

Open Rails 1.5 Manual

Release 1.5.0.0

Open Rails Team

11 November 2022

Contents

1 Legal **1**
- 1.1 Warranty . . . 1
- 1.2 Properties Acknowledgment . . . 1
- 1.3 Copyright Acknowledgment and License . . . 1

2 New since previous version of Open Rails **2**
- 2.1 Headlines . . . 2
- 2.2 What's been added . . . 2
- 2.3 What's been improved . . . 2
- 2.4 Other changes . . . 3

3 Introduction **4**
- 3.1 What is Open Rails? . . . 4
- 3.2 About Open Rails . . . 4
- 3.3 Does Open Rails Require You to Have MSTS Installed? . . . 5
- 3.4 Community . . . 5
- 3.5 Raildriver Support . . . 5
- 3.6 Highlights of the Current Version . . . 5
  - 3.6.1 Focus on Compatibility . . . 5
  - 3.6.2 Focus on Operations . . . 6
  - 3.6.3 Focus on Realistic Content . . . 6

4 Use of MSTS Files by Open Rails **7**
- 4.1 Overview . . . 7
  - 4.1.1 Your MSTS Installation and Custom Installations for Open Rails . . . 7
- 4.2 MSTS Directories Used by Open Rails . . . 7
- 4.3 MSTS Files Used in Whole or Part by Open Rails . . . 8
  - 4.3.1 Route Files . . . 8
  - 4.3.2 Environment .env files . . . 8
  - 4.3.3 Activities . . . 8
- 4.4 Using a Non-MSTS Folder Structure . . . 9
- 4.5 Original MSTS Files Usually Needed for Added MSTS-Compatible Content . . . 9
  - 4.5.1 Original MSTS Files Usually Needed for a Non-MSTS-Folder Structure . . . 9

5 Getting Started **11**
- 5.1 Installation Profiles . . . 12
- 5.2 Updating OR . . . 12
- 5.3 Further General Buttons . . . 12
  - 5.3.1 Tools . . . 12
  - 5.3.2 Documents . . . 12
  - 5.3.3 Preliminary Selections . . . 12

5.4 Gaming Modes . . . 13
5.4.1 Activity, Explore and Explore with activity modes . . . 13
5.4.2 Timetable Mode . . . 14
5.4.3 Run . . . 15
5.5 Firewall . . . 15
5.5.1 Multiplayer Mode . . . 15
5.5.2 Replay . . . 16

6 Open Rails Options **17**
6.1 General Options . . . 18
6.1.1 Alerter in cab . . . 18
6.1.2 Graduated release air brakes . . . 19
6.1.3 Retainer valve on all cars . . . 19
6.1.4 Brake pipe charging rate . . . 19
6.1.5 Pressure unit . . . 19
6.1.6 Other units . . . 19
6.1.7 Use TCS scripts . . . 19
6.1.8 Overspeed Monitor . . . 20
6.2 Audio Options . . . 20
6.2.1 Sound Volume . . . 21
6.2.2 Sound Detail Level . . . 21
6.2.3 External Sound . . . 21
6.3 Video Options . . . 21
6.3.1 Dynamic shadows . . . 22
6.3.2 Shadow for all shapes . . . 22
6.3.3 Model instancing . . . 22
6.3.4 Overhead wire . . . 22
6.3.5 Double overhead wires . . . 22
6.3.6 Vertical sync . . . 22
6.3.7 Viewing distance . . . 23
6.3.8 Distant mountains . . . 23
6.3.9 Extend object maximum viewing distance to horizon . . . 23
6.3.10 Viewing vertical FOV . . . 24
6.3.11 World object density . . . 24
6.3.12 Ambient daylight brightness . . . 24
6.3.13 Anti-aliasing . . . 24
6.4 Simulation Options . . . 25
6.4.1 Advanced adhesion model . . . 25
6.4.2 Adhesion moving average filter size . . . 25
6.4.3 Break couplers . . . 25
6.4.4 Curve dependent speed limit . . . 26
6.4.5 At game start, Steam - pre-heat boiler . . . 26
6.4.6 At game start, Diesel - run engines . . . 26
6.4.7 Forced red at station stops . . . 26
6.4.8 Open/close doors on AI trains . . . 26
6.4.9 Location-linked passing path processing . . . 27
6.4.10 Simple control and physics . . . 27
6.5 Keyboard Options . . . 28
6.6 Data Logger Options . . . 29
6.7 Content Options . . . 30
6.8 System Options . . . 31
6.8.1 Language . . . 31
6.8.2 Update mode . . . 32
6.8.3 Windowed . . . 32
6.8.4 Window size . . . 32
6.8.5 Glass on in-game windows . . . 32
6.8.6 Control confirmations . . . 32
6.8.7 Web server port . . . 33

6.8.8 Automatically tune settings to keep performance level . . . 33
6.9 Experimental Options . . . 34
6.9.1 Super-elevation . . . 34
6.9.2 Show shape warnings . . . 36
6.9.3 Signal light glow . . . 36
6.9.4 Correct questionable braking parameters . . . 36
6.9.5 Activity randomization . . . 36
6.9.6 Activity weather randomization . . . 36
6.9.7 MSTS Environments . . . 37
6.9.8 Level of detail bias . . . 37
6.9.9 Adhesion proportional to rain/snow/fog . . . 37
6.9.10 Adhesion factor correction . . . 37
6.9.11 Adhesion factor random change . . . 37

7 Driving a Train **38**
7.1 Game Loading . . . 38
7.2 Entering the Simulation . . . 39
7.2.1 Cab Letter-Boxing . . . 39
7.3 Open Rails Driving Controls . . . 40
7.3.1 Throttle Control . . . 40
7.3.2 Dynamic Braking . . . 40
7.3.3 Combined Control . . . 40
7.3.4 Blended Dynamic Brake . . . 40
7.3.5 Refill . . . 41
7.3.6 Specific Features to Optimize Locomotive Driving . . . 41
7.3.7 Examples of Driving Controls . . . 41
7.4 Driving aids . . . 41
7.4.1 F1 Information Monitor . . . 41
7.4.2 F3 . . . 43
7.4.3 F4 Track Monitor . . . 43
7.4.4 F5 Train Driving Info . . . 45
7.4.5 F6 Siding and Platform Names . . . 49
7.4.6 F7 Train Names . . . 50
7.4.7 F8 Switch Monitor . . . 52
7.4.8 F9 Train Operations Monitor . . . 52
7.4.9 F10 Activity Monitor . . . 53
7.4.10 Compass Window . . . 54
7.4.11 Odometer . . . 54
7.4.12 Activity Evaluation . . . 54
7.4.13 Basic Head Up Display (HUD) . . . 58
7.4.14 Electric Locomotives – Additional information . . . 59
7.4.15 Steam Engine – Additional Information . . . 59
7.4.16 Multiplayer – Additional Information . . . 60
7.5 Map Window . . . 60
7.5.1 Dispatcher Tab . . . 61
7.5.2 Timetable Tab . . . 63
7.6 Additional Train Operation Commands . . . 66
7.6.1 Diesel Power On/Off . . . 66
7.6.2 Initialize Brakes . . . 67
7.6.3 Connect/Disconnect Brake Hoses . . . 67
7.6.4 Doors and Mirror Commands . . . 67
7.6.5 Wheelslip Reset . . . 67
7.6.6 Toggle Advanced Adhesion . . . 67
7.6.7 Request to Clear Signal . . . 67
7.6.8 Change Cab . . . 68
7.6.9 Train Oscillation . . . 68
7.6.10 Manual emergency braking release . . . 68
7.7 Engaging a turntable or a transfertable . . . 68

7.8 Loading and Unloading Containers . . . 70
7.9 Autopilot Mode . . . 71
7.10 Changing the Train Driven by the Player . . . 71
7.10.1 General . . . 71
7.10.2 Switching to a static train . . . 74
7.10.3 Waiting point considerations . . . 75
7.11 Changing the View . . . 75
7.12 Toggling Between Windowed Mode and Full-screen . . . 77
7.13 Modifying the Game Environment . . . 77
7.13.1 Time of Day . . . 77
7.13.2 Weather . . . 78
7.13.3 Modifying Weather at Runtime . . . 78
7.13.4 Randomized Weather in activities . . . 78
7.13.5 Season . . . 78
7.14 Activity randomization . . . 78
7.15 Screenshot - Print Screen . . . 79
7.16 Suspending or Exiting the Game . . . 80
7.17 Save and Resume . . . 80
7.17.1 Saves from Previous OR Versions . . . 82
7.18 Save and Replay . . . 82
7.18.1 Exporting and Importing Save Files . . . 84
7.19 Analysis Tools . . . 84
7.19.1 Extended HUD for Consist Information . . . 84
7.19.2 Extended HUD for Locomotive Information . . . 85
7.19.3 Extended HUD for Brake Information . . . 85
7.19.4 Extended HUD for Train Force Information . . . 86
7.19.5 Extended HUD for Dispatcher Information . . . 87
7.19.6 Extended HUD for Debug Information . . . 90
7.19.7 Viewing Interactive Track Items . . . 91
7.19.8 Viewing Signal State and Switches . . . 92
7.19.9 Sound Debug Window . . . 92
7.20 OpenRailsLog.txt Log file . . . 93
7.21 Code-embedded Logging Options . . . 94
7.22 Testing in Autopilot Mode . . . 94

8 Open Rails Physics **95**
8.1 Train Cars (WAG, or Wagon Part of ENG file) . . . 95
8.1.1 Resistive Forces . . . 95
8.1.2 Coupler Slack . . . 96
8.1.3 Hot Wheel Bearings . . . 96
8.1.4 Derailment Coefficient . . . 97
8.1.5 Adhesion of Locomotives – Settings Within the Wagon Section of ENG files . . . 97
8.2 Engine – Classes of Motive Power . . . 99
8.2.1 Diesel Locomotives . . . 99
8.2.2 Electric Locomotives . . . 107
8.2.3 Steam Locomotives . . . 108
8.2.4 Unpowered Control Car . . . 126
8.3 Multiple Units of Locomotives in Same Consist . . . 126
8.3.1 Distributed Power . . . 126
8.3.2 Engines of AI Trains . . . 128
8.4 Open Rails Braking . . . 128
8.4.1 Train Brake Controller Positions . . . 129
8.4.2 Brake Shoe Adhesion . . . 131
8.4.3 Train Brake Pipe Losses . . . 132
8.4.4 Wheel Skidding due to Excessive Brake Force . . . 132
8.4.5 Using the F5 HUD Expanded Braking Information . . . 133
8.4.6 Dynamic Brakes . . . 135
8.4.7 Native Open Rails Braking Parameters . . . 135

8.4.8 Brake Retainers . . . 136
8.4.9 Emergency Brake Application Key . . . 136
8.4.10 Automatic Vacuum Brakes . . . 136
8.4.11 Non Automatic Vacuum Brakes . . . 138
8.4.12 Manual Brakes . . . 139
8.4.13 Steam Brakes . . . 139
8.4.14 Wheel Slide Protection . . . 139
8.4.15 SME (sometimes also called SEM) Brake System . . . 140
8.5 Dynamically Evolving Tractive Force . . . 140
8.6 Curve Resistance - Theory . . . 140
8.6.1 Introduction . . . 140
8.6.2 Factors Impacting Curve Friction . . . 141
8.6.3 Impact of Rigid Wheelbase . . . 141
8.6.4 Impact of Super Elevation . . . 141
8.6.5 Calculation of Curve Resistance . . . 142
8.6.6 Calculation of Curve Speed Impact . . . 143
8.6.7 Further background reading . . . 143
8.7 Curve Resistance - Application in OR . . . 143
8.7.1 OR Parameter Values . . . 143
8.7.2 OR Default Values . . . 144
8.7.3 Typical Rigid Wheelbase Values . . . 144
8.8 Super Elevation (Curve Speed Limit) – Theory . . . 144
8.8.1 Introduction . . . 144
8.8.2 19th & 20th Century vs Modern Day Railway Design . . . 145
8.8.3 Centrifugal Force . . . 145
8.8.4 Effect of Centrifugal Force . . . 145
8.8.5 Use of Super Elevation . . . 146
8.8.6 Limitation of Super Elevation in Mixed Passenger & Freight Routes . . . 146
8.8.7 Limitation of Super Elevation in High Speed Passenger Routes . . . 146
8.8.8 Maximum Curve Velocity . . . 147
8.8.9 Limitation of Velocity on Curved Track at Zero Cross Level . . . 147
8.8.10 Height of Centre of Gravity . . . 147
8.8.11 Calculation of Curve Velocity . . . 147
8.8.12 Typical Super Elevation Values & Speed Impact – Mixed Passenger & Freight Routes 148
8.8.13 Typical Super Elevation Values & Speed Impact – High Speed Passenger Routes . 148
8.9 Super Elevation (Curve Speed Limit) Application in OR . . . 148
8.9.1 OR Super Elevation Parameters . . . 148
8.9.2 OR Super Elevation Default Values . . . 149
8.10 Tunnel Friction – Theory . . . 149
8.10.1 Introduction . . . 149
8.10.2 Factors Impacting Tunnel Friction . . . 149
8.10.3 Importance of Tunnel Profile . . . 150
8.10.4 Calculation of Tunnel Resistance . . . 150
8.11 Tunnel Friction – Application in OR . . . 151
8.11.1 OR Parameters . . . 151
8.11.2 OR Defaults . . . 151
8.12 Wind Resistance . . . 151
8.13 Trailing Locomotive Resistance . . . 152
8.14 OR-Specific *Include Files* for Modifying MSTS File Parameters . . . 152
8.14.1 Modifications to .eng and .wag Files . . . 152
8.15 Common locomotive subsystems . . . 155
8.15.1 Battery switch . . . 155
8.15.2 Master key . . . 155
8.15.3 Service retention . . . 156
8.15.4 Electric train supply . . . 156
8.15.5 Train Control System . . . 157
8.15.6 Train Derailment . . . 158
8.16 EOT - End of train device . . . 158

8.16.1 General . . . 158
8.16.2 How to define an EOT . . . 158
8.16.3 How to attach and detach an EOT at the end of a train . . . 159
8.16.4 How to arm (enable) or disarm a one-way or two-way EOT . . . 160
8.16.5 Emergency brake through EOT . . . 161

9 Further Open Rails Rolling Stock Features 162
9.1 Train Engine Lights . . . 162
9.2 Tilting trains . . . 162
9.3 Freight animations and pickups . . . 163
9.3.1 OR implementation of MSTS freight animations and pickups . . . 163
9.3.2 OR specific freight animations and pickups . . . 163
9.3.3 Physics Variation with Loads . . . 167
9.4 Container management . . . 170
9.4.1 General . . . 170
9.4.2 How to define container data . . . 171
9.4.3 Pre-setting a .wag file to accommodate containers . . . 171
9.4.4 Allocation of the containers on the wagons . . . 172
9.4.5 How to allocate containers on wagons at start of game . . . 173
9.4.6 Container Station . . . 174
9.5 Multiple passenger viewpoints . . . 183
9.6 Bell animation . . . 183
9.7 Coupler and Airhose Animation . . . 184
9.8 C# engine scripting . . . 184
9.8.1 Developing scripts with Visual Studio . . . 186
9.8.2 Brake controller . . . 188
9.8.3 Circuit breaker . . . 188
9.8.4 Traction cut-off relay . . . 188
9.8.5 Diesel and electric power supply . . . 189
9.8.6 Passenger car power supply . . . 189
9.8.7 Train Control System . . . 190
9.8.8 Helper classes . . . 194

10 Open Rails Train Operation 196
10.1 Open Rails Activities . . . 196
10.1.1 Player Paths, AI Paths, and How Switches Are Handled . . . 196
10.2 Open Rails AI . . . 197
10.3 Control Mode . . . 197
10.3.1 Auto Mode . . . 197
10.3.2 Manual Mode . . . 198
10.3.3 Out-of-Control Mode . . . 200
10.3.4 Explorer Mode . . . 200
10.4 Track Access Rules . . . 200
10.5 Deadlock Processing . . . 200
10.6 Reversal Points . . . 201
10.7 Waiting Points . . . 202
10.7.1 General . . . 202
10.7.2 Absolute Waiting Points . . . 202
10.8 Signals at Station Stops . . . 202
10.9 Speedposts and Speed Limits Set by Signals . . . 203
10.10 Further Features of AI Train Control . . . 203
10.11 Location-linked Passing Path Processing . . . 203
10.12 Other Comparisons Between Running Activities in ORTS or MSTS . . . 205
10.12.1 End of run of AI trains . . . 205
10.12.2 Default Performance and Performance Parameters . . . 205
10.12.3 Calculation of Train Speed Limit . . . 205
10.12.4 Start of Run of AI train in a Section Reserved by Another Train . . . 206
10.12.5 Stop Time at Stations . . . 206

10.12.6 Restricted speed zones defined in activities . . . 206
10.13 Extended AI Train Shunting . . . 207
10.13.1 General . . . 207
10.13.2 Activity Design for Extended AI Train Shunting Functions . . . 207
10.14 Signal related files . . . 211
10.14.1 SignalNumClearAhead . . . 211
10.14.2 Location of OR-specific sigcfg and sigscr files . . . 212
10.14.3 OR-unique values for SignalNumClearAhead () . . . 212
10.14.4 C# signal scripting . . . 212
10.15 OR-specific Signaling Functions . . . 213
10.15.1 SPEED Signals – a New Signal Function Type . . . 213
10.15.2 Approach control functions . . . 215
10.15.3 TrainHasCallOn, TrainHasCallOn_Advanced Functions . . . 220
10.15.4 TrainHasCallOn_Restricted, TrainHasCallOn_Restricted_Advanced Functions . . . 221
10.15.5 Signalling Function NEXT_NSIG_LR . . . 222
10.15.6 Signalling Function HASHEAD . . . 223
10.15.7 Signalling flag OR_NOSPEEDREDUCTION . . . 223
10.16 OR-Specific Additions to Activity Files . . . 223
10.16.1 Manually modifying the .act file . . . 223
10.16.2 Using the TSRE5 activity editing capabilities . . . 224
10.16.3 Generating an extension activity file . . . 224
10.16.4 No Halt by Activity Message Box . . . 224
10.16.5 AI Train Horn Blow . . . 225
10.16.6 AI Horn Blow at Level Crossings . . . 225
10.16.7 Location Event triggered by AI Train . . . 225
10.16.8 Location Event and Time Event Sound File . . . 226
10.16.9 Weather Change Activity Event . . . 227
10.16.10 AI train Waiting Point modification through event . . . 228
10.16.11 Old formats . . . 229

11 Timetable Mode **230**
11.1 Introduction . . . 230
11.2 General . . . 230
11.2.1 Data definition . . . 230
11.2.2 File structure . . . 231
11.2.3 Timetable groups . . . 231
11.2.4 Pool files . . . 231
11.2.5 Weather files . . . 231
11.2.6 File and train selection . . . 231
11.3 Timetable Definition . . . 232
11.3.1 General . . . 232
11.3.2 Column definitions . . . 232
11.3.3 Row definitions . . . 232
11.3.4 Timing details . . . 233
11.4 Timetable Data Details . . . 233
11.4.1 Timetable Description . . . 233
11.4.2 Train Details . . . 233
11.4.3 Location Details . . . 233
11.4.4 Timing Details . . . 233
11.4.5 Special Columns . . . 234
11.4.6 Special Rows . . . 234
11.4.7 Control Commands . . . 236
11.5 Additional Notes on Timetables . . . 252
11.5.1 Static Trains . . . 252
11.5.2 Processing of #dispose Command For Player Train . . . 252
11.5.3 Termination of a Timetable Run . . . 252
11.5.4 Calculation of Running Delay . . . 253
11.5.5 No Automatic Coupling . . . 253

11.5.6 Use of Consists in Shunting Commands . . . 253
11.5.7 Signalling Requirements and Timetable Concept . . . 253
11.5.8 Known Problems . . . 255
11.6 Storing Trains with Pools . . . 256
11.6.1 Additional Notes . . . 256
11.7 Pool Definition . . . 256
11.7.1 Non-Turntable Pools . . . 257
11.7.2 Turntable Pools . . . 258
11.8 Changing Weather . . . 261
11.8.1 “Clear” event . . . 261
11.8.2 “Precipitation” event . . . 262
11.8.3 “Fog” event . . . 262
11.9 Example of a Timetable File . . . 263
11.10 What tools are available to develop a Timetable? . . . 264

12 Open Rails Multi-Player 265
12.1 Goal . . . 265
12.2 Getting Started . . . 265
12.3 Requirements . . . 265
12.4 Technical Issues . . . 266
12.5 Technical Support . . . 266
12.6 Starting a Multi-Player Session . . . 266
12.6.1 Starting as Dispatcher . . . 266
12.6.2 Starting as Client . . . 267
12.7 In-Game Controls . . . 267
12.8 Summary of Multi-Player Procedures . . . 270
12.9 Possible Problems . . . 270
12.10 Using the Public Server . . . 271
12.10.1 Additional info on using the Public Server . . . 271
12.11 Save and resume . . . 271
12.12 Setting up a Server from Your Own Computer . . . 271
12.12.1 IP Address . . . 272
12.12.2 Port Forwarding . . . 273

13 Open Rails Sound Management 277
13.1 OpenRails vs. MSTS Sound Management . . . 277
13.2 .sms Instruction Set . . . 277
13.2.1 Discrete Triggers . . . 278
13.2.2 OR-Specific Discrete Triggers . . . 279
13.2.3 Variable Triggers . . . 283
13.2.4 Sound Loop Management . . . 284
13.2.5 Testing Sound Files at Runtime . . . 284
13.3 Discrete triggers for container cranes . . . 284
13.4 Automatic switch and curve squeal track sound . . . 284
13.5 Override % of external sound heard internally for a specific trainset . . . 286

14 Open Rails Cabs 287
14.1 2D Cabs . . . 287
14.1.1 ETCS circular speed gauge . . . 287
14.1.2 Battery switch . . . 288
14.1.3 Master key . . . 288
14.1.4 Service retention . . . 289
14.1.5 Electric train supply . . . 289
14.1.6 Controls to switch on and off diesel engines . . . 289
14.1.7 Cab radio . . . 291
14.1.8 Cab light . . . 291
14.1.9 Dedicated buttons for brake controllers . . . 291
14.1.10 Signed Traction Braking control . . . 292
14.1.11 Signed Traction Total Braking control . . . 292

14.1.12 Odometer controls . . . 292
14.1.13 Distributed Power . . . 293
14.1.14 EOT (End of Train device) . . . 294
14.1.15 Animated 2D Wipers . . . 297
14.1.16 Control Labels . . . 298
14.1.17 Multiple screen pages on displays . . . 298
14.1.18 Further OR cab controls . . . 299
14.1.19 Cab controls for generic items . . . 299
14.1.20 High-resolution Cab Backgrounds and Controls . . . 299
14.1.21 Configurable Fonts . . . 302
14.1.22 Rotation of Gauges and Digital controls . . . 302
14.2 3D cabs . . . 304
14.2.1 Development Rules . . . 304
14.2.2 A Practical Development Example For a Digital Speedometer . . . 305
14.2.3 FUEL_GAUGE for steam locomotives . . . 305
14.2.4 Distributed power display . . . 306
14.2.5 Alignment for digital controls . . . 307

15 OR-Specific Route Features 308
15.1 Modifications to .trk Files . . . 308
15.2 Repetition of Snow Terrain Textures . . . 308
15.3 Snow Textures with Night Textures . . . 309
15.4 Operating Turntables and Transfertables . . . 309
15.4.1 Turntables . . . 309
15.4.2 Transfertables . . . 311
15.4.3 Locomotive and wagon elevators . . . 313
15.4.4 Path laying and operation considerations . . . 314
15.5 .w File modifiers . . . 314
15.6 Multiple car spawner lists . . . 315
15.7 Car spawners used for walking people . . . 316
15.8 Route specific TrackSections and TrackShapes . . . 317
15.9 Overhead wire extensions . . . 318
15.9.1 Double wire . . . 318
15.9.2 Triphase lines . . . 318
15.10 Loading screen . . . 319
15.11 MSTS-Compatible semaphore indexing . . . 319
15.12 Automatic door open/close on AI trains . . . 319
15.13 Removing forest trees from tracks and roads . . . 320
15.14 Multiple level crossing sounds . . . 320
15.15 Defining Curve Superelevation . . . 321
15.16 Overhead (catenary) wire . . . 321
15.17 Fading signal lamps . . . 321
15.18 Animated clocks . . . 322
15.18.1 Overview . . . 322
15.18.2 Details . . . 322

16 Developing OR Content 325
16.1 Rolling Stock . . . 325
16.2 Routes . . . 325
16.3 Activities . . . 326
16.4 Parameters and Tokens . . . 326
16.5 Testing and Debugging Tools . . . 326
16.6 Open Rails Best Practices . . . 326
16.6.1 Polys vs. Draw Calls – What's Important . . . 326
16.7 Support . . . 327

17 Version 1.3 Known Issues 328
17.1 Empty Effects Section in .eng File . . . 328
17.2 Curly brackets in file sigscr.dat . . . 328

17.3 Spurious emergency braking in Timetable mode . . . 328

18 In Case Of Malfunction **329**
18.1 Introduction . . . 329
18.2 Overview of Bug Types . . . 329
18.3 Maybe-Bugs . . . 329
18.4 Decided bugs . . . 330
18.5 Additional Notes . . . 331
18.6 Summary: Bug Report Checklists . . . 331
18.7 Bug Status in Launchpad . . . 332
18.8 Disclaimer . . . 332

19 Open Rails Software Platform **333**
19.1 Architecture . . . 333
19.2 Open Rails Game Engine . . . 334
19.3 Frames per Second (FPS) Performance . . . 334
19.4 Game Clock and Internal Clock . . . 335
19.5 Resource Utilization . . . 335
19.6 Multi-Threaded Coding . . . 335
19.7 Web Server . . . 336
19.7.1 Sample Web Pages . . . 336
19.7.2 Application Programming Interfaces (APIs) . . . 338

20 Plans and Roadmap **340**
20.1 User Interface . . . 340
20.2 Operations . . . 340
20.3 Open Rails Route Editor . . . 341

21 Acknowledgements **342**

22 Appendices **343**
22.1 Units of Measure . . . 343
22.2 Folders used by Open Rails . . . 346
22.3 Signal Functions . . . 346
22.3.1 Original MSTS Functions . . . 346
22.3.2 Extended MSTS Functions . . . 347
22.3.3 SIGNAL IDENT Functions . . . 347
22.3.4 Signal SubObject functions . . . 348
22.3.5 Approach Control Functions . . . 348
22.3.6 CallOn Functions . . . 349
22.3.7 SignalNumClearAhead Functions . . . 350
22.3.8 Local signal variables . . . 350
22.3.9 Functions for Normal Head Subtype . . . 351
22.3.10 Functions to verify full or partial route clearing . . . 352
22.3.11 Miscellaneous functions . . . 352
22.3.12 Timing Functions . . . 353
22.4 OR-specific additions to SIGCFG files . . . 353
22.4.1 General definitions . . . 353
22.4.2 ORTSSignalFunctions . . . 353
22.4.3 ORTSNormalSubtypes . . . 354
22.4.4 Signal Type definitions . . . 354
22.4.5 Light switch . . . 355
22.4.6 Script Function . . . 355
22.4.7 Normal Subtype . . . 355
22.4.8 Approach Control Settings . . . 356
22.4.9 Signal aspect parameters . . . 356
22.4.10 SPEED Signal . . . 357

Index **358**

CHAPTER 1

Legal

1.1 Warranty

NO WARRANTIES: openrails.org disclaims any warranty, at all, for its Software. The Open Rails software and any related tools, or documentation is provided “as is” without warranty of any kind, either express or implied, including suitability for use. You, as the user of this software, acknowledge the entire risk from its use. See the license for more details.

