 101
Traffic_SetLeaveParking, 109
Traffic_SetLights, 102
Traffic_SetPedestrian, 109
Traffic_SetPhysicalParameter, 108
Traffic_SetProfile, 103
Traffic_SetRoute, 104
Traffic_SetSignProperty, 106
Traffic_SetSigns, 105
Traffic_SetSpeed, 107
Traffic_SetSpeedLevel, 109
Traffic_SetTrackDeviation, 110
World_EnumSignSchemes, 117
World_EnumTiles, 116
World_FindWorldByName, 116
World_GetAbsoluteDistance, 113
World_GetFogSight, 112
World_GetInvisibleSourcePosition, 118
World_GetJunctionCenter, 115
World_GetLanes, 116
World_GetNextRoad, 119
World_GetRoadCount, 118
World_GetRoadCurvature, 120
World_GetRoadDistance, 113
World_GetRoadLength, 119
World_GetRoadToPlateVector, 114
World_GetTileSignVariant, 117
World_GetWeather, 111
World_GetWeatherParameter, 111
World_GetWorldName, 115
World_IsRoadOnInvisiblePlate, 117
World_IsRoadOnJunction, 119
World_Lookup, 121
World_LookupClear, 121
World_LookupCreate, 120
World_LookupLoad, 120
World_LookupRecord, 121
World_LookupRoadNumber, 114
World_LookupRoadTile, 114
World_LookupSave, 120
World_LookupSimplify, 122
World_SetFogSight, 112
World_SetNewLap, 115
World_SetNorth, 114
World_SetRoadFriction, 112
World_SetRoadMode, 114
World_SetWeather, 111
World_SetWeatherParameter, 111
World_SetWind, 112

GOSUB, 31
GOTO, 31

IF, 32
IFONCE, 33
INCLUDE, 128

LABEL, 31
LET, 30, 52
LETVEC, 30, 52
levels, 51

Maps, 185
WORLD_CITY2, 193
WORLD_MOTORWAY, 187
WORLD_MOUNTAIN, 189
WORLD_RACING, 199
WORLD_ROUNDCOURSE, 186
WORLD_SMALLTOWN, 190
WORLD_TOWN, 191
WORLD_TRAINER, 188
WORLD_WASTELAND, 192
mypos, 51

NAMESPACE, 127
NEXT, 35

PACKVEC, 30
PROC, 36

RESETONCE, 33
RETURN, 31
Road Position, 49

SLEEP, 35
STMT, 123
STRING, 29, 51
SWITCH, 34

TIMER, 29, 50

VAR, 29, 47
VEC, 29, 48
.angle, 49
.dist, 49
.offpitch, 49
.offroll, 49
.offx, 49
.offy, 49
.offyaw, 49
.offz, 49
.pitch, 49
.road, 49
.roll, 49
.track, 49
.v<n>, 49
.world, 49
.x, 49
.y, 49
.yaw, 49
.z, 49

WHILE, 34

Appendix E

List of Figures

3.1 Syntax Check Tool 26
3.2 Simulator Window 28
3.3 Example Code Flow 32
3.4 Map of WORLD_SMALLTOWN 39

4.1 Example Road Positions 50

5.1 Meaning of Angles 59

B.1 Map of WORLD_ROUNDCOURSE 186
B.2 Map of WORLD_MOTORWAY 187
B.3 Motorway Lane Widths 187
B.4 Map of WORLD_TRAINER 188
B.5 Map of WORLD_MOUNTAIN 189
B.6 Map of WORLD_SMALLTOWN 190
B.7 Map of WORLD_TOWN 191
B.8 Map of WORLD_WASTELAND 192
B.9 Map of WORLD_CITY2 193
B.10 Map of WORLD_RACING 199

Appendix F

List of Tables

3.1 Language Constants 18
3.2 Vector Component Naming Schemes 30
3.3 Operations for the CALC Command 31
3.4 Operators for Comparisons 33
3.5 Debug Key Codes 37