1.2 Properties Acknowledgment

Open Rails, Open Rails Transport Simulator, ORTS, openrails.org, Open Rails symbol and associated graphical representations of Open Rails are the property of openrails.org. All other third party brands, products, service names, trademarks, or registered service marks are the property of and used to identify the products or services of their respective owners.

1.3 Copyright Acknowledgment and License

© 2009-2022 openrails.org. This document is part of Open Rails. Open Rails is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.

You should have received a copy of the GNU General Public License as part of the Open Rails distribution in `/Copying.txt`. If not, see http://www.gnu.org/licenses/.

CHAPTER 2

New since previous version of Open Rails

A summary of the new and improved features can be found below. In addition, over 170 bugs have been fixed in this release.

Please keep reporting bugs and suggesting new features so Open Rails can continue to improve.

2.1 Headlines

- Container Loading and Unloading

2.2 What's been added

- End Of Train (EOT) devices
- Distributed Power display, in-cab and on webpage
- Progress bar for timetable "pre-run"
- HUD extended for diesel-mechanical loco

2.3 What's been improved

- 64-bit compatibility lifting the 3GB RAM limit
- Better use of VRAM

2.4 Other changes

- F5 key now opens Driver Info window. For HUD, use Alt+F5
- Some trial options are now always active including:
  - map window (Ctrl+9)
  - de-brief evaluation (F1)
  - web server

CHAPTER 3

Introduction

3.1 What is Open Rails?

Open Rails software (OR) is a community developed and maintained project from openrails.org. Its objective is to create a new transport simulator platform that is first, compatible with routes, activities, consists, locomotives, and rolling stock created for Microsoft Train Simulator (MSTS); and secondly, a platform for future content creation freed of the constraints of MSTS (in this manual MSTS means MSTS with MSTS Bin extensions, if not explicitly stated in a different way).

Our goal is to enhance the railroad simulation hobby through a community-designed and supported platform built to serve as a lasting foundation for an accurate and immersive simulation experience. By making the source code of the platform freely available under the GPL license, we ensure that OR software will continually evolve to meet the technical, operational, graphical, and content building needs of the community. Open architecture ensures that our considerable investment in building accurate representations of routes and rolling stock will not become obsolete. Access to the source code eliminates the frustration of undocumented behavior and simplifies understanding the internal operation of the simulator without the time-consuming trial and error-prone experimentation typically needed today.

Open Rails software is just what the name implies – a railroad simulation platform that's open for inspection, open for continuous improvement, open to third parties and commercial enterprises, open to the community and, best of all, an open door to the future.

3.2 About Open Rails

To take advantage of almost a decade of content developed by the train simulation community, Open Rails software is an independent game platform that has backward compatibility with MSTS content. By leveraging the community's knowledge base on how to develop content for MSTS, Open Rails software provides a rich environment for both community and payware contributors.

The primary objective of the Open Rails project is to create a railroad simulator that will provide *true to life* operational experience. The Open Rails software is aimed at the serious train simulation hobbyist; someone who cares about locomotive physics, train handling, signals, AI behavior, dispatching, and most of all running trains in a realistic, prototypical manner. While the project team will strive to deliver an unparalleled graphical experience, *eye candy* is not the primary objective of Open Rails software.

By developing a completely new railroad simulator, Open Rails software offers the potential to better utilize current and next generation computer resources, like graphics processing units (GPUs), multi-core

CPUs, advanced APIs such as PhysX, and widescreen monitors, among many others. The software is published so that the user community can understand how the software functions to facilitate feedback and to improve the capabilities of Open Rails software.

Open Rails is published under the GPL license which is "copyleft"[1] to ensure that the source code always remains publicly available.

3.3 Does Open Rails Require You to Have MSTS Installed?

No, it is not *required* by the Open Rails software itself. However. a great deal of the content accessed by OR includes files originally delivered with MSTS (e.g., tracks or general sounds). These files must be obtained from a properly licensed installation of MSTS.

There are examples where no MSTS content is used (often payware) and in such cases Open Rails does not require MSTS to be installed. Read *here* for further detail.

In all cases, all content files (original or MSTS) must be organized in an MSTS-compatible folder structure. Such a structure is described *here*. In this manual such a folder structure will be called an *MSTS installation* for convenience, even if this wording is not completely correct.

A proof that Open Rails itself does not need an MSTS installation at all to run is *this route <http://www.burrinjuck.coalstonewcastle.com.au/route/route-install/>*.

3.4 Community

Open Rails software is offered without technical support. Users are encouraged to use their favorite train simulation forums to get support from the community. We suggest:

- Train-Sim.Com
- UK Train Sim
- Elvas Tower

For users interested in multiplayer sessions, a forum is set up for you to seek and announce hosting sessions: http://www.tsimserver.com.

3.5 Raildriver Support

Open Rails offers native support for the RailDriver Desktop Train Cab Controller. Instructions for setting up RailDriver for Open Rails are included in the Installation Manual that is included with the Open Rails Installer, or it can be downloaded separately from the Open Rails website.

3.6 Highlights of the Current Version

3.6.1 Focus on Compatibility

With Release 1.0 the published goal was reached to make as much of the existing MSTS content as possible run in Open Rails. The development team's initial focus has been to provide a fairly complete visual replacement for MSTS that effectively builds on that content, achieving all the compatibility that is worthwhile, at the same time delivering a system which is faster and more robust than MSTS.

[1] https://gnu.org/copyleft

3.6.2 Focus on Operations

Release 1.1 cleared the way to improving on MSTS in many ways which can be summarized as moving from Foundation to Realism and eventually to Independence. That release already included features that are beyond MSTS; non-player trains can have movement orders (i.e. pickups, drop offs) based on files in MSTS format. The player can change the driven train. Multi-user operation has also been available for some time.

3.6.3 Focus on Realistic Content

The physics underlying adhesion, traction, engine components and their performance are based on a world-class simulation model that takes into account all of the major components of diesel, electric and steam traction. Release 1.2 refines elements such as braking, where braking friction now varies with speed, over-braking which now leads to skidding and wheel-slip is now modelled for steam locos too.

Existing models that do not have the upgraded Open Rails capabilities continue, of course, to perform well.

In Version 1.x releases, ancillary programs (*tools*) are also delivered, including:

- Track Viewer: a complete track viewer and path editor
- Timetable Editor: a tool for preparing *Timetables*

CHAPTER 4

Use of MSTS Files by Open Rails

4.1 Overview

4.1.1 Your MSTS Installation and Custom Installations for Open Rails

Open Rails reads only the content folders in each of the MSTS installations you choose to identify for it and will do so without modifying any of those files. None of the MSTS program folders are used and no changes to the MSTS directory tree are required.

Open Rails may also be used to read a non-MSTS directory structure that you create.

This document uses the term Root Folder to designate the parent folder of any MSTS or OR-Specific directory tree (.e.g, `\Train Simulator` is the *Root Folder* for MSTS).

4.2 MSTS Directories Used by Open Rails

Open Rails software reads and uses all of the data found in many MSTS directories:

```
\Consists
\Paths
\Services
\Shapes
\Sounds
\Textures
\Terrtex
\Tiles
\Traffic
\Trainset
\World
```

Open Rails uses a file parser to read the MSTS files and will locate many errors that are missed or unreported by the MSTS software or by other utilities. In most cases, the Open Rails software will ignore the error in the file and run properly. Open Rails software logs these errors in a log file on the user's desktop. This log file may be used to correct problems identified by the Open Rails software. The parser will also correct some of the problems that stumped MSTS. For example, if a texture is missing Open Rails will substitute a neutral gray texture and continue.

4.3 MSTS Files Used in Whole or Part by Open Rails

4.3.1 Route Files

Open Rails software uses some of the data in several MSTS Route files, depending on the MSTS features supported by Open Rails:

- Route Database file (.rdb) – CarSpawner is supported.
- Reference File (.ref) – a Route Editor is well under way.
- Track Database file (.tdb) – supported
- Route File (.trk) – Level Crossings and overhead wires are supported.
- Sigcfg (.dat) file – Signal & scripting capabilities are supported.
- Sigscr (.dat) file – Signal & scripting capabilities are supported.
- Speedpost (.dat) file – Supported
- Spotter (.dat) file – Supported
- Ssource (.dat) file – Supported
- Telepole (.dat) file – Supported
- Tsection (.dat) file – Supported
- Ttype (.dat) file – Supported
- Hazards (.haz) file – Supported

4.3.2 Environment .env files

Open Rails software does not support advanced water dynamic effects.

OR Defined Weather

Open Rails uses its own sky, cloud, sun, moon and precipitation effects developed exclusively for it. When using the *Explore Route* feature you may choose season, weather, and time of day. When using the *Run Activity* feature they are read from the activity file.

OR Weather using MSTS Compatibility

Open Rails can replace MSTS Environmental displays by its own (e.g., Kosmos)

4.3.3 Activities

Many passenger and freight activities created using the MSTS activity editor run without problems in Open Rails.

Some Activities created using the MSTS activity editor will have slightly different behavior as compared to running in MSTS. This is often due to slightly different train performance resulting from differences in how each simulator handles train physics.

A few activities fail to run at all. This appears to be due to the creativity of Activity Designers who have found ways to do things wholly unanticipated by the Open Rails Team. As these are discovered the Open Rails team will record the bug for future correction.

4.4 Using a Non-MSTS Folder Structure

Open Rails uses a subset of the MSTS folder structure to run. You must create a root folder of any suitable name and it must contain four folders, together with their related sub-folders:

```
\GLOBAL
\ROUTES
\TRAINS
\SOUND
```

No other files or folders are required in the root folder. Within the `\GLOBAL` folder two sub-folders are required:

```
\SHAPES
\TEXTURES
```

Within the `\TRAINS` folder two subfolders are required:

```
\CONSISTS
\TRAINSETS
```

4.5 Original MSTS Files Usually Needed for Added MSTS-Compatible Content

4.5.1 Original MSTS Files Usually Needed for a Non-MSTS-Folder Structure

A number of MSTS folders and files must be placed into any OR-Specific installation you have created. These may be obtained from your own MSTS Installation or, as noted below, from Train Sim Forums

\GLOBAL

Within the `\GLOBAL` folder only the file tsection.dat is required. The most current version is best and it can be downloaded from many Train Sim forums. Files sigcfg.dat and sigscr.dat are needed if there are routes that don't have their own specific files with the same names in their root folder.

\GLOBAL\SHAPES

Many routes use specific track sets, like XTRACKS, UK-finescale etc.

Routes which solely use such sets do not need any of the original MSTS files from GLOBAL, as all required files come from the relevant track set. These sets can be downloaded from many Train Sim forums. There are also many routes using super-sets of the original MSTS track sets. These routes will need some or all the files contained in the `SHAPES` and `TEXTURES` subfolders within the `GLOBAL` folder of your MSTS installation.

\TRAINS

Requirements are similar to routes. Again, only the folders for the trainsets which are actually used are required, but many third-party trainsets refer to original MSTS files like cabviews and, in particular, sound files. Many consists refer to engines or wagons from the original MSTS routes but those can be easily replaced with other engines or wagons.

\SOUND

Only very few routes provide a full new sound set, so the original files included in this folder are usually needed.

\ROUTES

Once all the above directories are populated with files you need only the specific route folder placed into `\Routes` to run Open Rails from a non-MSTS directory.

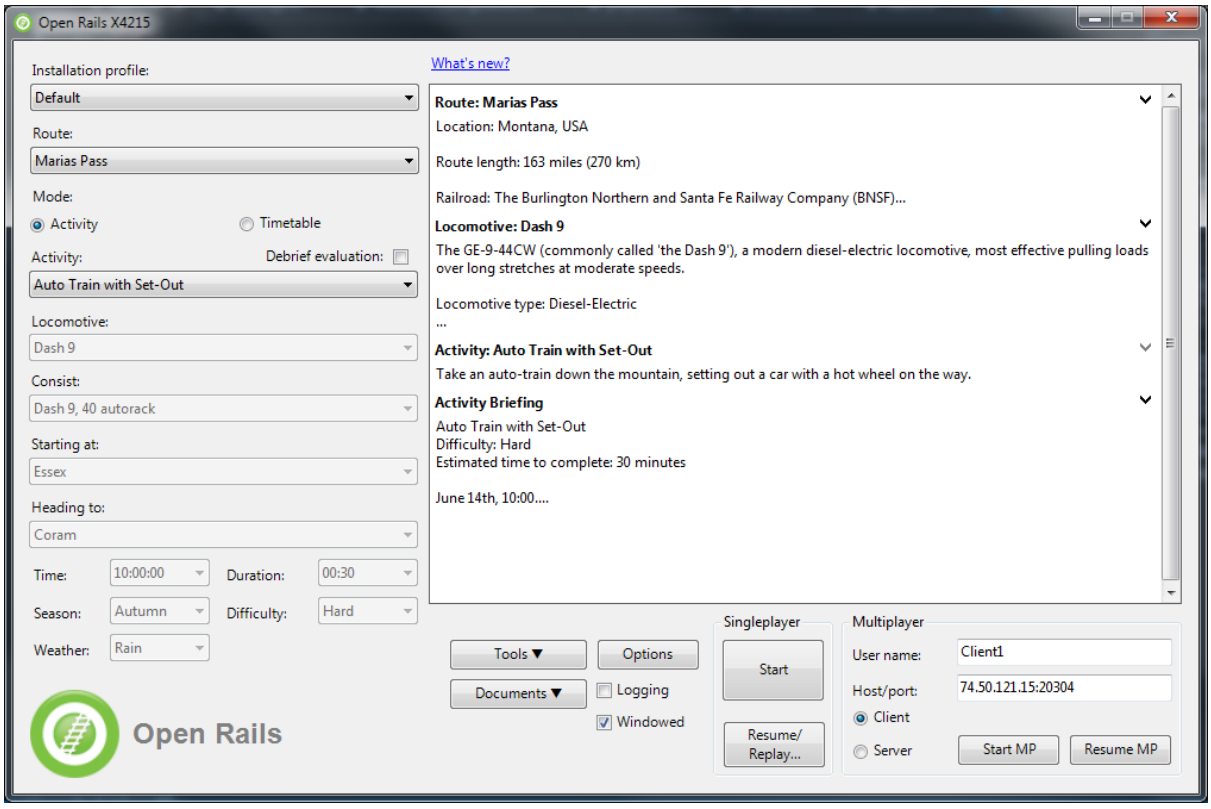
Note that many routes – in particular freeware routes – use content from the original MSTS routes, and therefore when installing new routes you may find their installation requires files from the original MSTS routes in order to be properly installed.

CHAPTER 5

Getting Started

After having successfully installed Open Rails (see the Installation Manual), to run the game you must double-click on the Open Rails icon on the desktop, or on the OpenRails.exe file.

The OpenRails main window will appear. If you have an MSTS installation in place, this will be displayed as your available installation profile.



If not, then you must download some content and add it as an Installation Profile.

5.1 Installation Profiles

Each profile may be a folder containing one or more routes, or an optional MSTS installation.

In the case where you already have an MSTS installation (see paragraph *Does Open Rails need MSTS to run?* for a precise definition of a MSTS installation) OR should already correctly point to that installation. To check this, you should initially see under `Installation Profile` the string `- Default -`. Under `Route` you should see the name of one of the MSTS routes in your MSTS installation.

You can easily add, remove or move other content profiles and select among them (e.g. if you have any so-called `mini-routes` installed.). Click on the `Options` button and select the `Content` tab. See the *Content Options* discussed below for more instructions.

5.2 Updating OR

When a new release of OR is available and your computer is online, a link `Update to xnnnn` appears in the upper right corner. The string `xnnnn` is the release number of the newest release that matches your selected level of update. Various level of updates called Update Channels are available. You may choose the desired level in the `Options-Update` window, described below.

When you click on the update link OR will download and install the new release. In this way your version of Open Rails is always up to date. Note, however, that previously saved games may not be compatible with newer versions, as described *here*.

Clicking the link `What's new`? in the upper centre part of the main menu window will connect to a website that summarizes the most recent changes to the OR program.

5.3 Further General Buttons

5.3.1 Tools

By clicking this button you get access to the ancillary tools (see *here*).

5.3.2 Documents

This button becomes selectable only if you have at least once updated to a testing version or to a stable version greater than 1.0. By clicking this button you get immediate access to the OR documentation.

5.3.3 Preliminary Selections

Firstly, under `Route:` select the route on which you wish to run.

If you check the `Logging` checkbox, Open Rails will generate a log file named `OpenRailsLog.txt` that resides on your desktop. This log file is very useful to document and investigate malfunctions.

At every restart of the game (that is, after clicking `Start` or `Server` or `Client`) the log file is cleared and a new one is generated.

If you wish to fine-tune Open Rails for your system, click on the `Options` button. See the Chapter: *Open Rails Options* which describes the extensive set of OR options. It is recommended that you read this chapter.

5.4 Gaming Modes

One of the plus points of Open Rails is the variety of gaming modes you can select.

5.4.1 Activity, Explore and Explore with activity modes

As a default you will find the radio button `Activity` selected in the start window, as *above*.

This will allow you to run an activity or run on of two types of explore mode.

If you select `- Explore Route -` (first entry under `Activity:`), you will also have to select the consist, the path, the starting time, the season and the weather with the relevant buttons.

If you select `+ Explore in activity mode +` (second entry under `Activity:`, you will have to select same items as with Explore route, but in this case the game will automatically generate an activity (with the player train only) and will execute it. By exploring the route in this mode you will able to switch to autopilot mode if you like (see *here*) and you will have access to some other activity features like *randomized weather* if selected.

To select the consist you have two possibilities: either you click under `Consist:`, and the whole list of available consists will appear, or you first click under `Locomotive:`, where you can select the desired locomotive, and then click under `Consist:`, where only the consists led by that locomotive will appear.

If you instead select a specific activity, you won't have to perform any further selections.

Activity Evaluation

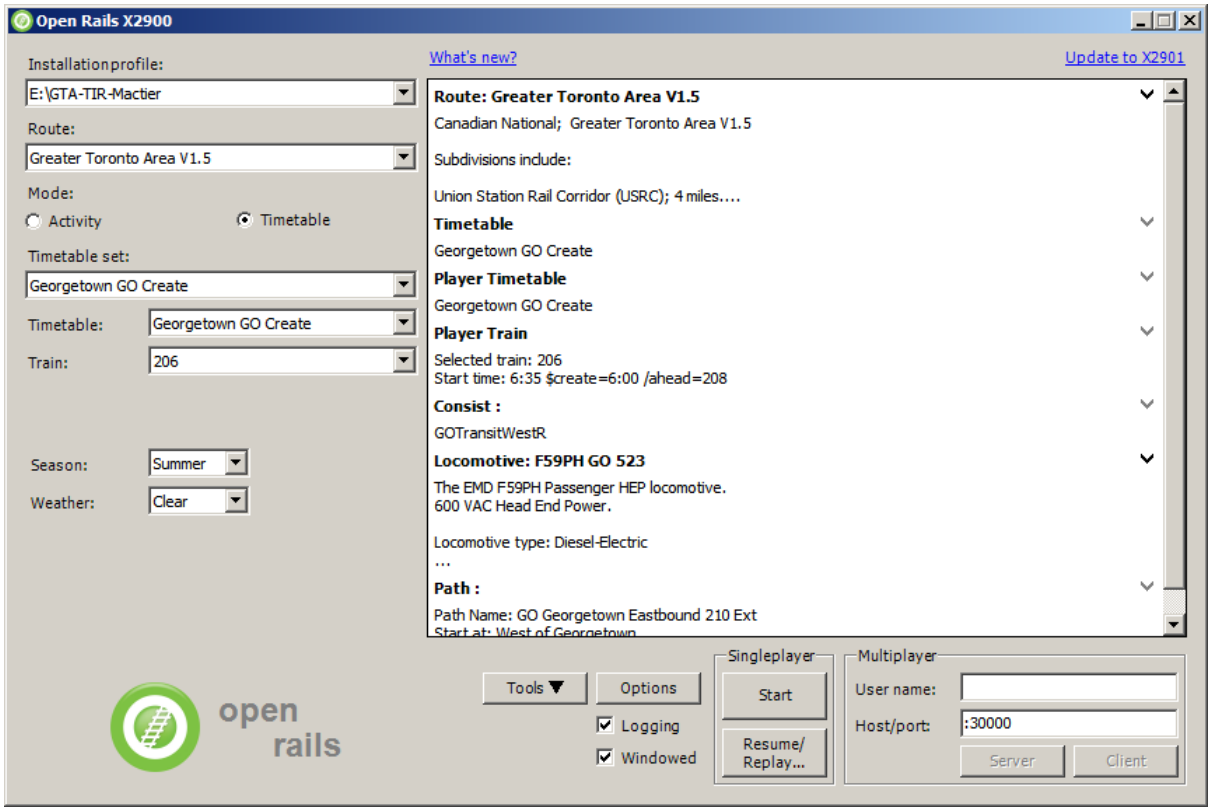
During the activity session, data about performance is stored and may be viewed as the activity progresses. At the end of the activity a report file is generated which provides a summary of the player's skills as a train driver.

Activity evaluation is described *here*.

If you have selected the related Experimental Option, at runtime you can switch *Autopilot mode* on or off, which allows you to watch OR driving your train, as if you were a trainspotter or a visitor in the cab. Autopilot mode is not available in Explore mode.

5.4.2 Timetable Mode

If you select the radio button `Timetable`, the main menu window will change as follows:



Timetable mode is unique to Open Rails, and is based on a `timetable` that is created in a spreadsheet formatted in a predefined way, defining trains and their timetables, their paths, their consists, some operations to be done at the end of the train run, and some train synchronization rules.

Timetable mode significantly reduces development time with respect to activities in cases where no specific shunting or train operation is foreseen. The complete description of the timetable mode can be found *here.*

The spreadsheet has a .csv format, but it must be saved in Unicode format with the extension . `timetable_or` in a subdirectory named `Openrails` that must be created in the route's `ACTIVITIES` directory.

A specific tool (Timetable editor) is available under the "Tools" button to ease generation of timetables.

For the game player, one of the most interesting features of timetable mode is that any one of the trains defined in the timetable can be selected as the player train.

The drop-down window `Timetable set:` allows you to select a timetable file from among those found in the route's `Activities/Openrails/` folder.

Now you can select in the drop-down window `Train:` from all of the trains of the timetable the train you desire to run as the Player train. Season and weather can also be selected.

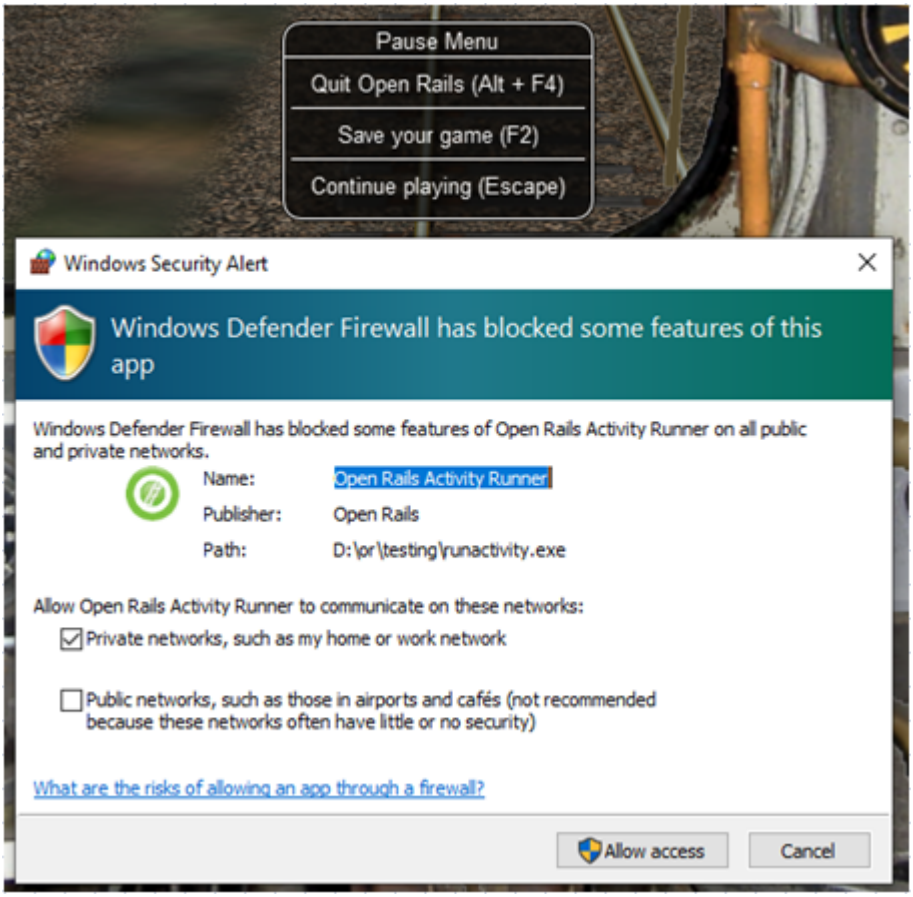
5.4.3 Run

Now, click on `Start`, and OR will start loading the data needed for your game. When loading completes you will be within the cab of your locomotive! You can read further in the chapter *Driving a Train*.

5.5 Firewall

The game uses a built-in web-server to deliver standard and custom web-pages to any browser - see *Web Server*.

When the game runs for the first time, the web-server will try to use a port on your PC to serve any browser that you might want to run. The Windows OS will detect this and pop up a prompt to ask permission for this.



We recommend that you grant permission as a private network even if you don't plan to use a browser straight away.

5.5.1 Multiplayer Mode

Open Rails also features this exciting game mode: several players, each one on a different computer in a local network or through the Internet, can play together, each driving a train and seeing the trains of the other players, even interacting with them by exchanging wagons, under the supervision of a player that acts as dispatcher. The multiplayer mode is described in detail *here.*

5.5.2 Replay

This is not a real gaming mode, but it is nevertheless another way to experience OR. After having run a game you can save it and replay it: OR will save all the commands that you gave, and will automatically execute the commands during replay: it's like you are seeing a video on how you played the game. Replay is described *later* together with the save and resume functions.

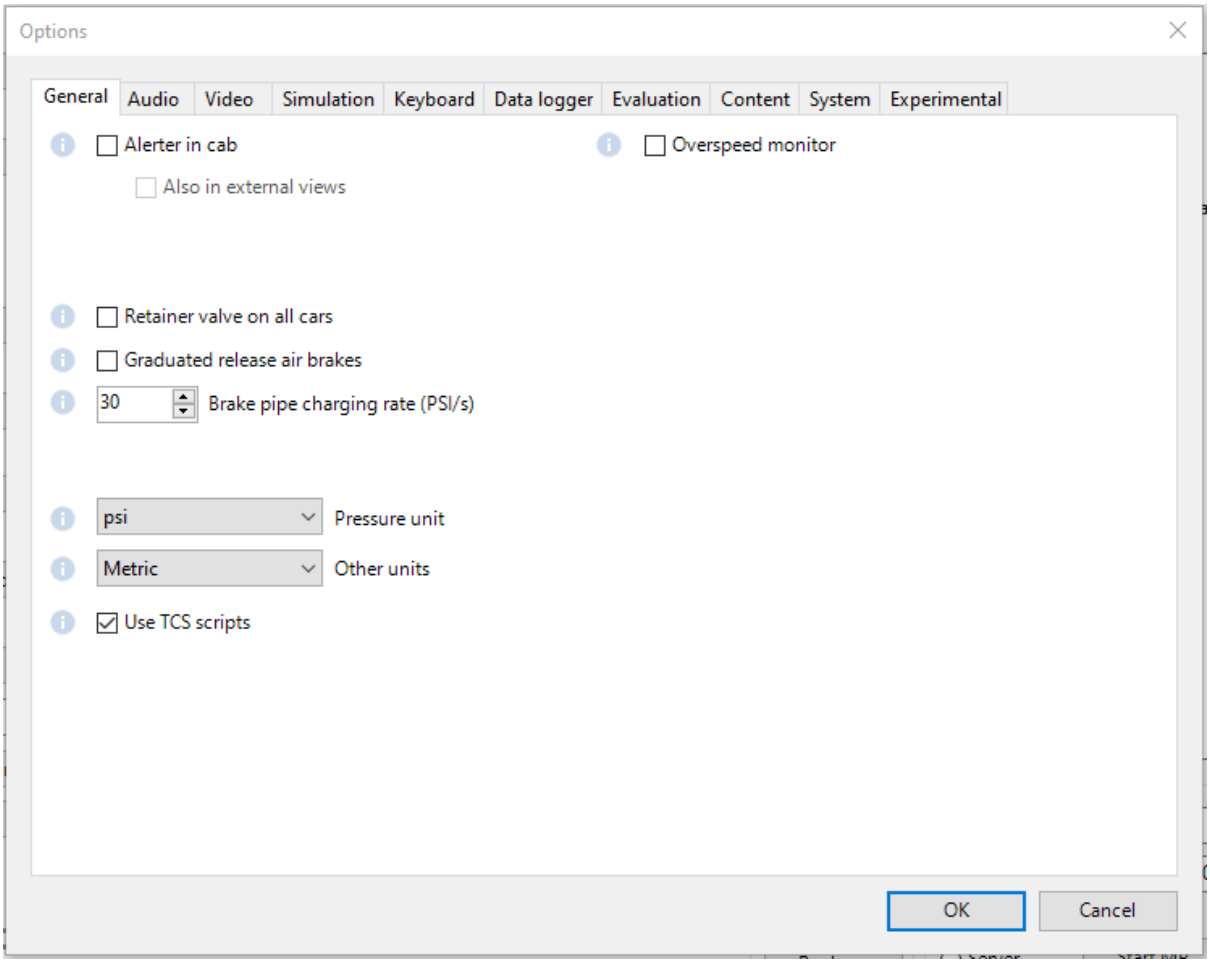
CHAPTER 6

Open Rails Options

Clicking on the *Options* button opens a multi-panel window. The *Menu* > *Options* panels contain the settings which remain in effect during your simulation. Most of the options are self-explanatory; you may set them according to your preference and system configuration. For example, you can turn off dynamic shadowing if your system has low FPS (frames-per-second) capability. The options configuration that you select is saved when you click *OK*. When you restart OR, it will use the last options configuration that you selected.

There are 10 option panels, described below.

6.1 General Options



6.1.1 Alerter in cab

As in real life, when this option is selected, the player driving the train is required to perform specific actions to demonstrate that he is *alive*, i.e. press the Alerter Button (or press the Key <Z>). As the player may sometimes use a view other than the cabview to follow the train, and therefore will not see the alerter warning, selecting the related option *Also in external views* enables the alerter in those views as well.

6.1.2 Graduated release air brakes

Selecting this option allows a partial release of the brakes. Generally speaking, operating with the option checked is equivalent to passenger standard and unchecked is equivalent to freight standard. A complete description of this option can be found *here*.

6.1.3 Retainer valve on all cars

The player can change the braking capability of all of the cars in the simulation to include *Brake Retainers*. These cause the brake cylinder on a car to retain some fixed pressure when the train brakes are released; this causes the car to produce a constant braking force. If this option is not checked, then brake retainers are only found on cars that have an appropriate entry, as described *here*, in their .wag files.

6.1.4 Brake pipe charging rate

The Brake Pipe Charging Rate (psi/s) value controls the charging rate of the main air brake pipe. Increasing the value will reduce the time required to recharge the train (i.e. when releasing the brakes after a brake application), while decreasing the value will slow the charging rate. See also the *paragraphs* on the ORTS implementation of the braking system.

If this parameter is set at 1000, a simplified, MSTS-like braking model is implemented, providing for faster brake release and being less influenced by incoherent braking parameters within .eng file.

6.1.5 Pressure unit

The player can select the unit of measure of brake pressure in the *HUD display*.

When set to *automatic* the unit of measure is the same as that used in the cabview of the locomotive.

6.1.6 Other units

This selects the units displayed for length, mass, pressure, etc. in the *F5 Train Driving Info Window and also the Alt+F5 HUD* of the simulation.

The option *Player's Location* sets the units according to the Windows *Language and Region* settings on the player's computer.

The option *Route* sets the units based on the data in the route files. The other options are self-explanatory.

These windows use the abbreviations *t-us* for short tons (2000 lb), *t-uk* for long tons (2240 lb) and *t* for metric tons (1000 kg).

Note: The units displayed by the *F4 Track Monitor* (e.g. velocity and distance) are always based on data read from the route files.

6.1.7 Use TCS scripts

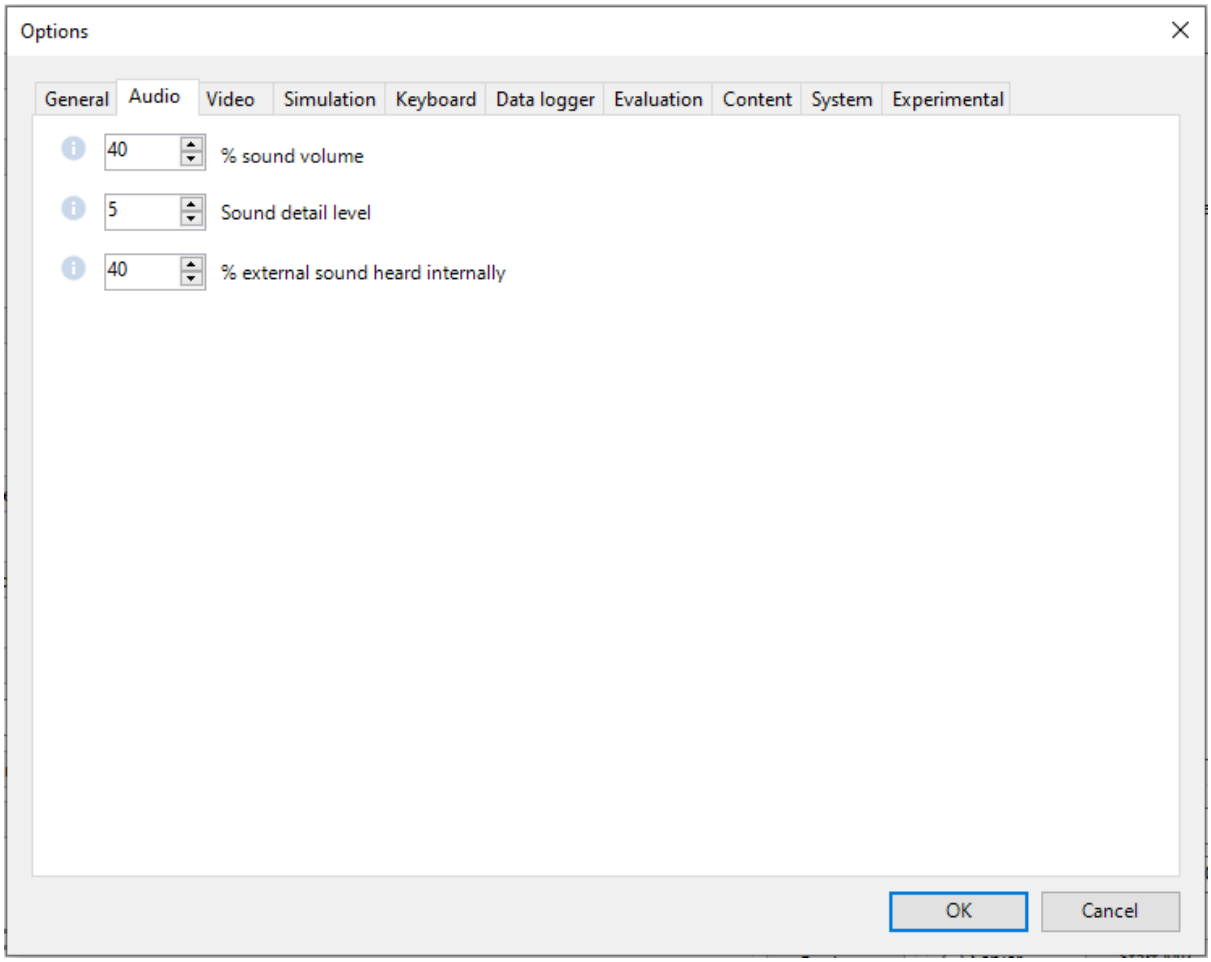
This option uses the train control system scripts for locomotives where these have been implemented.

6.1.8 Overspeed Monitor

If a Train Control Script (TCS) is specified for the loco and not disabled, then that takes priority. Otherwise, if the loco has an Overspeed Monitor specified in its ENG file, then that monitor will detect excessive speed and respond as it was specified, e.g. by applying emergency braking.

This monitor is enabled by checking the option.

6.2 Audio Options



6.2.1 Sound Volume

The *% sound volume* scroll button allows adjustment of the volume of OR sound.

Default is 40.

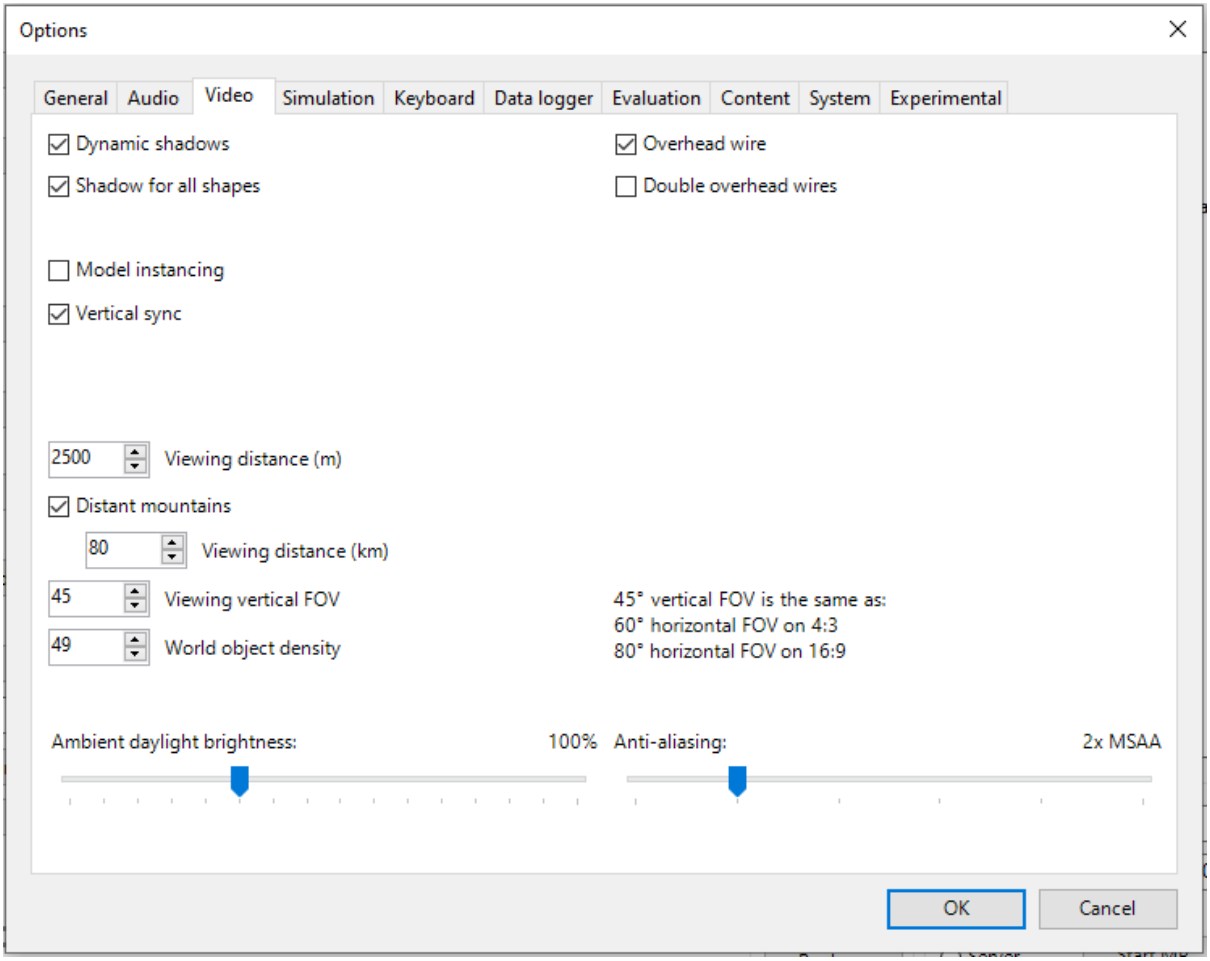
6.2.2 Sound Detail Level

Except for very slow computers, it is suggested that you set the Sound detail level to 5.

6.2.3 External Sound

The *% external sound heard internally* scroll button defines the percentage of the original volume of external sounds heard in cab and passenger views. This percentage may be overridden trainset by trainset as defined *here*.

6.3 Video Options



6.3.1 Dynamic shadows

This option causes movable objects such as trains to cast shadows. Unchecking the option will increase the frame rate.

The default setting is checked.

6.3.2 Shadow for all shapes

Check this option to cast shadows from static objects.

The default setting is unchecked.

Note: This may reduce the frame rate.

Note: Static objects provided with shadows (in the World files) will cast shadows anyway. This option adds shadows for other static objects.

6.3.3 Model instancing

When the option is checked, in cases where multiple instances of the same object have to be drawn, only a single draw call is sent to the GPU. Uncheck this option to avoid the graphical glitches which appear on some hardware, but this may reduce the frame rate.

The default setting is checked.

6.3.4 Overhead wire

This option will enable or disable display of the overhead wire.

6.3.5 Double overhead wires

MSTS uses a single wire for electrified routes; you may check this box so that ORTS will show the two overhead wires that are more common.

6.3.6 Vertical sync

Vertical Sync (VSync) attempts to lock Open Rails' output frame rate to your monitor's refresh rate for the smoothest image and to resist image "tearing".

VSync may help keep the frame rate more stable on complex routes, reducing sudden frame rate drops and apparent control lag in some cases. If Open Rails' frame rate drops below your monitor's frame rate, you may see stuttering or image "tearing". To prevent this, either uncheck the VSync option or reduce the values for video options such as view distance, anti-aliasing, or world object density.

The default setting is checked.

6.3.7 Viewing distance

This option defines the maximum distance at which terrain and objects are displayed. Where the content provides “Distant Mountains”, these are displayed independently (see below).

Note: When the option to tune settings automatically is applied, then this value will be overridden and dynamically changed to maintain a target frame rate.

Note: Some routes are optimized for a viewing distance of 2km as this is the maximum provided by MSTS.

The default distance is 2km.

6.3.8 Distant mountains

This option defines the maximum distance at which “Distant Mountains” are displayed.

Note: “Distant Mountains” are present in the route if it has a folder called LO_TILE.

The default setting is checked.

The default distance is 40km.



6.3.9 Extend object maximum viewing distance to horizon

With this option selected, all objects viewable up to the viewing distance (as defined above) are displayed, even if they have a level of detail (LOD) that is less distant.

Without this option, ORTS only displays objects up to their peak distance set by their level of detail (LOD) or the viewing distance, whichever is less.

Selecting this option shows all the objects that should be in view but it may reduce the frame rate. MSTS limits the viewing distance to just 2km and the peak LOD distances are usually 2km, so this option is especially useful for viewing routes created for MSTS at distances above 2km.

However, for routes that make good use of LOD settings, showing the objects that should be in view can be achieved at higher frame rates by unchecking this option. For example, if the viewing distance is 10km and the content has been created with some large objects having peak distance LODs set at 5km

and smaller objects having LODs which are much shorter, then this strategy will show all the objects that should be in view without reducing the frame rate.

The default setting is checked.

6.3.10 Viewing vertical FOV

This value defines the vertical angle of the world that is shown. Higher values correspond roughly to a zoom out effect. The default is 45 degrees.

6.3.11 World object density

This value can be set from 0 to 99 and the default value is 49. When 49 is selected, all content defined in the route files and intended for the player to see is visible. Lower values will hide some categories of objects which tends to increase frame rates.

In legacy routes, all the content was assigned to categories 0-10. In more modern routes, content may be assigned to categories between 0 and 49. Content builders are advised to reserve values 50 to 99 for objects used in building the route.

6.3.12 Ambient daylight brightness

With this slider you can set the daylight brightness.

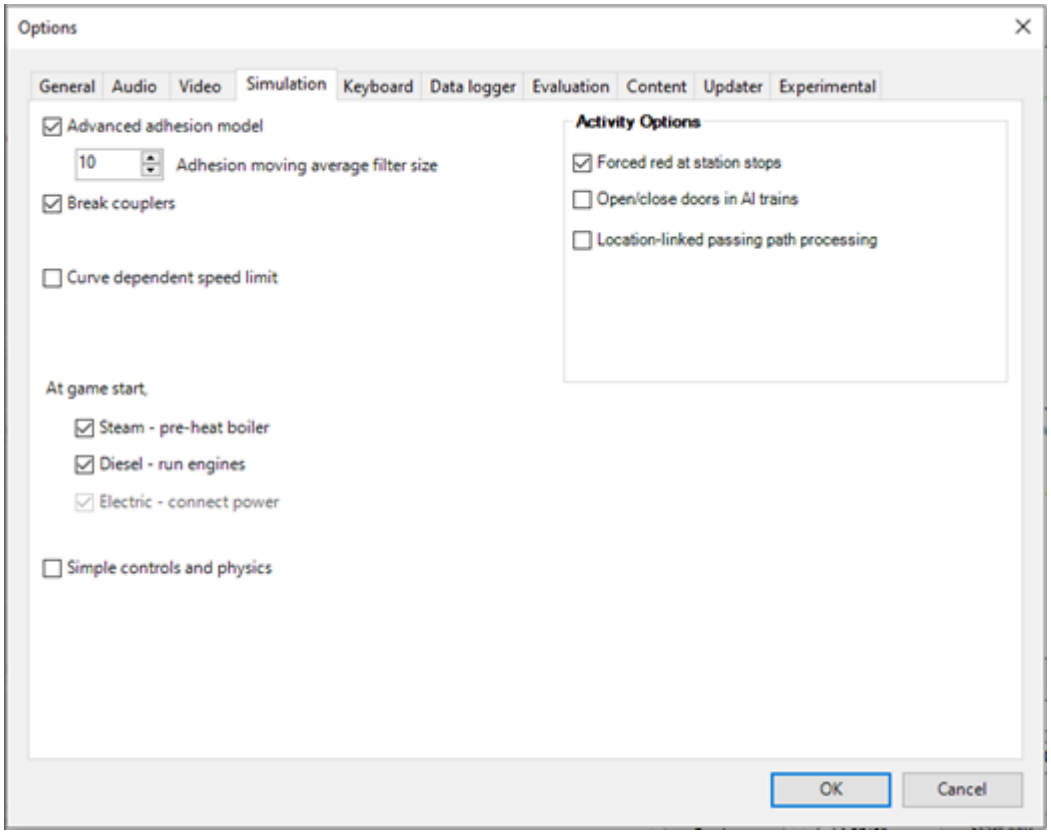
6.3.13 Anti-aliasing

Controls the anti-aliasing method used by Open Rails. Anti-aliasing is a computer graphics technique that smooths any harsh edges, otherwise known as “jaggies,” present in the video image. Currently, Open Rails only supports the multisample anti-aliasing (MSAA) method. Higher applications of anti-aliasing will require exponentially more graphics computing power.

The default setting is MSAA with 2x sampling.

6.4 Simulation Options

The majority of these options define train physics behavior.



6.4.1 Advanced adhesion model

OR supports two adhesion models: the basic one is similar to the one used by MSTS, while the advanced one is based on a model more similar to reality.

For more information read the section on *Adhesion Models* later in this manual.

6.4.2 Adhesion moving average filter size

The computations related to adhesion are passed through a moving average filter. Higher values cause smoother operation, but also less responsiveness. 10 is the default filter size.

6.4.3 Break couplers

When this option is selected, if the force on a coupler is higher than the threshold set in the .eng file, the coupler breaks and the train is divided into two parts. ORTS will display a message to report this.

6.4.4 Curve dependent speed limit

When this option is selected, ORTS computes whether the train is running too fast on curves, and if so, a warning message is logged and displayed on the monitor. Excessive speed may lead to overturn of cars, this is also displayed as a message. This option is described in detail *here* (theory) and also *here* (OR application). OR does not display the damage.

6.4.5 At game start, Steam - pre-heat boiler

With this option selected, the temperature and pressure of steam in the boiler is ready to pull the train. Uncheck this option for a more detailed behaviour in which the player has to raise the boiler pressure.

If not, the boiler pressure will be at 2/3 of maximum, which is only adequate for light work. If your schedule gives you time to raise the pressure close to maximum, then switch from AI Firing to Manual Firing (Ctrl+F) and increase the Blower (N) to 100% to raise a draught. Replenish the coal using R and Shift+R to keep the fire mass close to 100%. Full pressure may be reached in 5 minutes or so.

The default setting is checked.

6.4.6 At game start, Diesel - run engines

When this option is checked, stationary diesel locos start the simulation with their engines running. Uncheck this option for a more detailed behaviour in which the player has to start the loco's engine.

The default setting is checked.

6.4.7 Forced red at station stops

In case a signal is present beyond a station platform and in the same track section (no switches in between), ORTS will set the signal to red until the train has stopped and then hold it as red from that time up to two minutes before starting time. This is useful in organizing train meets and takeovers, however it does not always correspond to reality nor to MSTS operation. So with this option the player can decide which behavior the start signal will have.

This option is checked by default.

Note: Unchecking the option has no effect when in *Timetable mode*.

6.4.8 Open/close doors on AI trains

This option enables door open/close at station stops on AI trains having passenger trainsets with door animation. Doors are opened 4 seconds after train stop and closed 10 seconds before train start. Due to the fact that not all routes have been built with correct indication of the platform side with respect to the track, this option can be individually disabled or enabled on a per-route basis, as explained *here*. With option enabled, doors open and close automatically also when a player train is in *autopilot mode*. The option is active only in activity mode.

6.4.9 Location-linked passing path processing

When this option is NOT selected, ORTS acts similarly to MSTS. That is, if two trains meet whose paths share some track section in a station, but are both provided with passing paths as defined with the MSTS Activity Editor, one of them will run through the passing path, therefore allowing the meet. Passing paths in this case are only available to the trains whose path has passing paths.

When this option is selected, ORTS makes available to all trains the main and the passing path of the player train. Moreover, it takes into account the train length in selecting which path to assign to a train in case of a meet.

For content developers

A more detailed description of this feature can be found under *Location-Linked Passing Path Processing* in the chapter *Open Rails Train Operation*.

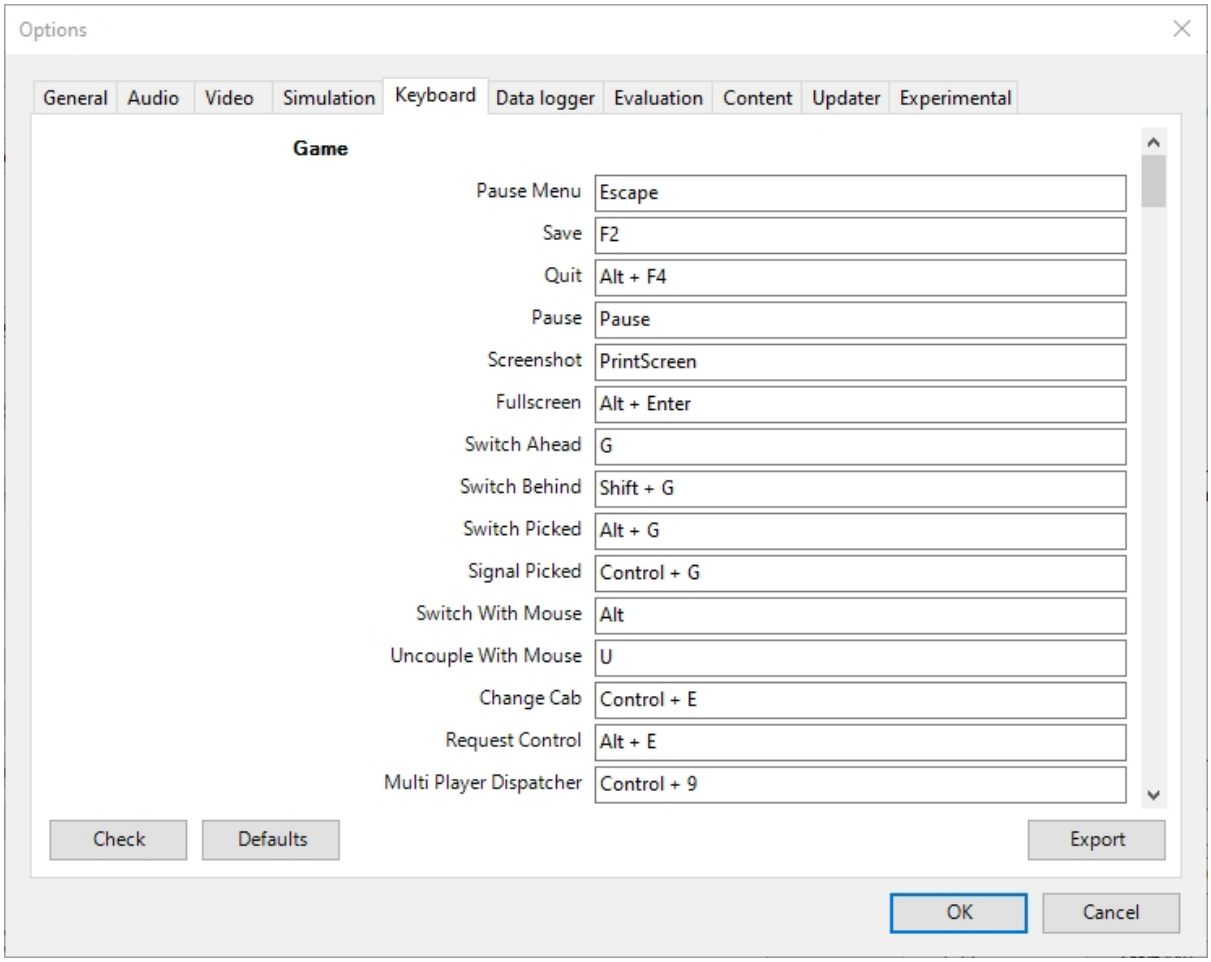
6.4.10 Simple control and physics

This is an option which players can set to simplify either the train controls or physics. This feature is intended for players who want to focus on "running" trains and don't want to be bothered by complex controls or prototypical physics which may require some additional expertise to operate.