4.1 Vector Component Naming Schemes 49
4.2 Operations for the CALC Command 54

5.1 Argument Types 56

Appendix G

List of Listings

3.1 Title Syntax . . . 18
3.2 Block Syntax . . . 19
3.3 Scenario Syntax . . . 19
3.4 Example custom.txt . . . 19
3.5 Name Syntax . . . 21
3.6 NameNo Syntax . . . 21
3.7 NameId Syntax . . . 21
3.8 Maxlevel Syntax . . . 22
3.9 Defaultlevel Syntax . . . 22
3.10 Levels Syntax . . . 22
3.11 LevelsNo Syntax . . . 22
3.12 LevelsId Syntax . . . 23
3.13 LevelsSub Syntax . . . 23
3.14 Encoding Syntax . . . 23
3.15 Example Header . . . 23
3.16 BEGIN and END Syntax . . . 24
3.17 Checksum . . . 25
3.18 Command(s) . . . 27
3.19 Variable Declaration Syntax . . . 29
3.20 Assignment Syntax . . . 30
3.21 Simplified Assignment Syntax . . . 31
3.22 Jump Syntax . . . 31
3.23 IF/ELSE/ENDIF Syntax . . . 32
3.24 Condition Syntax . . . 33
3.25 Extended Condition Syntax . . . 33
3.26 SWITCH/CASE/ENDSWITCH Syntax . . . 34
3.27 SWITCH/CASE/ENDSWITCH Replacement . . . 34
3.28 WHILE/ENDWHILE Syntax . . . 35
3.29 WHILE/ENDWHILE Replacement . . . 35
3.30 FOR/NEXT Syntax . . . 35
3.31 FOR/NEXT Replacement . . . 35
3.32 Sleep Syntax . . . 35
3.33 Function Call Syntax . . . 36
3.34 Example Procedure . . . 36
3.35 Debug Command Syntax . . . 37

3.36 Function Call Syntax . . . 37
3.37 Scenarios\Custom\example01.sce . . . 38
3.38 Scenarios\Custom\example02.sce . . . 40
4.1 VAR Syntax . . . 47
4.2 VAR Examples . . . 47
4.3 VEC Syntax . . . 48
4.4 VEC Examples . . . 48
4.5 Vector Element by Suffix . . . 48
4.6 TIMER Syntax . . . 50
4.7 STRING Syntax . . . 51
4.8 Global Variables . . . 51
4.9 Local Global Variables . . . 51
4.10 Example String Constant . . . 52
4.11 Examples of Numeric Constants . . . 52
4.12 Using a Predefined Constant . . . 52
4.13 Assignment Syntax . . . 53
4.14 Example LETVEC Usage . . . 53
4.15 Alternatives to LETVEC . . . 53
4.16 Example PACKVEC Usage . . . 53
4.17 CALC Syntax . . . 54
5.1 Function Call Syntax . . . 55
5.2 Vector as Argument, Version 1 . . . 55
5.3 Vector as Argument, Version 2 . . . 55
5.4 Vector as Argument, Version 3 . . . 56
5.5 Storing Scalar Function Results . . . 56
5.6 Discarding Function Results . . . 57
5.7 Multiple Function Results . . . 57
5.8 Driver_SetAcceleratorPedal (Example Usage) . . . 64
5.9 Standard Scenario Opening Sequence . . . 66
5.10 System_SetImage (Example Usage) . . . 86
6.1 STMT Syntax . . . 123
6.2 leftlist variants . . . 123
6.3 vector selector syntax . . . 124
6.4 rightlist variants . . . 124
7.1 Extended name syntax . . . 127
7.2 Namespace example . . . 127
7.3 Library inclusion syntax . . . 128
7.4 Example.sci . . . 129
7.5 Example.sce . . . 129
7.6 WorldEditor.sce . . . 130
7.7 Simple Traffic Lights . . . 131
7.8 Standard Traffic Light Insertion . . . 131
7.9 Traffic Light Behind Junction . . . 132
8.1 Network Container Structure . . . 136
8.2 Network Function Call Syntax . . . 137