Initially this option affects only trains that use vacuum braking but other controls may be added in future versions.

With vacuum braking, it is sometimes necessary to operate two different controls to apply and release the brakes. With "Simple control and physics" checked, the player is able to operate the brakes just with the brake valve and doesn't need to consider the steam ejector separately.

6.5 Keyboard Options



In this panel you will find listed the keyboard keys that are associated with all ORTS commands.

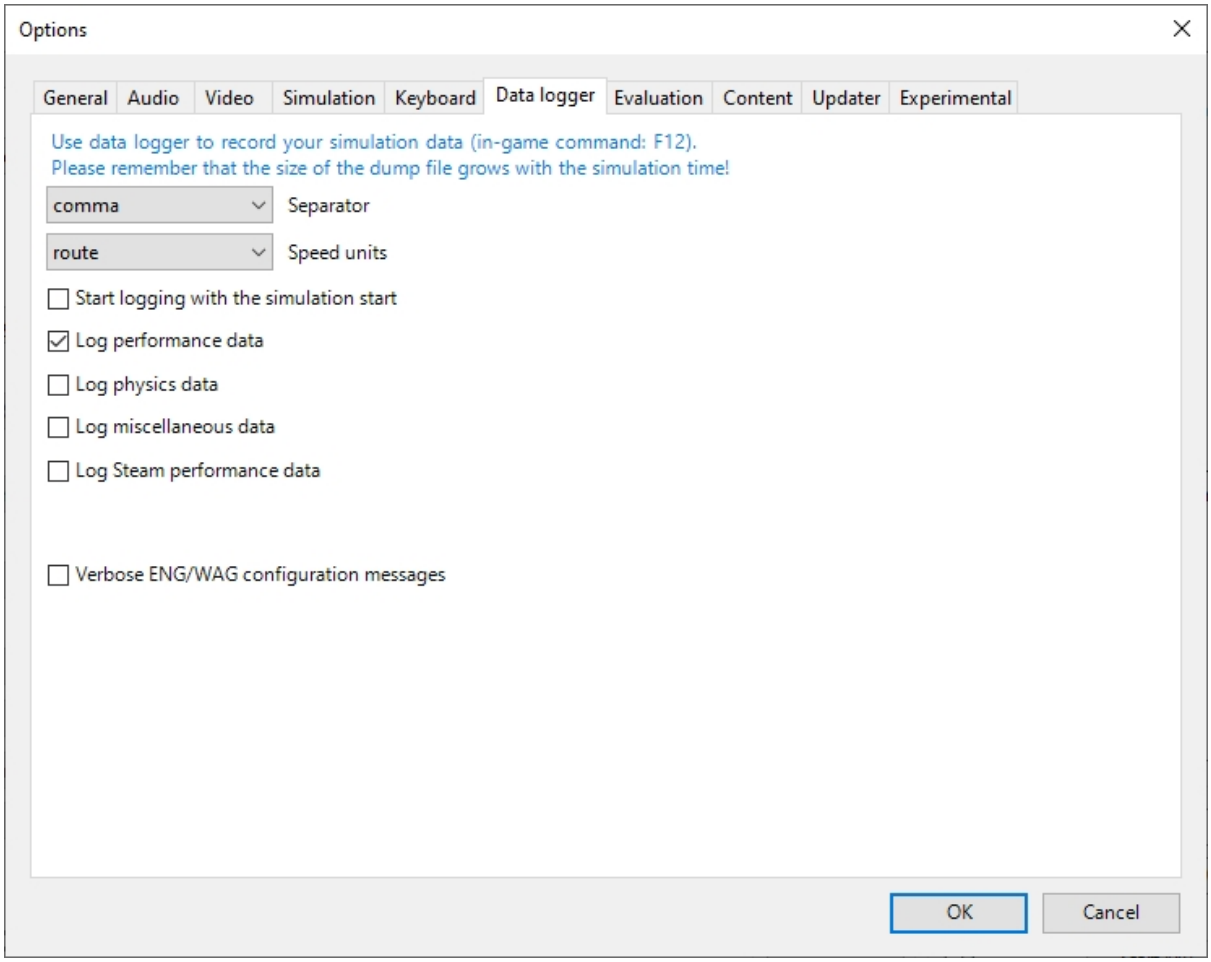
You can modify them by clicking on a field and pressing the new desired key. Three symbols will appear at the right of the field: with the first one you validate the change, with the second one you cancel it, with the third one you return to the default value.

By clicking on *Check* ORTS verifies that the changes made are compatible, that is, that there is no key that is used for more than one command.

By clicking on *Defaults* all changes that were made are reset, and the default values are reloaded.

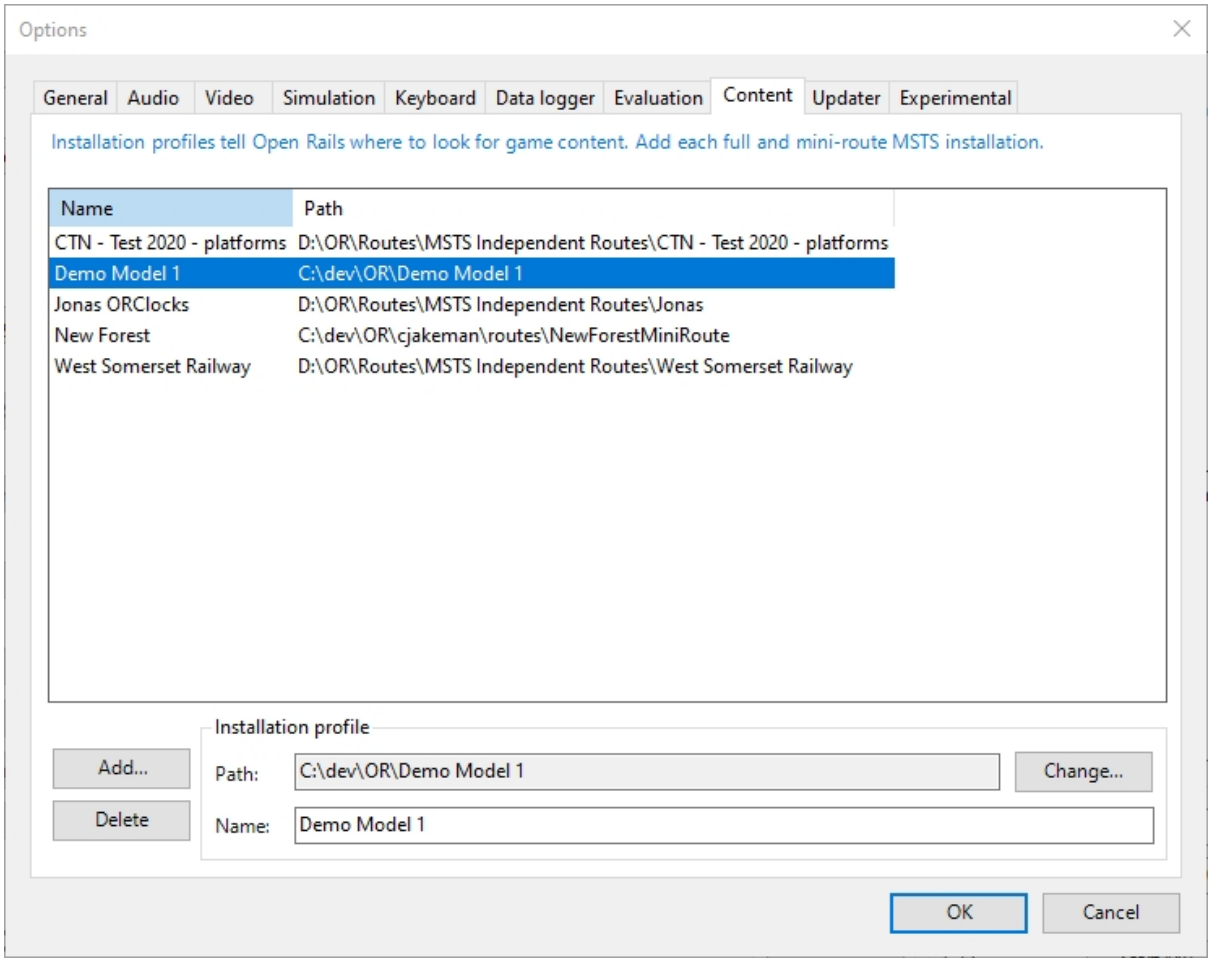
By clicking on *Export* a printable text file `Open Rails Keyboard.txt` is generated on the desktop, showing all links between commands and keys.

6.6 Data Logger Options



By selecting the option *Start logging with the simulation start* or by pressing <F12> a file with the name dump.csv is generated in the configured Open Rails logging folder (placed on the Desktop by default). This file can be used for later analysis.

6.7 Content Options



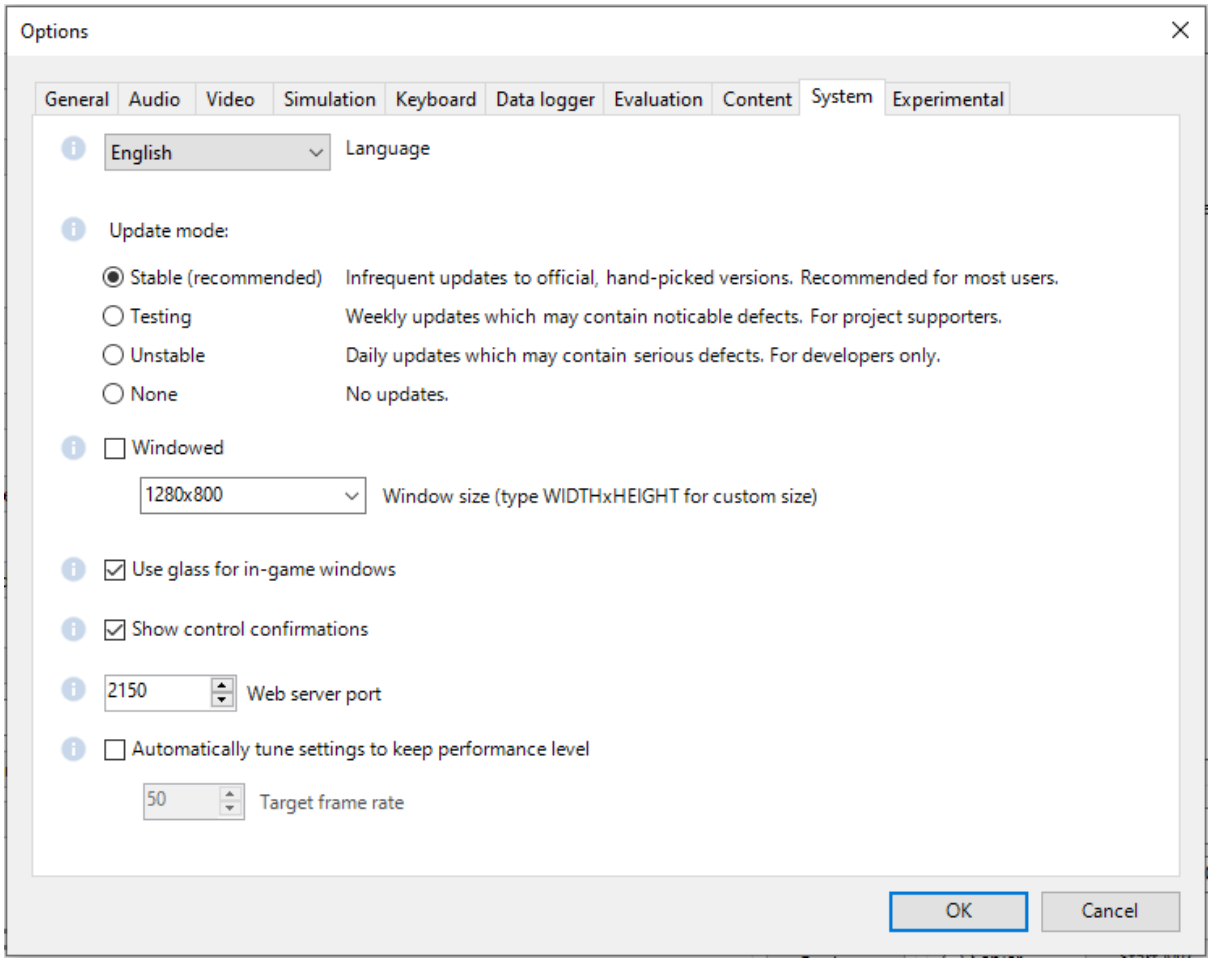
This window allows you to add, remove or modify access to content. Each profile may be a folder containing one or more routes, or an MSTS installation. Profiles located on other drives, or on a USB key, can be added even if they are not always available.

Click on the *Add* button, and locate the desired installation. ORTS will automatically enter a proposed name in the *Name:* window that will appear in the *Installation set:* window on the main menu form. Modify the name if desired, then click *OK* to add the new path and name to Open Rails.

Please do not store content or any files within the folder containing the Open Rails software. This is because the Updater operates by wiping out the contents of that folder before replacing it with a new updated version. It would be unfriendly for users to lose content that they have stored there, so attempts to add a profile folder stored there are blocked and lead to an error message.

To remove an entry (note that this does not remove the installation itself!) select the entry in the window, and click *Delete*, then *OK* to close the window. To modify an entry, use the *Change…* button to access the location and make the necessary changes.

6.8 System Options



6.8.1 Language

ORTS is an internationalized package. It supports many languages, and others can be added by following the instructions contained in the *Localization Manual* which can be found in the Open Rails `Documentation` folder.

When *System* is selected, ORTS automatically selects the language of the hosting OS, if that language is available.

6.8.2 Update mode

These options set which channel is active to update the ORTS version. More details are given *here*.

6.8.3 Windowed

If the `Windowed` checkbox is checked, Open Rails will run in a window instead of full screen.

Once the game has started, you can toggle between windowed mode and full screen by pressing `Alt+Enter`.

The default setting is unchecked.

6.8.4 Window size

This pair of values defines the size of the ORTS window. There are some pre-configured pairs of values and you can also enter a specific width and height to be used.

The format is <width>x<height>, for example 1024x768.

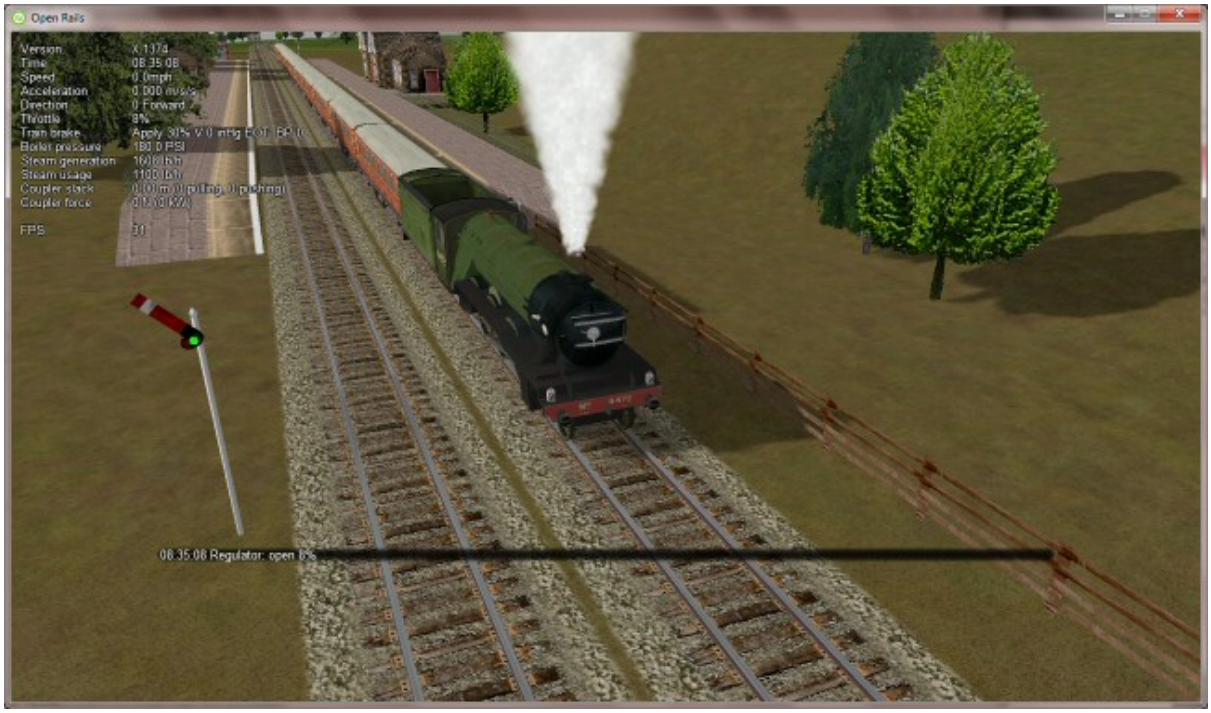
6.8.5 Glass on in-game windows

When this option is checked, the in-game windows are shown semi-transparently.

The default setting is checked.

6.8.6 Control confirmations

Whenever you make adjustments to the train controls (e.g. open the throttle) Open Rails briefly shows a message near the bottom of the screen.



This is helpful for operations that don't have visible feedback and also allows you to control the train without being in the cab.

The default setting is checked. Uncheck this option if you prefer to monitor your cab instruments and don't want to see these messages.

OR uses the same message scheme for system messages such as "Game saved" or "Replay ended" but you cannot suppress these system messages.

Once the game has started, you can toggle the confirmations on and off by pressing `Ctrl+Alt+F10`.

6.8.7 Web server port

The web server can be accessed from a browser on the local machine at `http://localhost:<port>`, where `<port>` is the specified port number. Change the default value of 2150 if it conflicts with other services.

If you open the web server port (just granting RunActivity.exe an exemption is not sufficient) in Windows Firewall, the server can also be accessed from a device on the local network, such as a smartphone, tablet or another PC, using your system's IP address. E.g.: If your Open Rails PC is at IP address 192.168.0.99, browse to `http://192.168.0.99:2150`, where `2150` is the specified port number.

Sample web pages are included in the Open Rails installation and the browser will show a menu of sample pages.

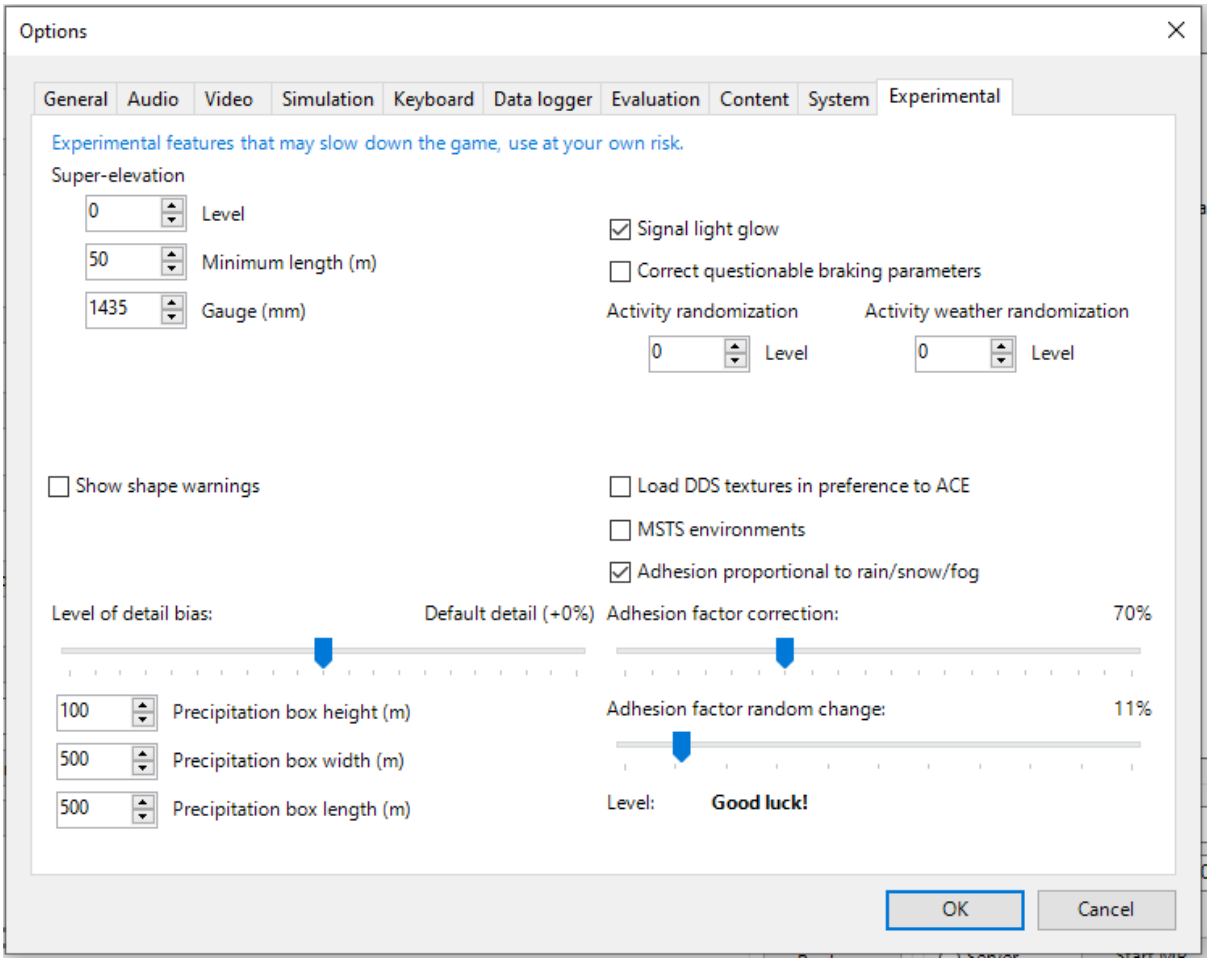
As well as a web browser, data from the web server can also be fetched by any program which can make a web request, such as C# or Python, using the *Application Programming Interface* (API).

6.8.8 Automatically tune settings to keep performance level

When this option is selected ORTS attempts to maintain the selected Target frame rate FPS (Frames per second). To do this it decreases or increases the viewing distance of the standard terrain. If the option is selected, also select the desired FPS in the *Target frame rate* field.

The default setting is unchecked.

6.9 Experimental Options



Some experimental features being introduced in Open Rails may be turned on and off through the *Experimental* tab of the Options window, as described below.

6.9.1 Super-elevation

If the value set for *Level* is greater than zero, ORTS supports super-elevation for long curved tracks. The value *Minimum Length* determines the length of the shortest curve to have super-elevation. You need to choose the correct gauge for your route, otherwise some tracks may not be properly shown.

When super-elevation is selected, two viewing effects occur at runtime:

1. If an external camera view is selected, the tracks and the running train will be shown inclined towards the inside of the curve.
2. When the cab view is selected, the external world will be shown as inclined towards the outside of the curve.





OR implements super-elevated tracks using Dynamic Tracks. You can change the appearance of tracks by creating a `<route folder>/TrackProfiles/ TrProfile.stf` file. The document `How to Provide Track Profiles for Open Rails Dynamic Track.pdf` describing the creation of track profiles can be found in the *Menu* > *Documents* drop-down or the Open Rails `/Source/Documentation/` folder. Forum discussions about track profiles can also be found on Elvas Tower.

6.9.2 Show shape warnings

When this option is selected, when ORTS is loading the shape (.s) files it will report errors in syntax and structure (even if these don't cause runtime errors) in the *Log file* `OpenRailsLog.txt` on the desktop.

6.9.3 Signal light glow

When this option is set, a glowing effect is added to signal semaphores when seen at distance, so that they are visible at a greater distance. There are routes where this effect has already been natively introduced; for these, this option is not recommended.

6.9.4 Correct questionable braking parameters

When this option is selected, Open Rails corrects some braking parameters if they are out of a reasonable range or if they are incoherent. This is due to the fact that many existing .eng files have such issues, that are not a problem for MSTS, which has a much simpler braking model, but that are a problem for OR, which has a more sophisticated braking model. The problem usually is that the train brakes require a long time to release, and in some times do not release at all.

The following checks and corrections are performed if the option is checked (only for single-pipe brake system):

- if the compressor restart pressure is smaller or very near to the max system pressure, the compressor restart pressure and if necessary the max main reservoir pressure are increased;
- if the main reservoir volume is smaller than 0.3 m^3 and the engine mass is higher than 20 tons, the reservoir volume is raised to 0.78 m^3;
- the charging rate of the reservoir is derived from the .eng parameter `AirBrakesAirCompressorPowerRating` (if this generates a value greater than 0.5 psi/s) instead of using a default value.

For a full list of parameters, see *Developing ORTS Content - Parameters and Tokens*

6.9.5 Activity randomization

The related `Level` box may be set to integer values from zero to three. When a level of zero is selected, no randomization is inserted. When a level greater than zero is selected, some activity parameters are randomly changed, therefore causing different behaviors of the activity at every run. Level 1 generates a moderate randomization, level 2 a significant randomization and level 3 a high randomization, that may be unrealistic in some cases. This feature is described in greater detail *here*.

6.9.6 Activity weather randomization

The `Level` box works as the one for activity randomization, and has the same range. When a level greater than zero is selected, the initial weather is randomized, and moreover it changes during activity execution. The randomization is not performed if at activity start the train is within a lat/lon rectangle corresponding to the arid zone of North America (lat from 105 to 120 degrees west and lon from 30 to 45 degrees north). The randomization is not performed either if the activity contains weather change events.

6.9.7 MSTS Environments

By default ORTS uses its own environment files and algorithms, e.g. for night sky and for clouds.

With this option selected, ORTS applies the MSTS environment files. This includes support of Kosmos environments, even if the final effect may be different from the current MSTS one.

6.9.8 Level of detail bias

Many visual objects are modelled at more than one level of detail (LOD) so, when they are seen at a distance, Open Rails can switch to the lesser level of detail without compromising the view. This use of multiple LODs reduces the processing load and so may increase frame rates.

Lowering the LOD Bias setting below 0 reduces the distance at which a lower level of detail comes into view, and so boosts frame rates but there may be some loss of sharpness.

Raising the LOD Bias setting above 0 increases the distance at which a lower level of detail comes into view. This may be useful to sharpen distant content that was created for a smaller screen or a wider field of view than you are currently using.

The default setting is 0.

Note: If your content does not use multiple LODs, then this option will have no effect.

6.9.9 Adhesion proportional to rain/snow/fog

Adhesion is dependent on the intensity of rain and snow and the density of fog. Intensities and density can be modified at runtime by the player.

6.9.10 Adhesion factor correction

The adhesion is multiplied by this percentage factor. Therefore lower values of the slider reduce adhesion and cause more frequent wheel slips and therefore a more difficult, but more challenging driving experience.

6.9.11 Adhesion factor random change

This factor randomizes the adhesion factor corrector by the entered percentage. The higher the value, the higher the adhesion variations.

CHAPTER 7

Driving a Train

7.1 Game Loading

Once you have pressed `Start`, Open Rails loads and processes all the data needed to run the game. During this phase, the route's splash screen is shown with an indicator bar at the bottom.



The first time a session is loaded, an animated bar just shows activity. Subsequent loads of that session show the bar growing across the screen to indicate progress.

If a timetable has been selected, then the game also simulates the progress of the timetable from the first train in the timetable up to start time of the player's train. This is done at high speed, but may still take some time. A second bar appears above the first one to show the progress of this stage.

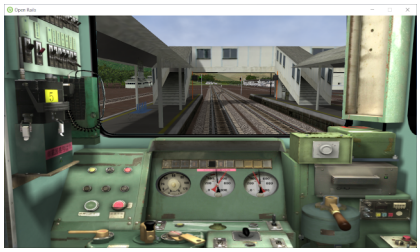
During loading, if logging is selected, the log file `OpenRailsLog.txt` will already begin storing data.

7.2 Entering the Simulation

At the end of the loading phase, you are in the cab of the train you will drive.(Note: some newer locomotives have 3D cabs - if no cab interior display appears, then type `<Alt+1>` to switch between 2D and 3D cabs.) Depending on the configuration of the activity (in case of activity mode), your train will be in motion or stopped. To look around in the simulation, you can select different views using the keyboard, as described in *Changing the View*.

7.2.1 Cab Letter-Boxing

OR manages not only cab interiors using 2D images in a MSTS-compatible way, but also supports 3D models. Most 2D cab images follow MSTS practice, being 1024 x 768 pixels to suit monitors with a 4:3 aspect ratio.



So, the problem arises – how to display these 4:3 cabs on a 16:9 or 16:10 monitor?

One possibility is to enlarge these images to fill the width of wider monitors, as shown in the image below.

In doing so, we lose a portion from the top and bottom of the image. You can use the Up and Down Arrow keys to pan and reveal these missing portions.



Instead of enlarging the image, OR can also 'letterbox' it by showing the full height and filling the missing space with black bars. You can activate this mode in-game by pressing Ctrl+1.



7.3 Open Rails Driving Controls

Open Rails follows MSTS very closely, providing controls to drive steam, electric and diesel locomotives, both on their own or working together, but also offers additional capabilities.

A very wide range of systems and instruments specified in the `ENG` and `CVF` files is supported.

To control the train, you have at your disposal a set of keyboard commands that is equivalent to those of MSTS, plus some new ones. You can get a printable version of the command set as described in paragraph *Keyboard options*, or you can press `<F1>` to immediately get the scrollable F1 Information Window as shown and described *below*.

Alternatively, you can operate the cabview controls by mouse click (buttons) and mouse drag (levers and rotary switches).

7.3.1 Throttle Control

Steam locomotives have a continuous throttle or regulator, but many diesel and electric locomotives have a notched throttle which moves only in steps. To avoid jerks, some of these steps may be *smooth*, where the power is gradually and automatically adjusted to achieve the setting.

7.3.2 Dynamic Braking

Dynamic braking is the use of the traction motors of a locomotive (electric or diesel-electric) as generators to slow the train. Initially, dynamic braking was applied in mountainous territory where conventional freight-car brakes were prone to overheating on long downgrades. It was also limited to speeds above 10mph. Dynamic braking controls are usually notched.

In OR, the dynamic brake (controlled by the keys `<,>` and `<.>`) is not available unless the throttle is fully closed; similarly the throttle is not available unless the dynamic brake is fully released (off).

As defined in the `CVF` file, the tractive and braking forces may be shown on two different instruments, on one instrument with two needles or on a single instrument where the braking is shown as a negative value.

7.3.3 Combined Control

Some locomotives are fitted with a *combined control* where a single lever is used to provide throttle and brake control together, with negative throttle positions used to apply the brake. The brake element may be either dynamic or conventional train brakes.

There may be a delay changing between throttle and brake operation, representing the time required to change the operation of the traction motors from motors to generators.

7.3.4 Blended Dynamic Brake

Some locomotives have blended dynamic brake, which means that the trainbrake lever also controls the dynamic brake. Currently this is implemented to be MSTS compatible, the dynamic brake force percentage follows the train brake pipe pressure (full service/suppression will set 100% dynamic brake). The blending percentage run up/ run down follows the airbrake application `MaxApplicationRate()`, and release rates `MaxReleaseRate()`, and also respects the dynamic brake delay setting `DynamicBrakesDelayTimeBeforeEngaging()` .eng parameters.

Blending can also work if there is no dynamic brake lever configured for the locomotive. If there is dynamic brake lever defined, then the higher command will be applied, except if `OrtsDynamicBlendingOverride( 1 )` is added to the `Engine()` block, which makes the lever override the blending command, if the dynamic brake lever is not at full release position.

`OrtsDynamicBlendingForceMatch( 1 )` parameter can be added to `Engine()` block, which makes the dynamic brake system to try to achieve the same brake force as the airbrake would have (even if the airbrake is bailed off), in the current train brake lever position. Example: if the trainbrake has 22 kN brake force at 40% train brake setting, then the dynamic brake will try to achieve, and maintain 22 kN braking force, instead of just setting 40% dynamic brake percentage.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

7.3.5 Refill

Diesel and steam locomotives must refill their supplies of fuel occasionally, perhaps daily, but steam locomotives need water more frequently and have a range of little more than 100 miles. Use the <T> key to refill with fuel or water at a fuel or water supply location. Use the <Y> key to pick up water from a water trough under a moving locomotive.

If the locomotive or tender is alongside the pickup point, e.g. a water tank, then the refilling takes place as the key is held down. If the locomotive is further away, then the distance to the nearest pickup is shown instead.

Note also that the key <Ctrl+T> will provide immediate refill at any time.

7.3.6 Specific Features to Optimize Locomotive Driving

You are encouraged to read the chapter on *Open Rails Physics* to optimize your driving capabilities and to achieve a realistic feeling of what happens in a real moving train.

7.3.7 Examples of Driving Controls

For content developers

- For continuous throttle, see MSTS model `TRAINS\TRAINSET\ACELA\acela.eng`
- For a notched non-smooth throttle, see `TRAINS\TRAINSET\GP38\gp38.eng`
- For a combined throttle and dynamic brake, see `TRAINS\TRAINSET\DASH9\dash9.eng`
- For a combined throttle and train brake, see `TRAINS\TRAINSET\SERIES7000\series7000.eng`

7.4 Driving aids

Open Rails provides a large number of driving aids, which support the player during train operation.

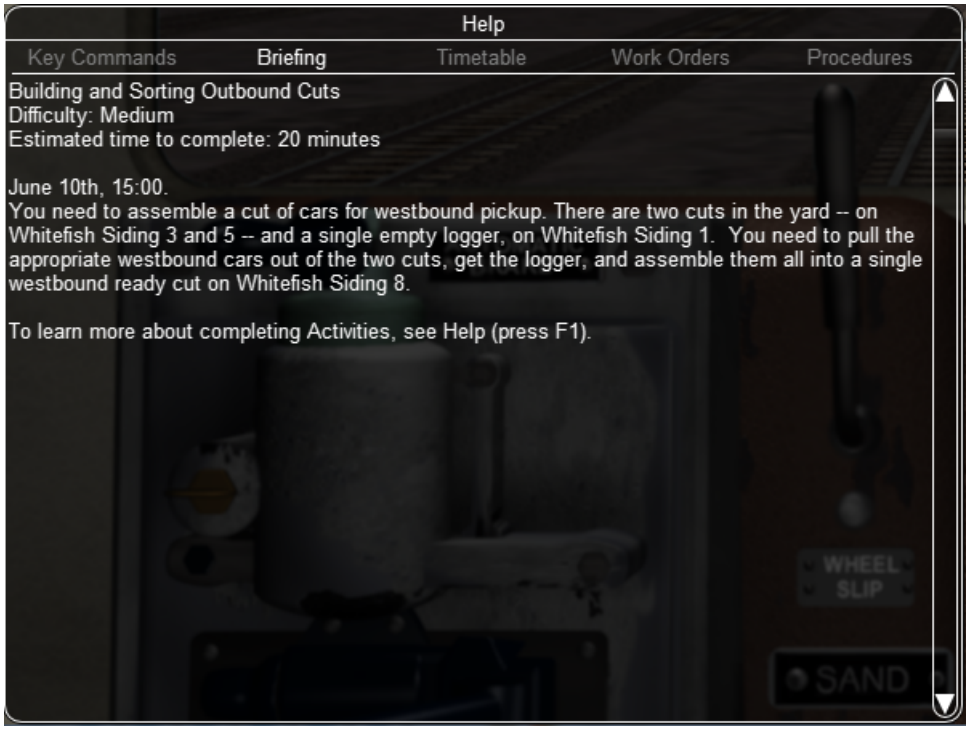
7.4.1 F1 Information Monitor

The F1 key displays the following set of panels in a tabbed format, selected by clicking with the mouse on the desired heading:

`Key Commands`: displays the actions of the keyboard keys



`Briefing`: displays what the activity or timetable creator has entered as information to be provided to the player:



`Timetable`: shows the list of the station stops, if any, with scheduled and actual times of arrival and departure. During the activity the actual performance will be shown on the F10 *Activity Monitor*.

`Work Orders`: if defined by the activity or timetable creator, lists the coupling and uncoupling operations to be performed. When an operation has been completed, the string `Done` appears in the last column:



`Procedures`: basic instructions for driving trains in Open Rails.

7.4.2 F3

This key is not currently used.

7.4.3 F4 Track Monitor

This window, which is displayed by pressing F4, has two different layouts according the the train's *control mode*: `Auto Signal` mode, `Manual` mode or `Explorer` mode. (It is strongly suggested to follow the link and read the related paragraph.)

`Auto Signal` or `Auto` mode is the default mode when running activities or timetables.

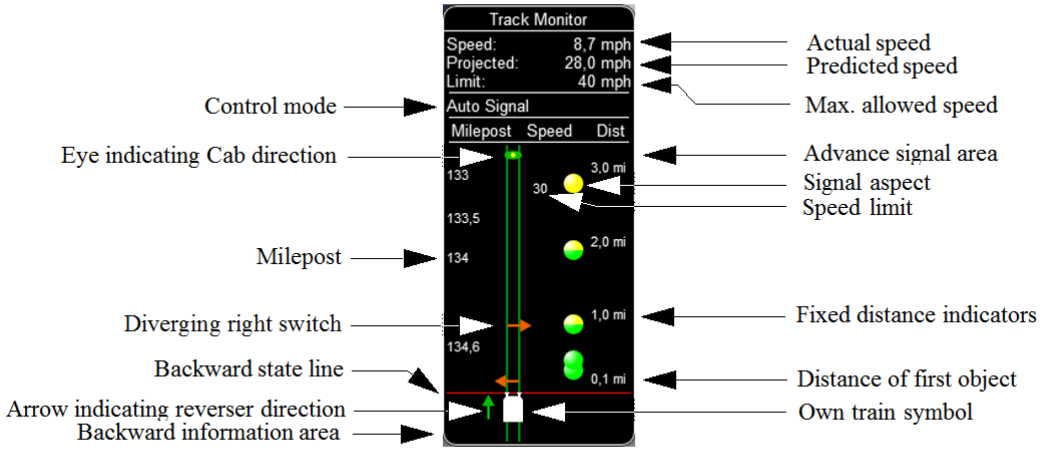
There are however two main cases where you must switch to `Manual` mode by pressing `<Ctrl+M>`:

- when the activity requires shunting without a predefined path
- when the train runs out of control due to SPAD (*Signal Passed At Danger* or passing a red signal) or exits the predefined path by error. If such situations occur you will usually get an emergency stop. To reset the emergency stop and then move to correct the error, you must first switch to `Manual` mode.

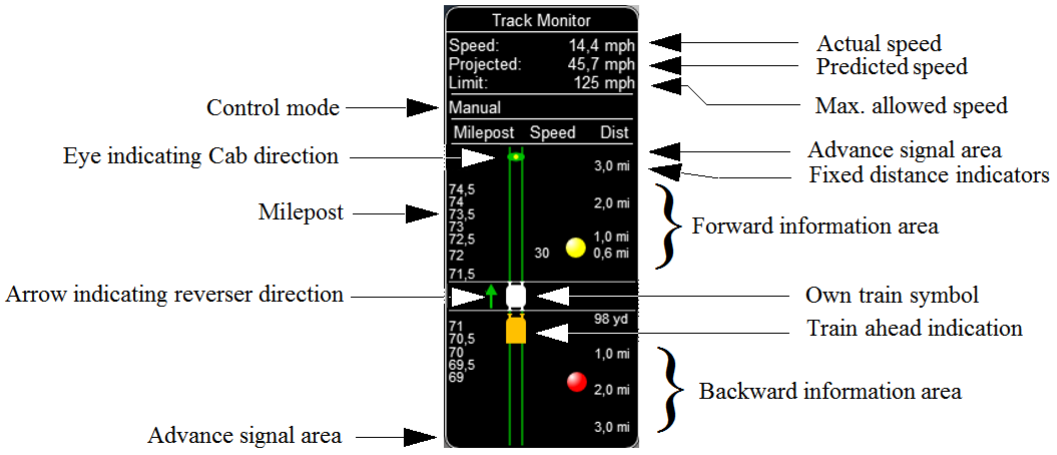
To switch to manual mode press `<Ctrl+M>`. In timetable mode you must first stop the train to pass to manual mode.

You can return to auto mode by pressing `<Ctrl+M>` again when the head of the train is again on the correct path, with no SPAD situation. In standard situations you can also return to auto mode while the train is moving. Details are described in the paragraph of the link shown above.

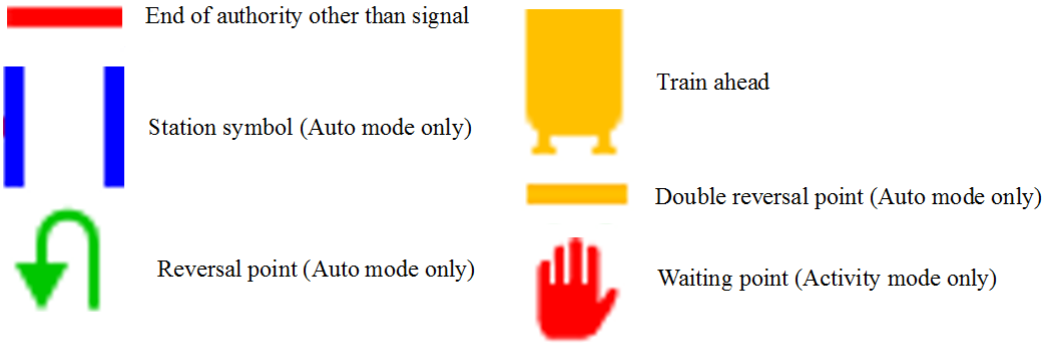
Track Monitor display in Auto Signal mode:



Track Monitor display in Manual mode / Explorer mode:



Track Monitor: Displayed Symbols (common for Auto and Manual mode unless indicated otherwise) :



Notes on the Track Monitor:

- Distance value is displayed for first object only, and only when within distance of the first fixed marker. Distance is not shown for next station stop.

- When no signal is within the normal display distance but a signal is found at a further distance, the signal aspect is displayed in the advance signal area. The distance to this signal is also shown. This only applies to signals, not to speedposts.
- For Auto mode:
  - If the train is moving forward, the line separating the Backward information area is shown in red, and no Backward information is shown.
  - If the train is moving backward, the separation line is shown in white, and Backward information is shown if available.
  - For reversal points, see *here*.
- For Manual mode:
  - If the train is on its defined path (and toggling back to Auto control is possible), the own train symbol is shown in white, otherwise it is shown in red.
- The colour of the track-lines is an indication of the train's speed compared to the maximum allowed speed:
  - Dark green: low speed, well below allowed maximum
  - Light green: optimal speed, just below maximum
  - Orange: slight overspeed but within safety margin
  - Dark red: serious overspeed, danger of derailment or crashing

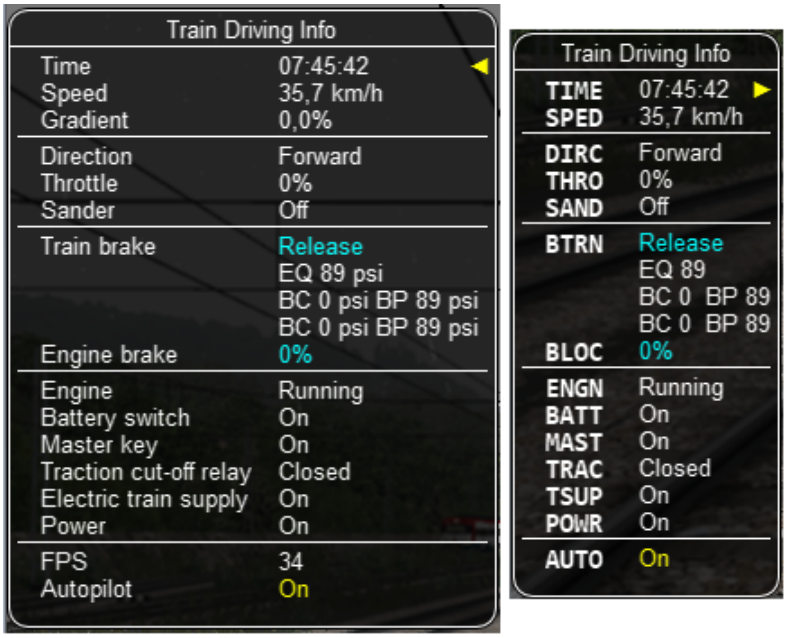
Note that the placement of the display objects with respect to the distance offset is indicative only. If multiple objects are placed at short intermediate distances, the offset in the display is increased such that the texts do not overlap. As a result, only the first object is always shown at the correct position, all other objects are as close to their position as allowed by other objects closer to the train.

Pressing <Shift+F4> toggles the Track Monitor's *immersive mode*. In this mode, the window conceals upcoming signal aspects and upcoming signal speed limits and does not display upcoming diverging switches. However, it retains the locations of signals, mileposts, permanent speed limits, sidings, and stations. This level of assistance reflects the route knowledge that a train driver could be expected to know by memory.

7.4.4 F5 Train Driving Info

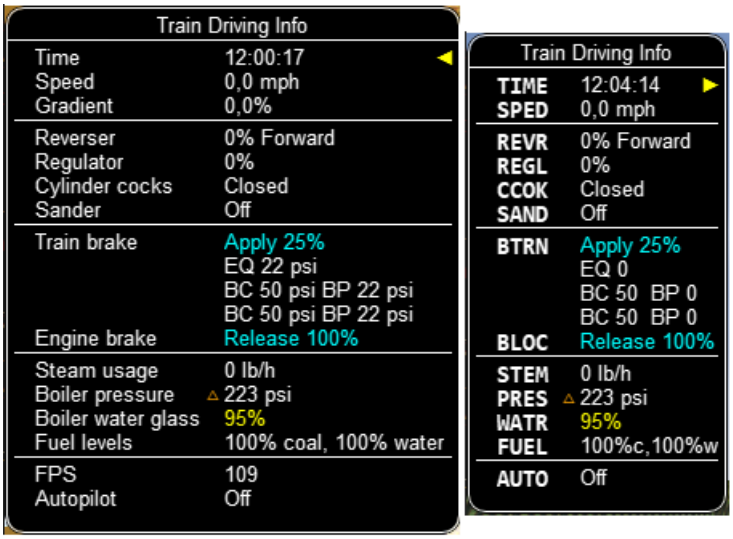
By pressing <F5> you get some important data displayed in a dedicated window.

Pressing <Shift+F5> toggles between the full and the abbreviated text mode. You may also switch mode by clicking on the yellow arrow symbol. The default mode is full text.

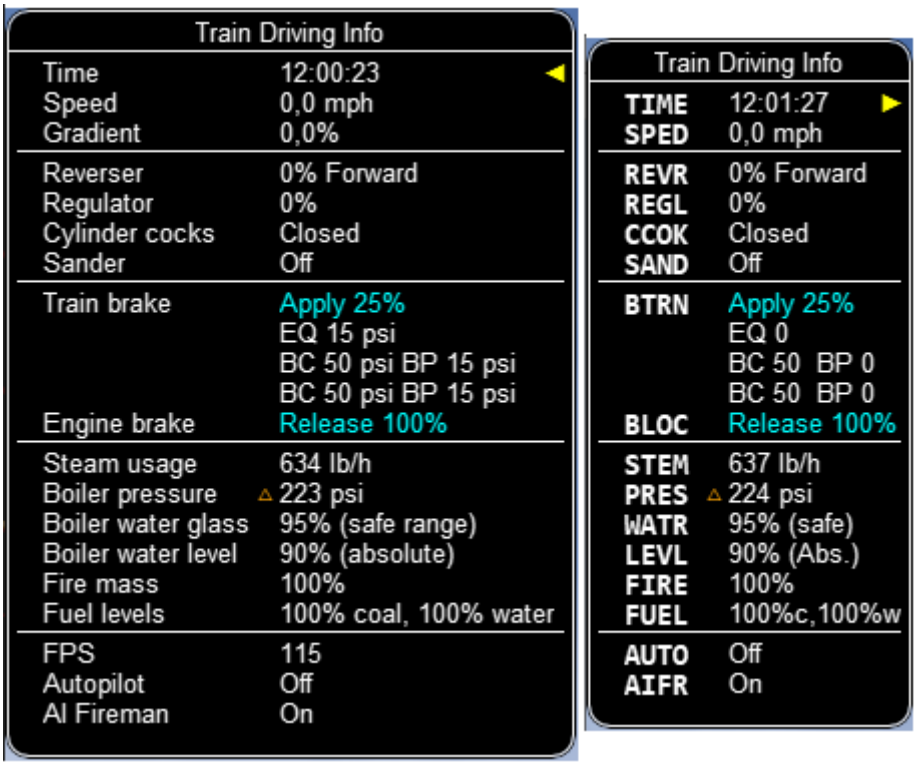


The window displays data appropriate to each type of locomotive as follows.

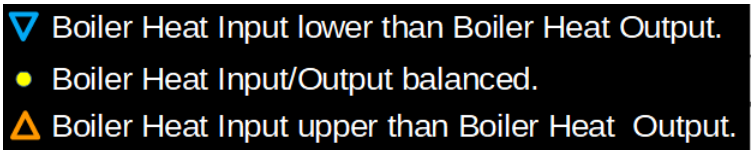
The Steam locomotive:



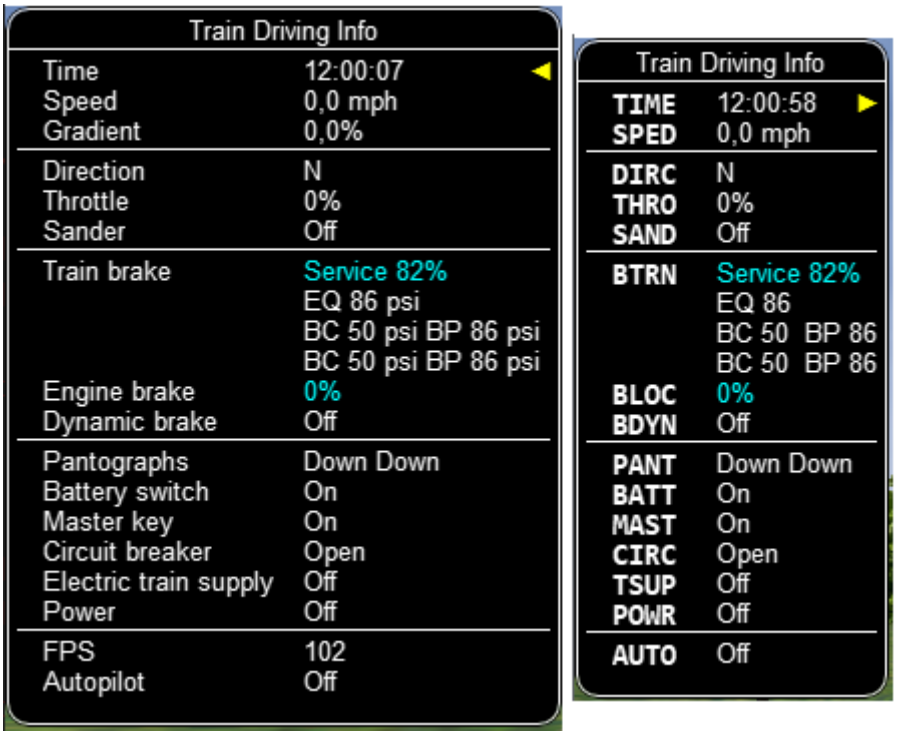
The Steam locomotive using manual firing and AI Fireman:



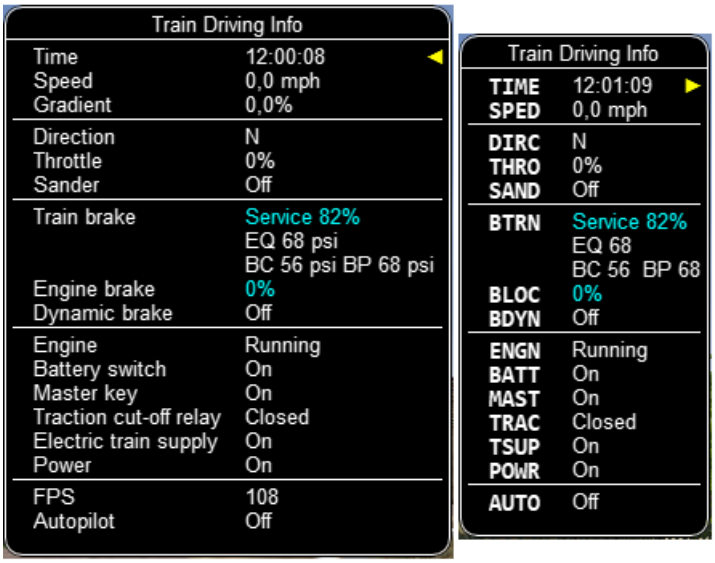
Boiler pressure indicator:



The Electric locomotive:



The Diesel locomotive:



To help the user with ageing eyesight, the Time value is clickable as indicated by the white arrow below. This action toggles between Regular and Bold font styles. The style of font used in this window is also applied to the Multiplayer Info window if it's open.

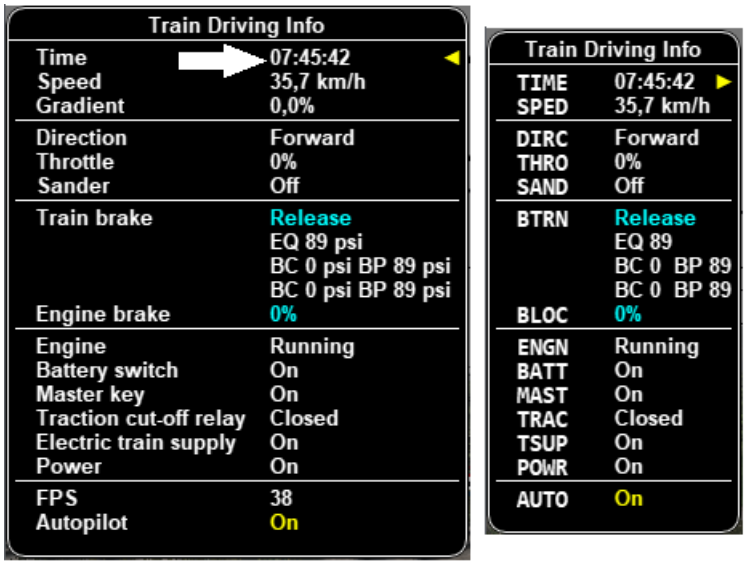


Table 1: ABBREVIATIONS TABLE

Ordered by:		Locomotive	Ordered by:	
Field names	Abbreviations	Type	Abbreviations	Field names
AI Fireman	AIFR	Steam	AIFR	AI Fireman
Autopilot	AUTO	All	AUTO	Autopilot
Battery switch	BATT	Diesel and Electric	BATT	Battery switch
Boiler pressure	PRES	Steam	BDYN	Dynamic brake
Boiler water glass	WATR	Steam	BLOC	Engine brake
Boiler water level	LEVL	Steam	BTRN	Train brake
Circuit breaker	CIRC	Electric	CCOK	Cylinder cocks
Cylinder cocks	CCOK	Steam	CIRC	Circuit breaker
DerailCoeff	DRLC	All	DIRC	Direction
Direction	DIRC	All	DOOR	Doors open
Doors open	DOOR	All	DRLC	DerailCoeff
Dynamic brake	BDYN	Diesel and Electric	ENGN	Engine
Electric train supply	TSUP	Diesel and Electric	FIRE	Fire mass

continues on next page

Table 1 – continued from previous page

Ordered by:		Locomotive	Ordered by:	
Field names	Abbreviations	Type	Abbreviations	Field names
Engine	ENGN	Diesel	FPS	FPS
Engine brake	BLOC	All	FUEL	Fuel levels
Fire mass	FIRE	Steam	GEAR	Fixed gear
Fixed gear	GEAR	Steam	GEAR	Gear
FPS	FPS	All	GRAD	Gradient
Fuel levels	FUEL	Steam	GRAT	Grate limit
Gear	GEAR	Diesel	LEVL	Boiler water level
Gradient	GRAD	All	MAST	Master key
Grate limit	GRAT	Steam	PANT	Pantographs
Master key	MAST	Diesel and Electric	POWR	Power
Pantographs	PANT	Electric	PRES	Boiler pressure
Power	POWR	Electric	REGL	Regulator
Regulator	REGL	Steam	RETN	Retainers
Replay	RPLY	All	REVR	Reverser
Retainers	RETN	If set on all cars	RPLY	Replay
Reverser	REVR	Steam	SAND	Sander
Sander	SAND	All	SPED	Speed
Speed	SPED	All	STEM	Steam usage
Steam usage	STEM	Steam	THRO	Throttle
Throttle	THRO	Diesel and Electric	TIME	Time
Time	TIME	All	TRAC	Traction cut-off relay
Traction cut-off relay	TRAC	Diesel	TSUP	Electric train supply
Train brake	BTRN	All	WATR	Boiler water glass
Wheel	WHEL	All	WHEL	Wheel

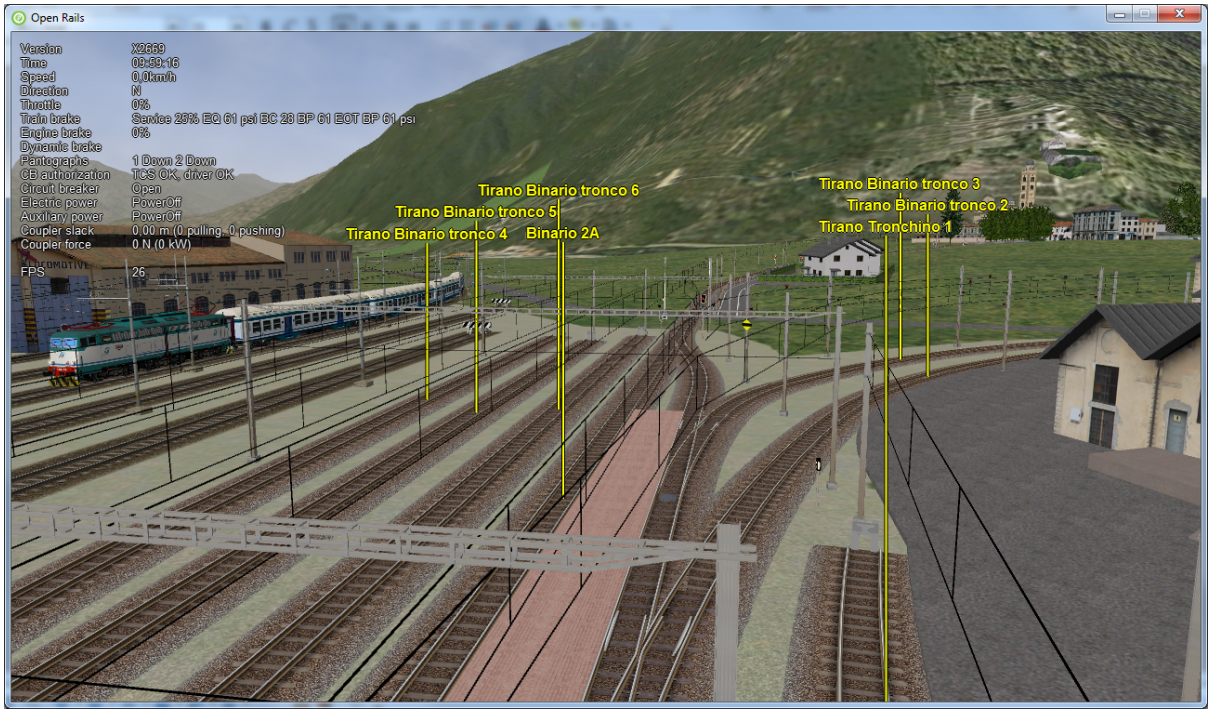
7.4.5 F6 Siding and Platform Names

Hit the <F6> key to reveal labels naming the siding and platforms. Hit it again to hide them.

Items more distant will show more faded and platforms disappear altogether if more than 1km away from the user; sidings disappear if more than 0.5km away.

Use <Shift+F6> to cycle through platforms only (in yellow), sidings only (in orange), and both together.

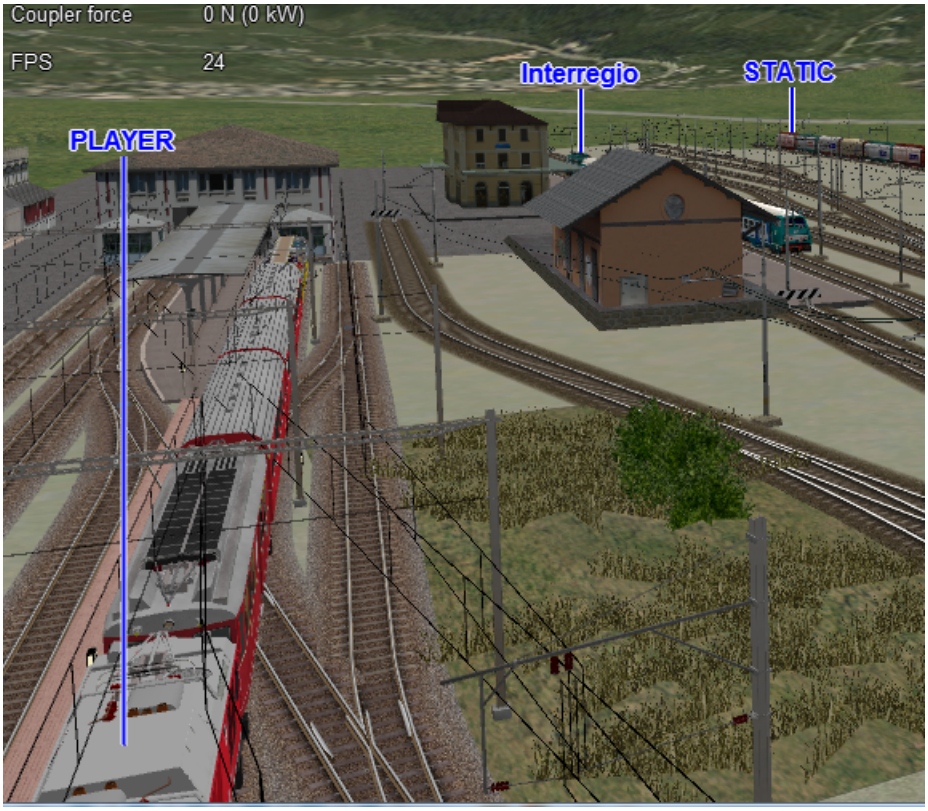
If the user is in Activity Mode or Timetable Mode, then a 4th step is added to the cycle and this step removes any labels not relevant to the activity or timetable.

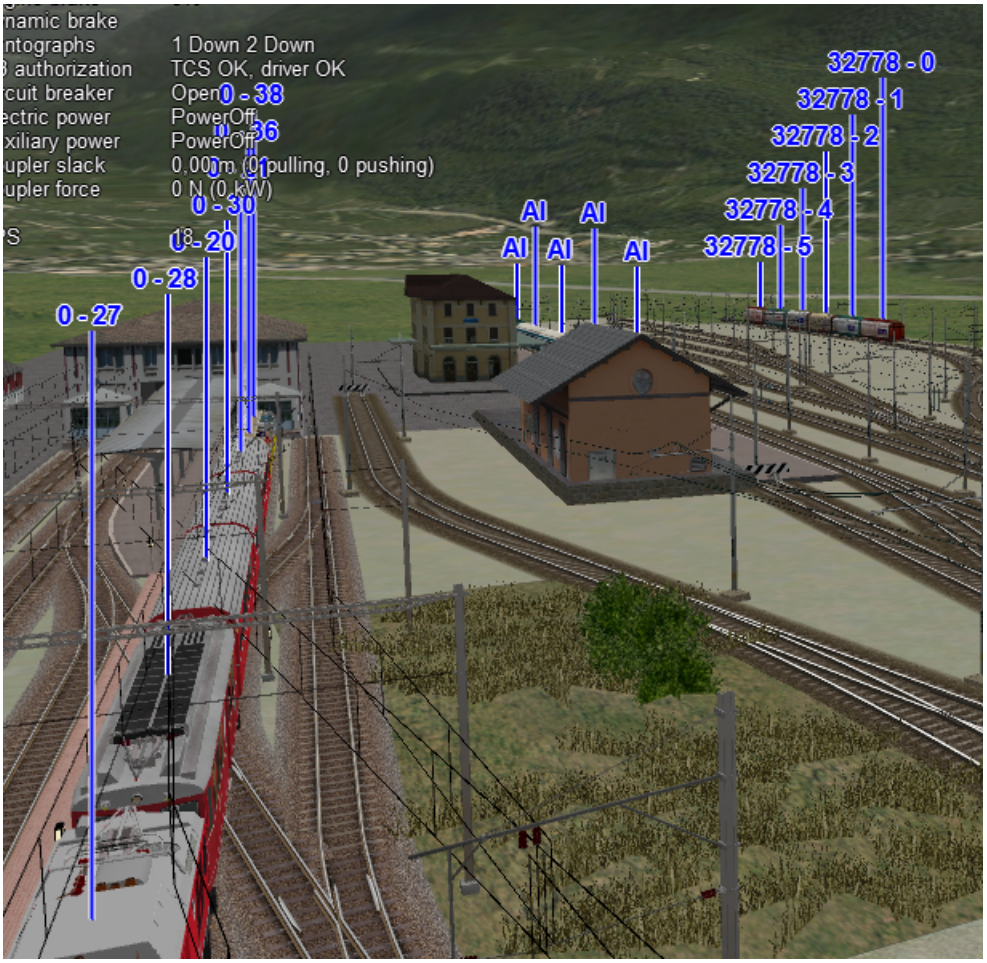


7.4.6 F7 Train Names

Hitting the <F7> key displays train service names (player train always has `Player` as identification).

Hitting <Shift+F7> displays the rolling stock IDs.



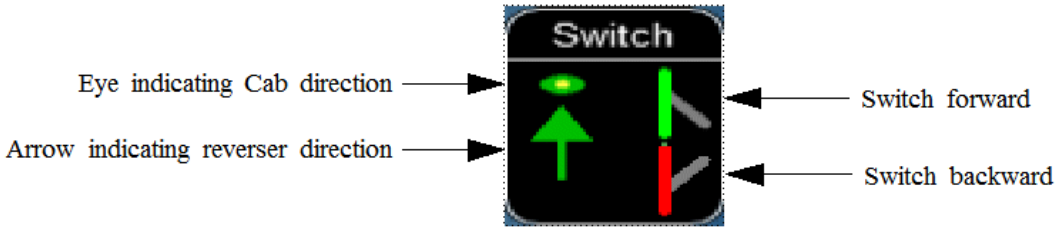


In a multiplayer session, player-controlled trains will have the id specified by the player:



7.4.7 F8 Switch Monitor

Use the Switch Monitor, enabled by the F8 key, to see the direction of the turnout directly in front and behind the train.



There are 4 ways to change the direction:

- Click on the turnout icon in the Switch Monitor;
- Press the G key (or, for the turnout behind the train, the <Shift+G> key);
- Hold down the Alt key and use the left mouse button to click on the switch in the Main Window;
- Use the *dispatcher window*.

Please note that with the last two methods you can throw any switch, not only the one in front but also the one behind the train.

However, note also that not all switches can be thrown: in some cases the built-in AI dispatcher holds the switch in a state to allow trains (especially AI trains) to follow their predefined path.

The arrow and eye symbols have the same meaning as in the track monitor. The switch is red when it is reserved or occupied by the train, and green when it is free.

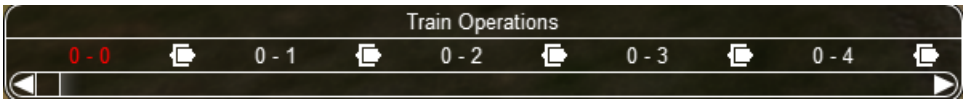
A switch shown in green can be operated, a switch shown in red is locked.

7.4.8 F9 Train Operations Monitor

The Open Rails Train Operations window is similar in function to the F9 window in MSTS, but includes additional features to control the air brake connections of individual cars. For example, it is possible to control the connection of the air brake hoses between individual cars, to uncouple cars without losing the air pressure in the train's air brake hose, or uncouple cars with their air brakes released so that they will coast.

The unit which the player has selected as the unit from which to control the train, i.e. the lead unit, is shown in red.

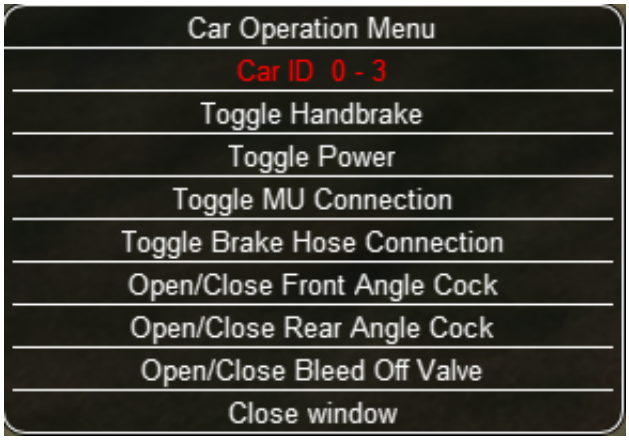
Cars are numbered according to their UiD in the Consist file (.con) or UiD in the Activity file (.act). Scrolling is accomplished by clicking on the arrows at the left or right bottom corners of the window.



Clicking on the coupler icon between any two cars uncouples the consist at that point.

You can also uncouple cars from your player train by pressing the <U> key and clicking with the mouse on the couplers in the main window.

By clicking on any car in the above window, the Car Operation Menu appears. By clicking in this menu it is possible:



- to apply and release the handbrake of the car;
- to power on or power off the car (if it is a locomotive). This applies for both electric and diesel locomotives;
- to connect or disconnect locomotive operation with that of the player locomotive;
- to connect or disconnect the car's air hoses from the rest of the consist;
- to toggle the angle cocks on the air hoses at either end of the car between open and closed;
- to toggle the bleed valve on the car to vent the air pressure from the car's reservoir and release the air brakes to move the car without brakes (e.g. humping, etc.).

By toggling the angle cocks on individual cars it is possible to close selected angle cocks of the air hoses so that when the cars are uncoupled, the air pressure in the remaining consist (and optionally in the uncoupled consist) is maintained. The remaining consist will then not go into `Emergency` state.

When working with cars in a switch yard, cars can be coupled, moved and uncoupled without connecting them to the train's air braking system (see the *Alt+F5 HUD for Braking*). Braking must then be provided by the locomotive's independent brakes. A car or group of cars can be uncoupled with air brakes active so that they can be recoupled after a short time without recharging the entire brake line (`Bottling the Air`). To do this, close the angle cocks on both ends of the car or group before uncoupling. Cars uncoupled while the consist is moving, that have had their air pressure reduced to zero before uncoupling, will coast freely.

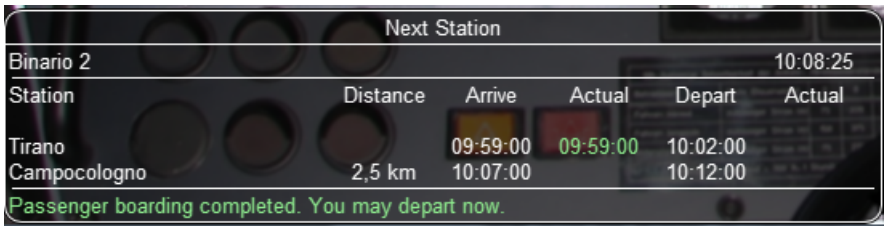
In Open Rails, opening the bleed valve on a car or group of cars performs two functions: it vents the air pressure from the brake system of the selected cars, and also bypasses the air system around the cars if they are not at the end of the consist so that the rest of the consist remains connected to the main system. In real systems the bypass action is performed by a separate valve in each car. In the *Alt+F5 HUD for Braking* display, the text `Bleed` appears on the car's display line until the air pressure has fallen to zero.

More information about manipulating the brakes during coupling and uncoupling can also be found *here*.

7.4.9 F10 Activity Monitor

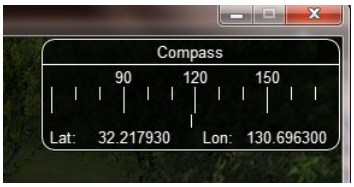
The Activity Monitor is similar in function to MSTS. It records the required `Arrival` time of your train and the actual arrival time as well as the required `Depart` time and the actual departure time.

A text message alerts the engineer as to the proper departure time along with a whistle or other departure sound.



7.4.10 Compass Window

Open Rails software displays a compass that provides a heading based on the camera's direction together with its latitude and longitude.



To activate the compass window press the <0> key. To deactivate the compass window, press the <0> key a second time.

7.4.11 Odometer

The odometer display appears in the centre of the main window, toggled on or off by the keys <Shift+Z>. The direction of the count is toggled by the keys <Shift+Ctrl+Z>, and the odometer is reset or initialized by <Ctrl+Z>.

When set for counting down, it initializes to the total length of the train. As the train moves, the odometer counts down, reaching zero when the train has moved its length. When set for counting up, it resets to zero, and measures the train's total movement.

For example, if the odometer is set for counting down and you click Ctrl+Z as the front of the train passes a location, then when it reaches zero you will know, without switching views, that the other end of the train has just reached the same point, e.g. the entrance to a siding, etc.

The odometer can be accessed also through cabview controls, if they are defined within the cabview, see *here*.

7.4.12 Activity Evaluation

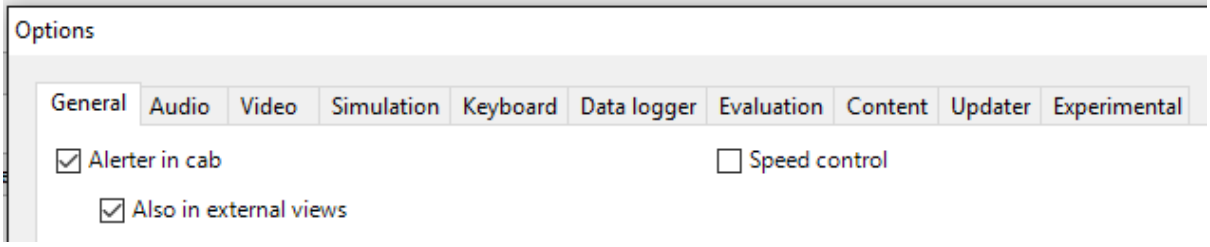
Description

This feature displays a real-time evaluation of the player's performance during the activity run and a final report at the end of an activity. The evaluation reports various parameters to provide to the player info in order to improve his train driving ability. While the activity is running, relevant data are stored and displayed. The stored data is used to generate a report at the end of the activity.

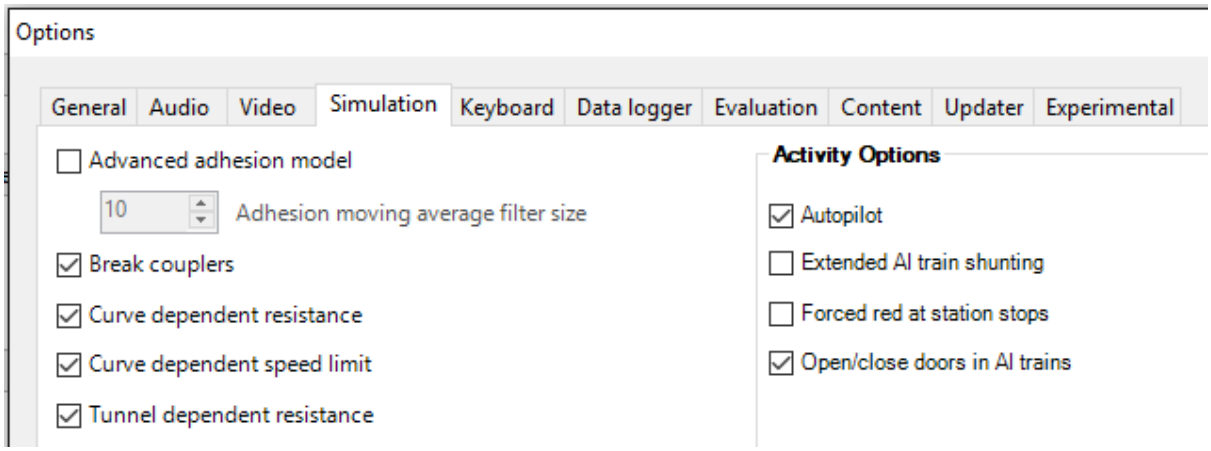
How It Works

Activity evaluation is enabled only for Activity mode. Checking some checkboxes within the various option tabs of the main menu provides additional parameters for the activity evaluation.

Here an example about the Options/General tab:

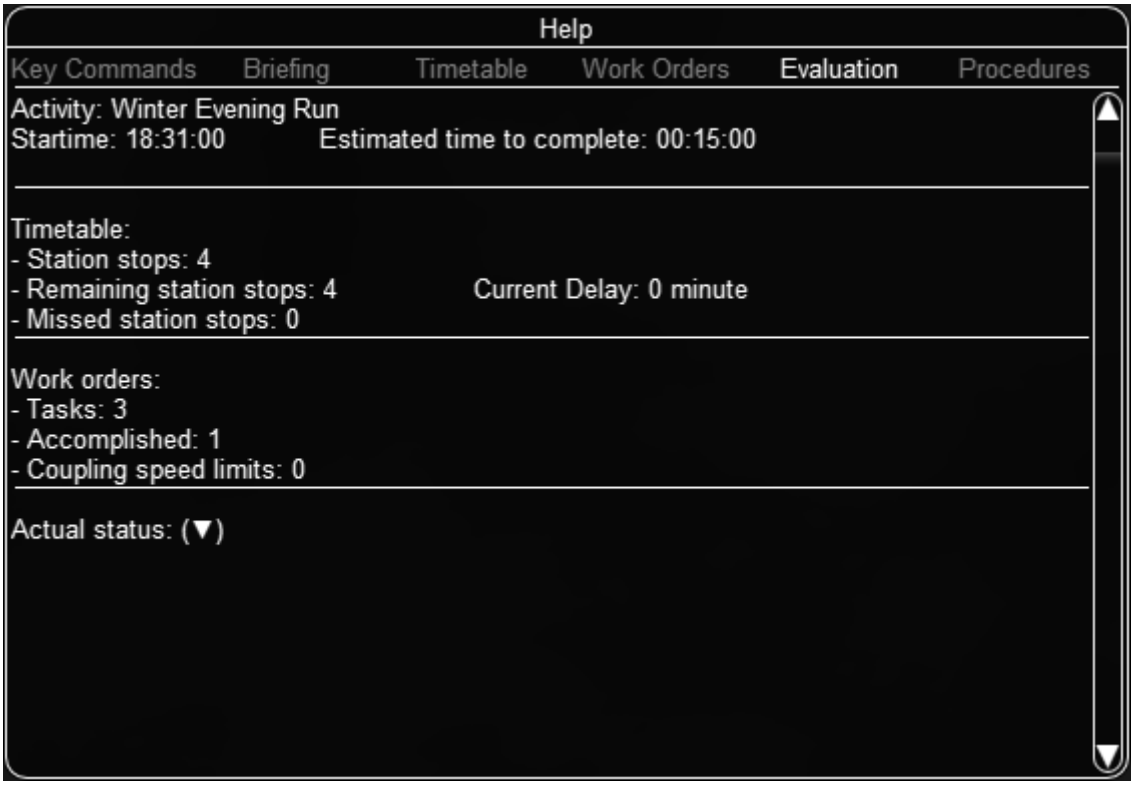


and here an example about the Options/Simulation tab:

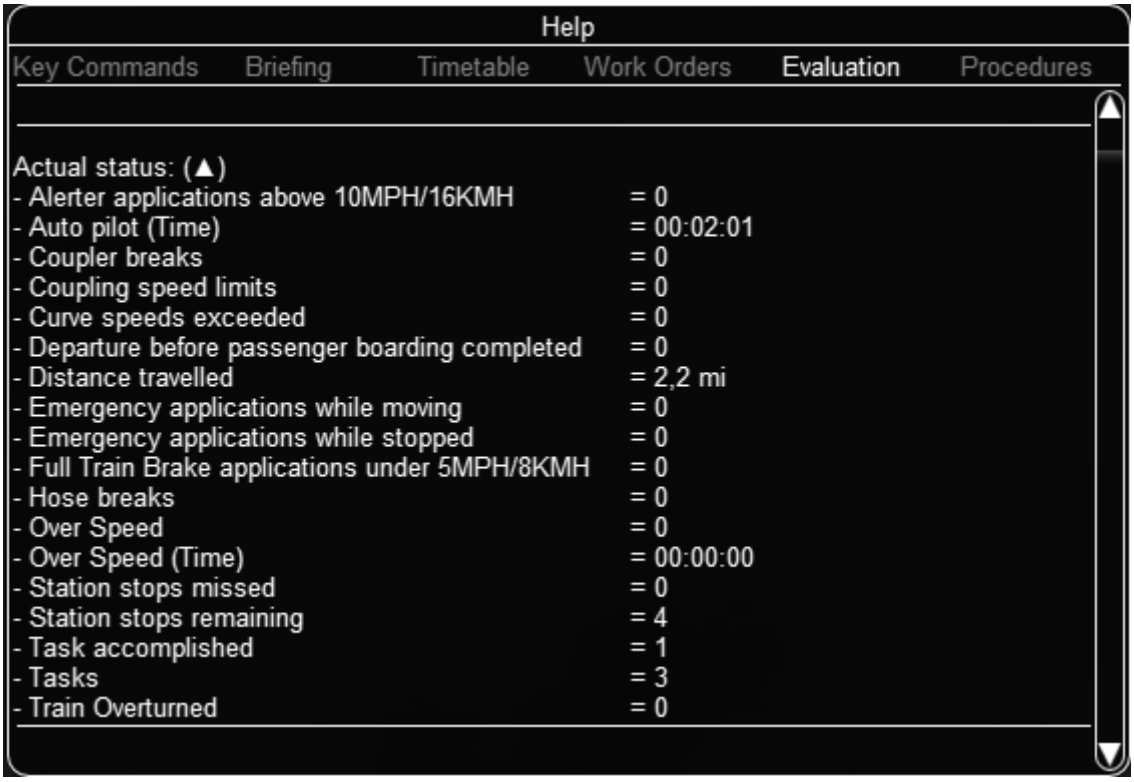


Checkboxes shown as unchecked in the two above pictures may be checked or unchecked, but don't have any effect on activity evaluation.

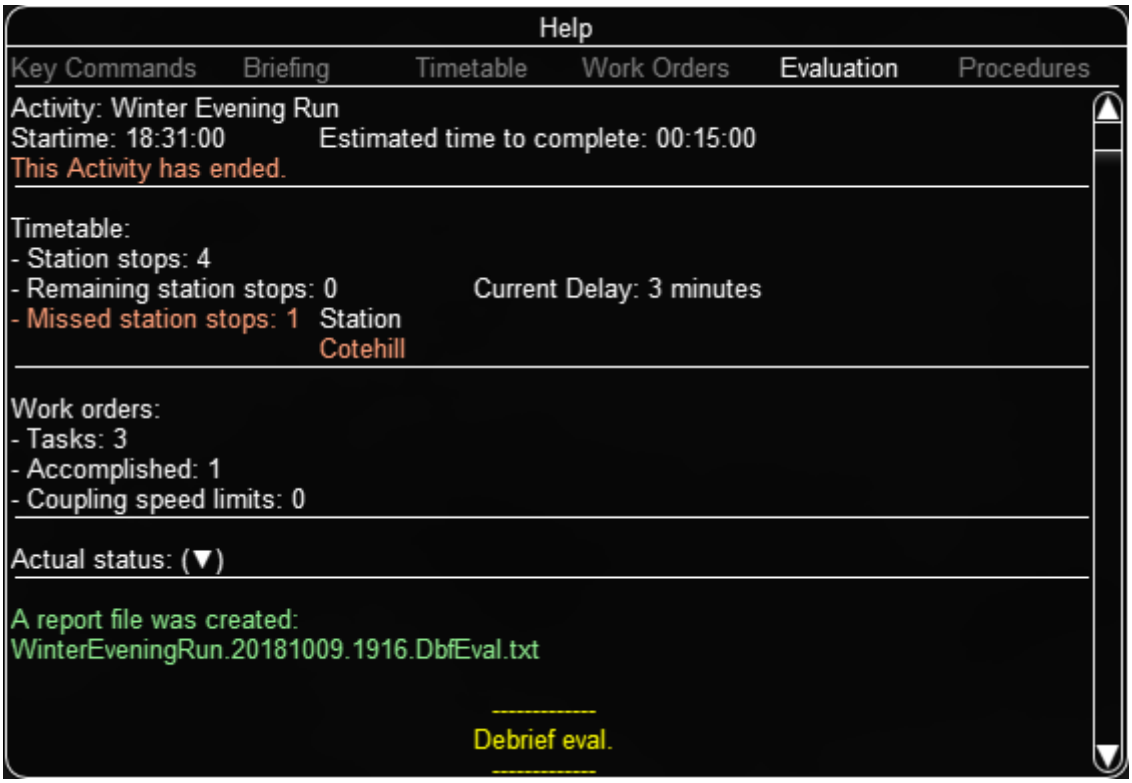
A tab named "Evaluation" is present on the F1 Help Information Monitor. Once the activity is running it displays dynamic information about the player train performance up to that moment.



Clicking **Actual status: (↓)** shows an expanded real-time display.



Clicking **Actual status: (↑)** collapses all items.



Once the activity has ended, as soon as the player views the Evaluation tab, a report file is created and shown in an editor window as follows.

This report is made up of several sections.

```
--------------------------------------------------------------------------------
This is a Debrief Eval for Open Rails.
--------------------------------------------------------------------------------
Version      = <none>.
Build        = 0.0.6855.25017 (2018-10-08 13:53:54Z).
Debrief file = C:\Users\OpenRails\AppData\Roaming\Open Rails\WinterEveningRun.20181009.1916.DbfEval.txt.
Executable   = RunActivity.exe.
--------------------------------------------------------------------------------
                                -------------
                                Debrief eval.
                                -------------

0-Information:
  Route          = Settle & Carlisle Line
  Activity       = Winter Evening Run
  Difficulty     = Medium
  Startime       = 18:31:00
  Estimated Time = 00:15:00
  Elapsed Time   = 00:19:02
  Autopilot Time = 00:13:56
  Travelled      = 9,1 mi
  Consist engine = 2 Diesel.
  Burned Diesel  = 11,7 g-us

1-Station Arrival, Departure, Passing Evaluation:
  Station Arrival      = 3
  Delay                = 00:03:32
  Missed station stops = 1    Station
                              Cotehill
  Departure before passenger boarding completed = 0
  Overall rating total = 38

2-Work orders:
  Task                Location              Status
  Assemble Train      -                     Done
  Drop Off            -                     -
  Drop Off            -                     -
  Coupling speed limits = 0
  Overall rating total  = 33

3-Speed Evaluation:
  Over Speed            = 0
  Over Speed (Time)     = 00:00:00
  Overall rating total  = 100

4-Freight Durability/Passenger Comfort Evaluation:
  Curve speeds exceeded          = 0
  Hose breaks                    = 0
  Coupler breaks                 = 0
  Train Overturned               = 0
  Overall rating total           = 100

5-Emergency/Penalty Actions Evaluation:
  Full Train Brake applications under 5MPH/8KMH  = 0
  Emergency applications while moving            = 0
  Emergency applications while stopped           = 0
  Alerter applications above 10MPH/16KMH         = 0
  Overall rating total                           = 100

                               Rating & Stars
                               **************

1- Station Arrival, Departure, Passing Evaluation  =  *
2- Work orders Evaluation                          =  *
3- Speed Evaluation                                =  * * * * *
4- Freight Durability/Passenger Comfort Evaluation =  * * * * *
5- Emergency/Penalty Actions Evaluation            =  * * * * *

--------------------------------------------------------------------------------
```

The report file OpenRailsEvaluation.txt is saved alongside the log file OpenRailsLog.txt and the default location for this is the Windows Desktop.

The Save Game (F2) command also copies any evaluation report alongside the save files so it can be kept

and reviewed. This copy is deleted when the other files for that save are deleted.

7.4.13 Basic Head Up Display (HUD)

By pressing <Alt+F5> you get some important data displayed at the top left of the display in the so-called Head Up Display (HUD). If you want the HUD to disappear, press <Alt+F5> again.

The HUD has 6 different pages. The basic page is shown at game start. To sequentially switch to the other pages press <Shift+Alt+F5>. After having cycled through all of the extended HUD pages, the basic page is displayed again.

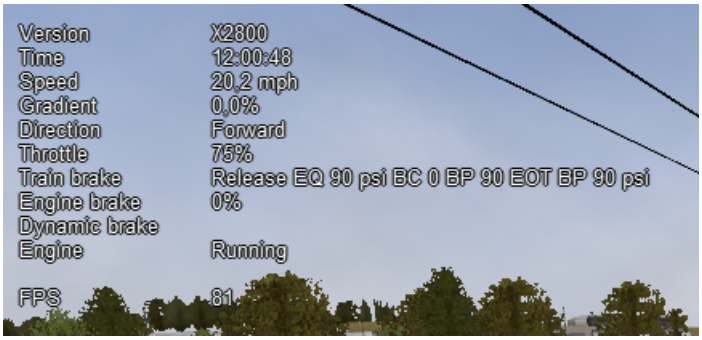
The basic page shows fundamental information. The other pages go into more detail, and are used mainly for debugging or to get deeper information on how OR behaves. They are listed in the *Analysis tools* subchapter.

The following information is displayed in the basic display:

- Version = The version of the Open Rails software you are running
- Time = Game time of the Activity
- Speed = the speed in Miles/Hr. or Kilometers/Hr.
- Gradient = Route gradient in % in that point
- Direction = Position of the Reverser - Electric, Diesel and Steam.
- Throttle = Displays the current position of the throttle, expressed as a percentage of full throttle. Throttle correctly uses Notches and configured % of power for Diesel engines or % of throttle for steam engines.
- Train Brake = Shows the current position of the train brake system and the pressure value of the train brakes. Braking correctly reflects the braking system used; hold/release, self-lapping or graduated release. The Train brake HUD line has two Brake Reservoir pressure numbers: the first is the Equalization Reservoir (EQ) and the second is the Brake Cylinder (BC) pressure. The two BP numbers report the brake pressure in the lead engine and in the last car of the train. The unit of measure used for brake pressure is defined by the option *Pressure unit*.
- Engine Brake = percentage of independent engine brake. Not fully releasing the engine brake will affect train brake pressures.
- Dynamic brake = if engaged, shows % of dynamic brake
- Engine = shows the running status of the engine. In case of a gear-based engine, after the `Engine` line a `Gear` line appears displaying the actual gear. `N` means no gear inserted.
- FPS = Number of frames rendered per second

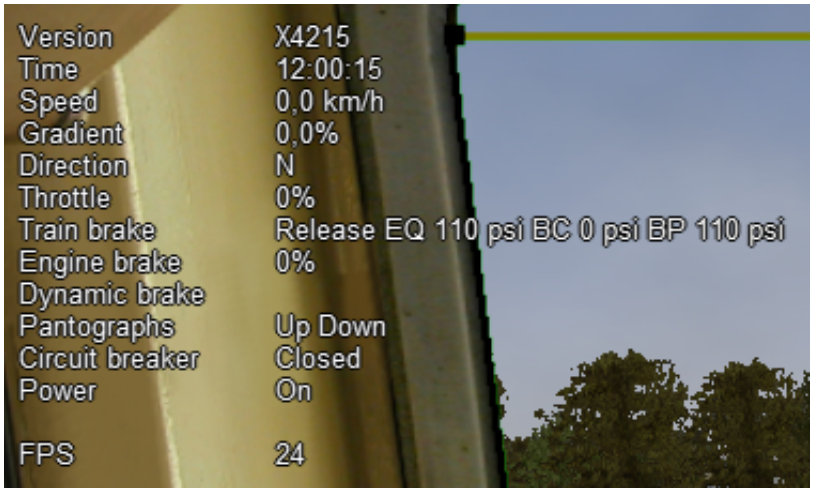
When applicable, an additional line indicationg whether Autopilot is active or not will be shown.

An example of the basic HUD for Diesel locomotives:



7.4.14 Electric Locomotives – Additional information

For electric locomotives information about the pantograph state is also shown, as well as info about the circuit breaker state and whether the locomotive has power (at least one pantograph raised and circuit breaker closed) or not.

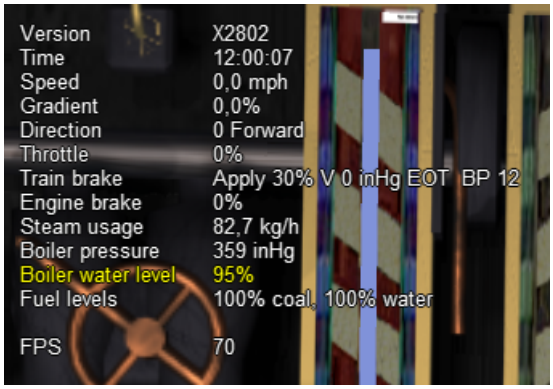


7.4.15 Steam Engine – Additional Information

When using a steam engine the following additional information is displayed in the HUD:

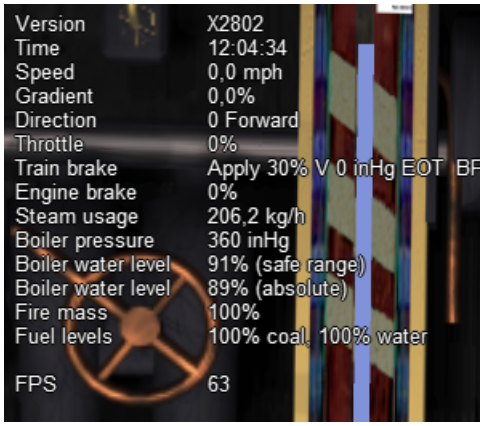
- Steam Usage in lbs/h, based on entirely new physics code developed by the Open Rails team. It is calculated by parsing the .eng file for the following parameters: number of cylinders; cylinder stroke; cylinder diameter; boiler volume; maximum boiler pressure; maximum boiler output; exhaust limit; and basic steam usage.
- Boiler pressure.
- Water level.
- Levels of coal and water in %.

An example of the basic HUD for Steam locomotives:



For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

The default *firing* setting is automatic fireman. If manual firing is engaged with `<Ctrl+F>`, then additional information is included:



7.4.16 Multiplayer – Additional Information

If a multiplayer session is active, the following additional information is shown: the actual status of the player (dispatcher, helper or client), the number of players connected and the list of trains with their distances from the train of the player viewing the computer.

7.5 Map Window

Use the map window to monitor and control train operation.

The map window is opened and closed from the graphics window by pressing <Ctrl+9>. You can toggle between the graphics window and an opened map window by pressing <Alt+Tab>.

The map window contains 2 tabs: Dispatcher and Timetable. Both provide maps of the route with each train following its own path.

The map window is resizable and can also be maximized, e.g. on a second display.

To pan, use the left mouse button to drag the map around.

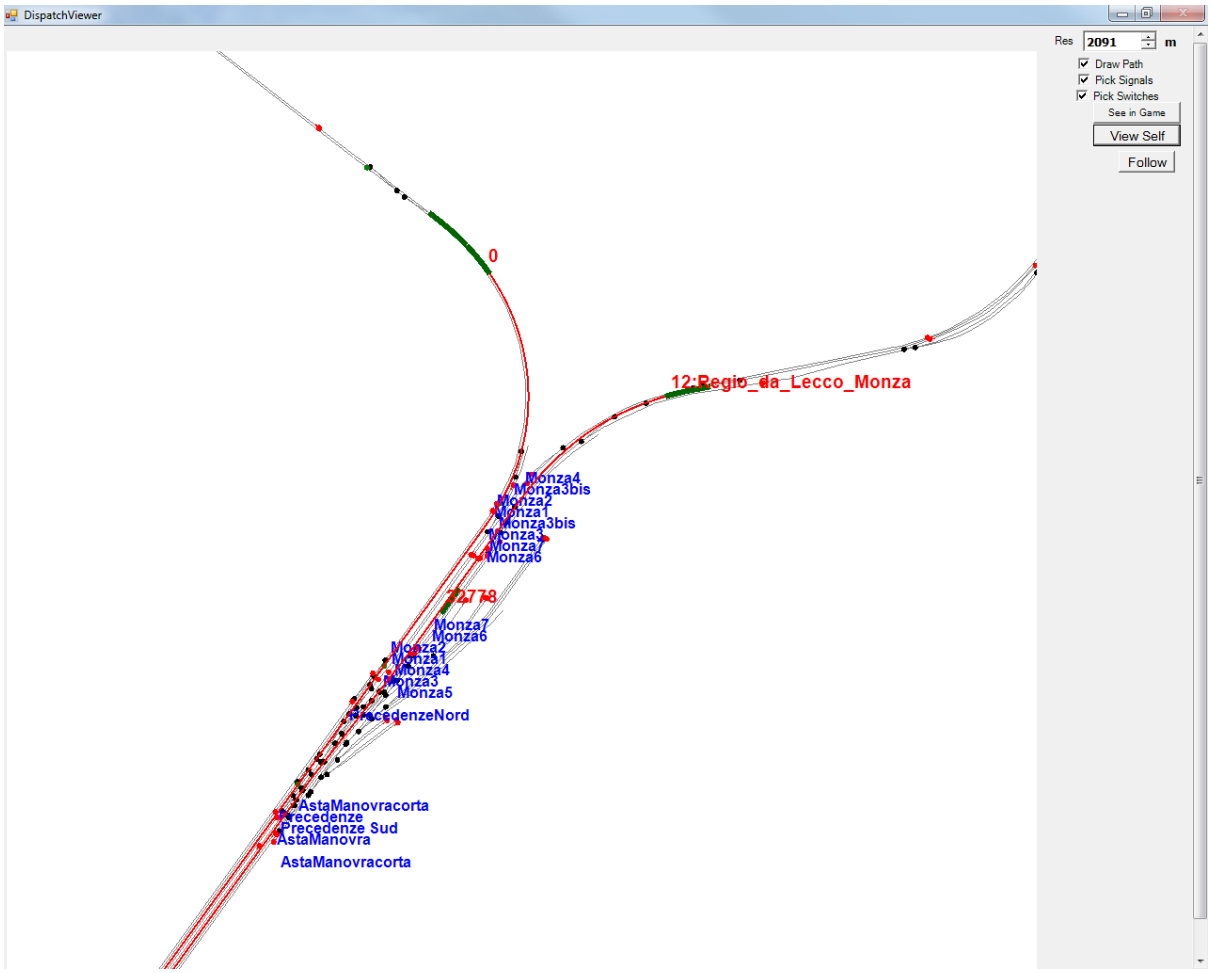
To zoom, use left and right mouse buttons together and drag vertically or use the mouse wheel.

To zoom in centred on a location, press Shift and click the left mouse button at that location.

To zoom out from a location, press Alt and click the left mouse button.

To zoom out fully, press Ctrl and click the left mouse button.

7.5.1 Dispatcher Tab



The dispatcher window shows the route layout, monitors the movement of all trains and allows you to change switches and signals. While the player train is identified by the `0` label, AI trains are identified by an Id number (as shown in the *Extended HUD for Dispatcher Information*), followed by the service name. Static consists are identified as in MSTS.

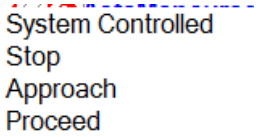
The state of the signals is shown (only three states are drawn), that is

- `Stop` – drawn in red
- `Clear_2` – drawn in green
- while all signals with restricting aspect are drawn in yellow.

The state of the switches is also shown. A switch shown with a black dot indicates the main route, while a grey dot indicates a side route.

When the `Draw path` is checked, the first part of the path that the train will follow is drawn in red. If a trailing switch in the path is not in the correct position for the path, a red X is shown on it.

When left- or right-clicking on a signal, a pop-up menu appears:



Using the mouse, you can force the signal to Stop, Approach or Proceed. Later you can return it to System Controlled mode.

For signals using the TrainHasCallOn functions as described *here*, an additional option labeled `Enable CallOn` will appear in the pop-up menu. The use of this function allows a train to enter into an occupied platform if the dispatcher allows so.

By left- or right-clicking on a switch, a small pop-up menu with the two selections `Main route` and `Side route` appears. By clicking on them you can throw the switch, provided the OR AI dispatcher allows it.

Using the dispatcher window for AI trains is described *here below*.

The two checkboxes `Pick Signals` and `Pick Switches` are checked as default. You can uncheck one of them when a signal and a switch are superimposed in a way that it is difficult to select the desired item.

You can click a switch (or signal) in the dispatcher window and press `<Ctrl+Alt+G>` to jump to that switch with the free-roam (8-key) camera.

If you click on `View Self` the dispatcher window will center on the player train. However, if the train moves, centering will be lost.

You can select a train by left-clicking with the mouse its green reproduction in the dispatcher window, approximately half way between the train's head and its name string. The train body becomes red. Then if you click on the button `See in game` the main Open Rails window will show this train in the views for the 2, 3, 4 or 6 keys, (and the 5-key view if available for this train). Display of the new train may require some time for OR to compute the new image if the train is far away from the previous camera view.

Take into account that continuous switching from train to train, especially if the trains are far away, can lead to memory overflows.

If after a train selection you click on `Follow` the dispatcher window will remain centered on that train.

Using dispatcher tab for AI trains

What is described here is valid only for activity mode and explore in activity mode.

There are cases where it would be advisable to re-route an AI train to manage standoffs, train passings, train priorities. In this case, using the dispatcher window it is possible to re-route an AI train (e.g. on a siding) and then to get it back on the original route. The feature anyhow also allows to re-route it without getting it back on the original route.

It is suggested to look at this video which explains some practical case https://youtu.be/-f0XVg7bSgU before continuing reading.

To perform this correctly and in a way closer to reality, some rules have to be followed. The concept is that switches must be manually thrown only if they aren't reserved by a train. To be sure of this it is necessary to force to stop the last signal(s) between train(s) and switch, in case such signal is not already at stop. Once the switch is manually thrown, the signal in front of the train that has to be re-routed must be set to the "System controlled" state if it had been forced to stop before. At that point OR breaks down the old train's route and re-computes a new one, taking into account the moved switch. More switches may be forced on the route (e.g. both the switches to enter the siding and the ones to re-enter the main line).

Signals must never be forced to clear or approach.

If an AI train is re-routed on a route which isn't at the moment re-entering the original route, its path information in the dispatcher info HUD is displayed in yellow.

Station platform stops are re-assigned to adjacent platforms, if available. Events and waiting points in the abandoned part of route will be lost.

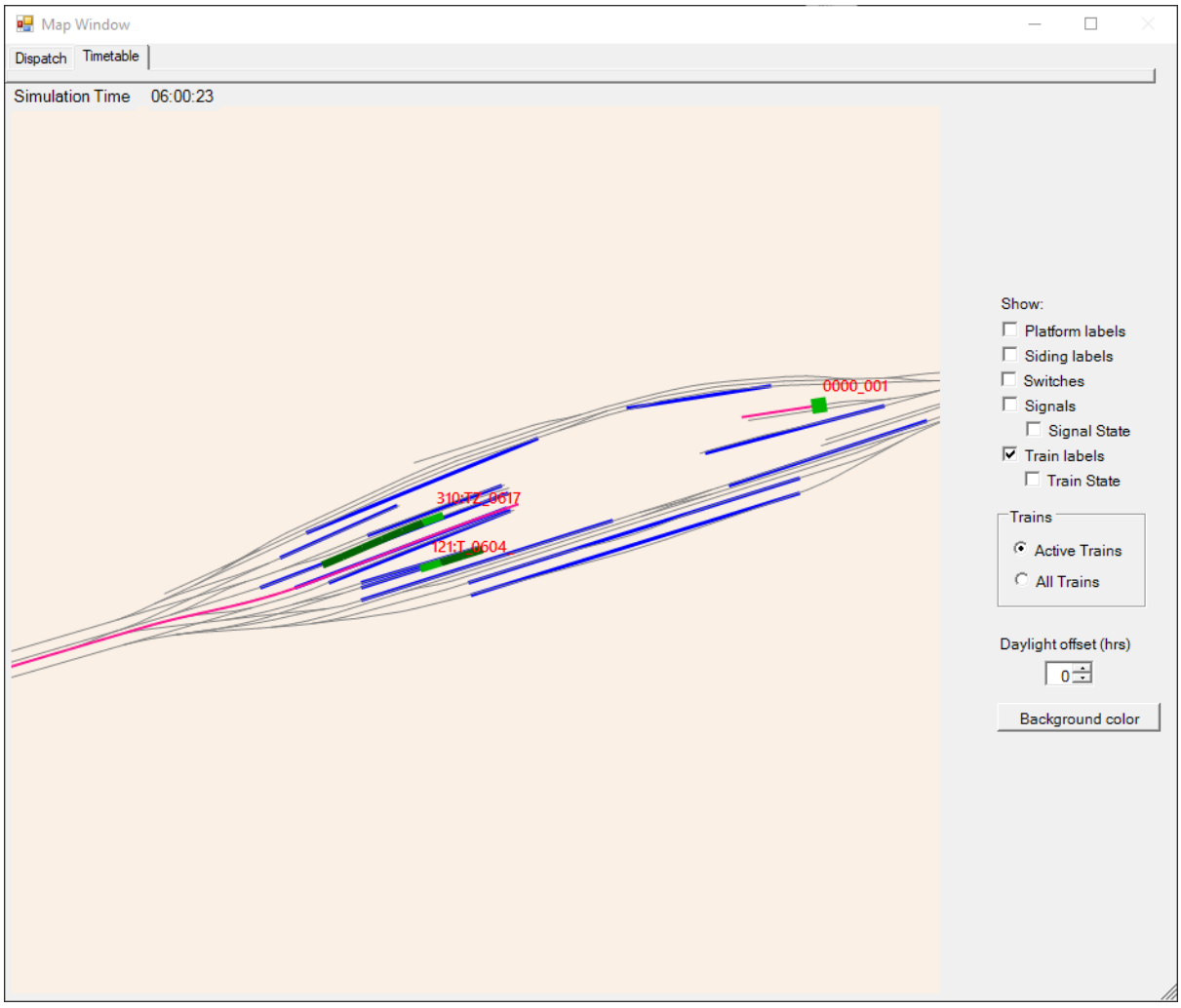
The re-routed train may be also the player train (be it autopiloted or not).

7.5.2 Timetable Tab

This tab shows the same route and trains as the dispatch tab but, with its focus on a timetable of trains, is provided to assist timetable builders.

In this tab, for clarity, you can use the checkboxes to hide or reveal the labels for platforms, sidings, switches, signals and trains. The simulation time is also on view.

As shown below, the basic red train label identifies the train.



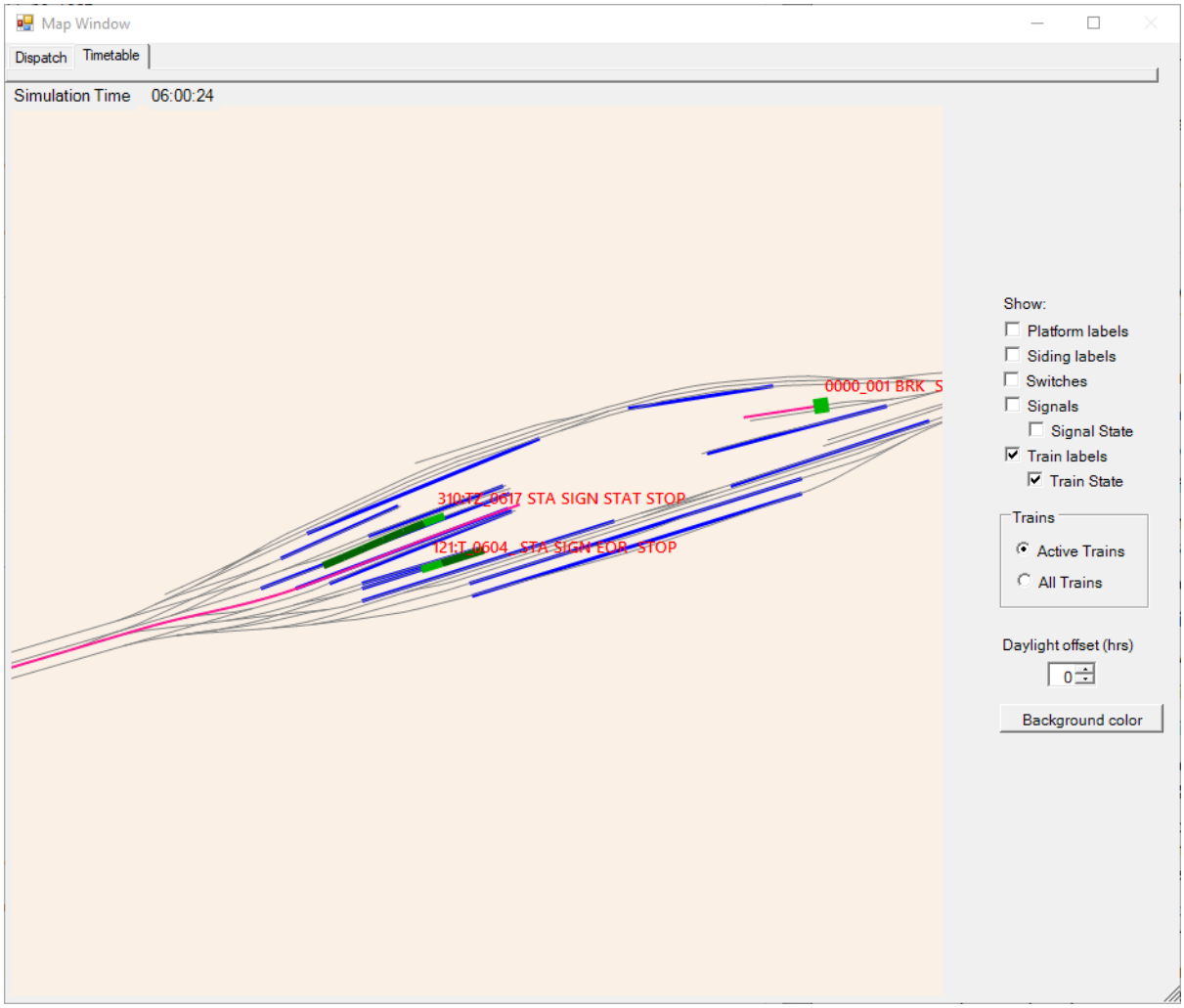
Trains are drawn in green except that locos are drawn in brown. To indicate direction, the leading vehicle is draw in a lighter shade.

The “Active trains” selection shows trains that are currently delivering a service. The “All trains” selection also shows inactive and static trains with labels in dark red.

Inactive trains are not part of a current service - i.e. their start time has not been reached or they have arrived at their destination and not yet been re-formed for another service - see *#dispose commands*.

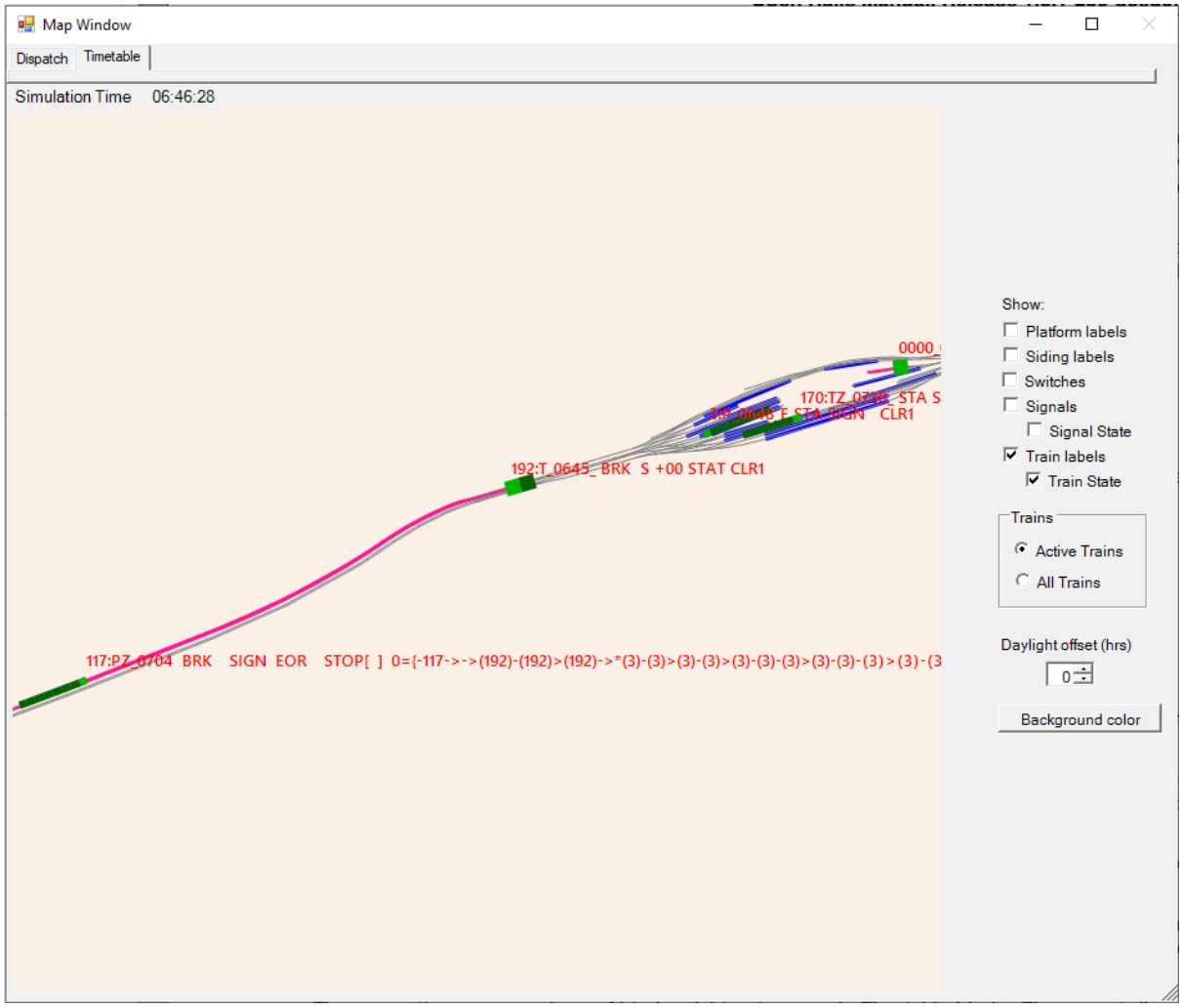
Static trains do not move and are shown in gray. They created with the $static command.

When you select the “Train state” checkbox, the train labels extend to provide key information matching that from the *HUD* as shown below:

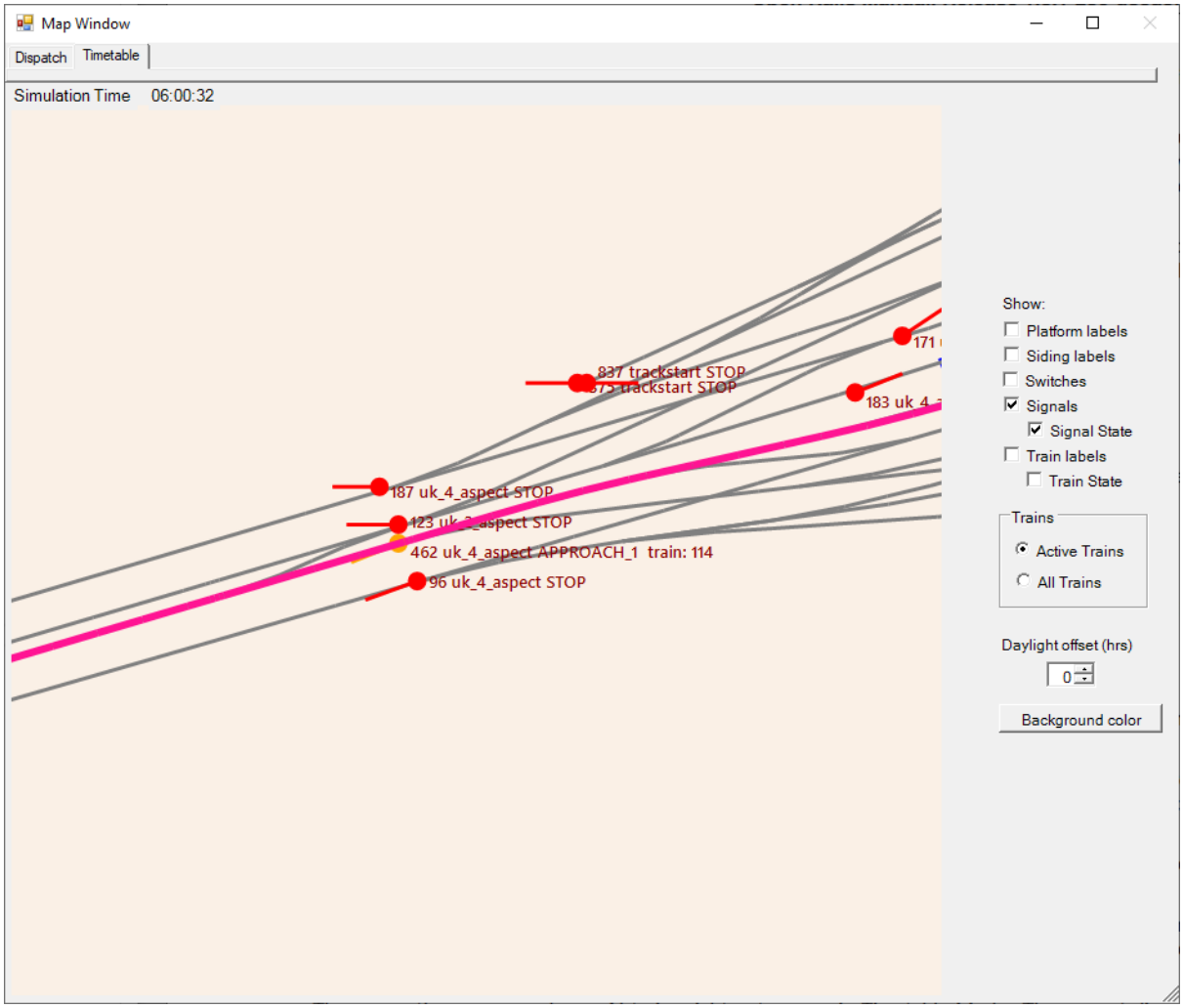


The path element of the train state can be very lengthy, so this is only shown where the path contains the characters # & * ^ ~ which indicate *a track section that is in contention*.

In the image below, train 192 crosses the path of train 117.



The “Signal state” checkbox reveals the aspect of each signals and also indicates the id number of the train that is approaching. In this image, signal 462 is showing an APPROACH_1 aspect for train 114.



The adjustment for “Daylight offset (hrs)” is provided for convenience to advance the sun as it moves across the sky so that night time trains can be more easily observed in daylight.

7.6 Additional Train Operation Commands

OR supports an interesting range of additional train operation commands. Some significant ones are described here.

7.6.1 Diesel Power On/Off

With the key <Shift+Y> the player diesel engine is alternately powered on or off. At game start the engine is powered on.

With the key <Ctrl+Y> the helper diesel locomotives are alternately powered on or off. At game start the engines are powered on.

Note that by using the Car Operation Menu you can also power on or off the helper locomotives individually.

7.6.2 Initialize Brakes

Entering this command fully releases the train brakes. Usually the train must be fully stopped for this to be allowed. This action is usually not prototypical. Check the keyboard assignment for the keys to be pressed. The command can be useful in three cases:

- A good number of locomotives do not have correct values for some brake parameters in the .eng file; MSTS ignores these; however OR uses all these parameters, and it may not allow the brakes to release fully. Of course, it would be more advisable to correct these parameters.
- It may happen that the player does not want to wait for the time needed to recharge the brakes; however the use of the command in this case is not prototypical of course.
- The player may wish to immediately connect brake lines and recharge brakes after a coupling operation; again, the use of the command is not prototypical.

Note that this command does not work if the *Emergency Brake* button has been pressed – the button must be pressed again to cancel the emergency brake condition.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

7.6.3 Connect/Disconnect Brake Hoses

This command should be used after coupling or decoupling. As the code used depends on keyboard layout, check the keys to be pressed as described in *keyboard options* or by pressing F1 at runtime. More information on connecting brakes and manipulating the brake hose connections can be found *here* and *here*.

7.6.4 Doors and Mirror Commands

Note that the standard keys in OR for these commands are different from those of MSTS.

7.6.5 Wheelslip Reset

With the keys `<Ctrl+X>` you get an immediate wheelslip reset.

7.6.6 Toggle Advanced Adhesion

Advanced adhesion can be enabled or disabled by pressing `<Ctrl+Alt+X>`.

7.6.7 Request to Clear Signal

When the player train has a red signal in front or behind it, it is sometimes necessary to ask for authorization to pass the signal. This can be done by pressing `<Tab>` for a signal in front and `<Shift+Tab>` for a signal behind. You will receive a voice message reporting if you received authorization or not. On the Track monitor window the signal colours will change from red to red/white if permission is granted.

7.6.8 Change Cab

All locomotives and some passenger cars have a forward-facing cab which is configured through an entry in the ENG file. For example, the MSTS Dash9 file `TRAINSET\DASH9\dash9.eng` contains the entry:

```
CabView ( dash9.cvf )
```

Where a vehicle has a cab at both ends, the ENG file may also contain an entry for a reversed cab:

```
CabView ( dash9_rv.cvf )
```

OR will recognise the suffix `_rv` as a rear-facing cab and make it available as follows.

When double-heading, banking or driving multiple passenger units (DMUs and EMUs), your train will contain more than one cab and OR allows you to move between cabs to drive the train from a different position. If you change to a rear-facing cab, then you will be driving the train in the opposite direction.

If there are many cabs in your train, pressing `<Ctrl+E>` moves you through all forward and rear-facing cabs in order up to the last cab in the train. If you end up in a rear-facing cab, your new *forward* direction will be your old *backward* direction. So you will now drive the train in the opposite direction.

A safety interlock prevents you from changing cabs unless the train is stationary and the direction lever is in neutral with the throttle closed.

7.6.9 Train Oscillation

You can have train cars oscillating (swaying) by hitting `<Ctrl+V>`; if you want more oscillation, click `<Ctrl+V>` again. Four levels, including the no-oscillation level, are available by repeating `<Ctrl+V>`.

7.6.10 Manual emergency braking release

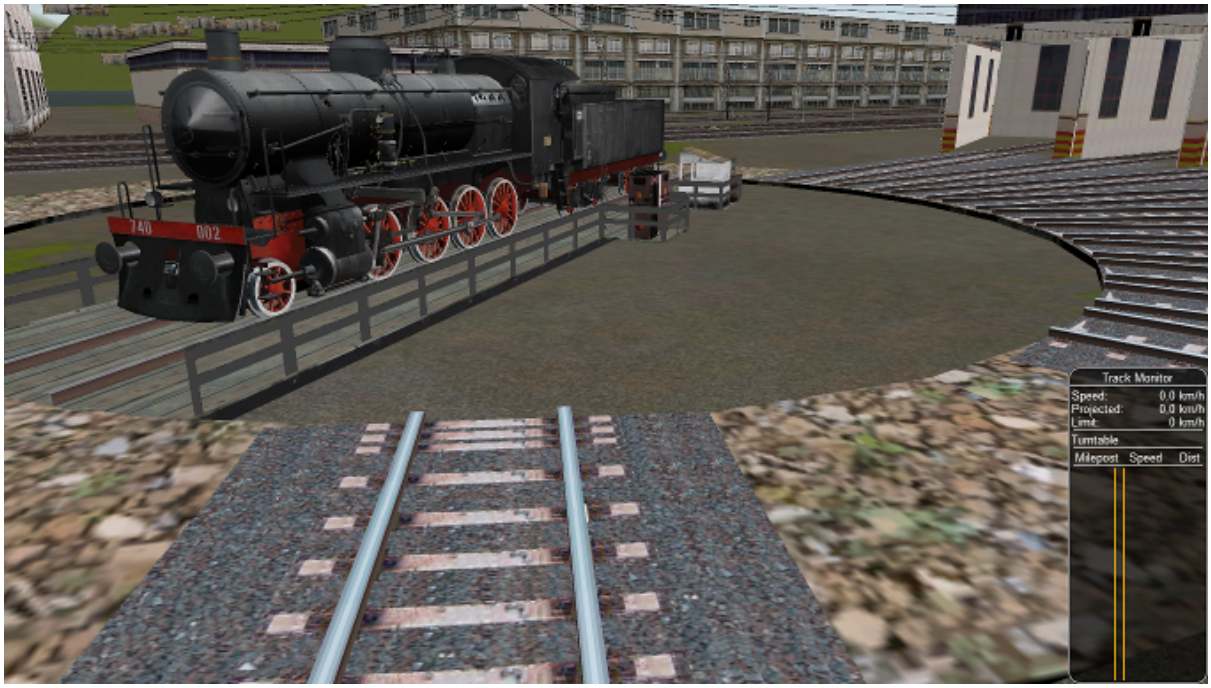
In some cases where the emergency braking is triggered by the simulator, it is possible to release the emergency braking by pressing `<Shift+Backspace>`.

The cases where the reset is allowed are:

- Signal passed at danger
- Trailed misaligned switch

7.7 Engaging a turntable or a transfertable

Turntable and transfertable operation is possible in explore mode, activity mode and timetable mode.



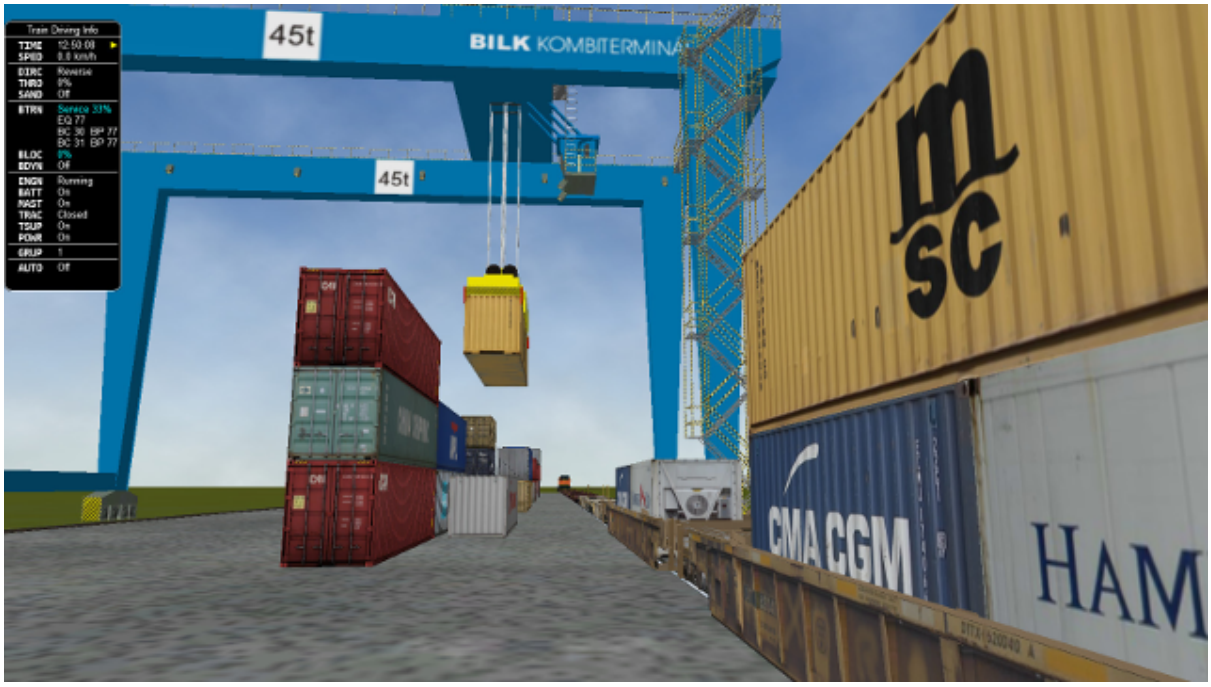
A turntable or transfertable can be moved by the player only if it is viewed by him on the screen. If more than one turntable or transfertable is on view, the nearest one can be moved. The trainset (or trainsets) to be rotated or translated must be completely on the turntable or transfertable to start rotation. Messages of type "Train front on turntable" and "train rear on turntable" help stating that the train is fully on the rotating or transferring bridge. Before starting rotating or translating the train must be fully stopped, with reverser in neutral position and zeroed throttle. Moreover, if in activity or timetable mode, the player must first pass to *manual mode* pressing <Ctrl+M>. At this point you can rotate the turntable clockwise (or move the trasfertable to the right of its origin) with keys <Alt+C>, and counterclockwise (or move the trasfertable to the left of its origin) with keys <Ctrl+C>. You must keep the keys pressed to continue rotation or translation . When the first of the two rails of the rotating or translating bridge is between the two rails where you want to stop, release the keys. Rotation or translation will continue up to perfect alignment. If necessary exit from manual mode (if you are again on a path in activity mode) and move the loco out of the turntable or transfertable. During rotation the train is in *Turntable* state (this can be seen in the *Track Monitor*).



It is also possible to rotate or translate standalone wagons. They have to be pushed or pulled to the turntable or transfertable, the locomotive must exit the turntable or transfertable and the wagon can be rotated or translated. It is suggested to read also *this paragraph* to better understand what is possible with turntables and transfertables.

7.8 Loading and Unloading Containers

Provided that the wagons and the container cranes in the route fulfill the rules indicated *here*, containers can be unloaded and loaded on wagons at locations where a container crane is present.



The loading and unloading operations are started by the player, by pressing the key <T> for loading, and the key <Shift-T> . The operation is performed on the first wagon (starting from the locomotive) which is within the container crane displacement range and which fulfills the required conditions (e.g. loading space available for loading, container present for unloading). So, if a train has only empty wagons and the locomotive is within the container crane displacement range, the first wagon is loaded first, then the second and so on up to the last wagon within the crane displacement range. At that point, if there are further wagons to be loaded, the train must be moved forward so that a new group of wagons is within the crane displacement range, and Load operations can be resumed.

Every keypress loads or unloads a single wagon.

In some cases it can occur that during a load operation the crane stops motion and the following message appears on the display: `"Wagon out of range: move wagon towards crane by {0} metres"`; this occurs when the wagon is at the boundary of the crane displacement range; the player must move the wagon towards the inside of the crane displacement range and stop the train. The crane will then continue its loading mission up to the end.

Saves (key <F2>) are rejected and a message appears on the display when a loading or unloading operation is ongoing.

7.9 Autopilot Mode

When in activity mode or in Explore in activity mode, through this feature it is possible to stay in the cab of the player train, but to let Open Rails move the train, respecting path, signals, speeds and station stops.

It is possible to switch the player train between autopilot mode and player driven mode at run time.

Autopilot mode is not a simulation of a train running with cruise control; instead, it is primarily a way to test activities more easily and quickly; but it can also be used to run an activity (or part of it, as it is possible to turn autopilot mode on or off at runtime) as a trainspotter or a visitor within the cab.

Autopilot mode is active only in activity mode (i.e. not in explorer or timetable modes).

When starting the game with any activity, you are in player driving mode. If you press Alt+A, you enter the autopilot mode: you are in the loco's cabview with the train moving autonomously accordingly to path and station stops and of course respecting speed limits and signals. You still have control over the horn, bell, lights, doors, and some other controls that do not affect train movement. The main levers are controlled by the autopilot mode, and indications are correct.

You can at any moment switch back to player driven mode by pressing `<Alt+A>`, and can again switch to autopilot mode by again pressing `<Alt+A>`.

When in player driven mode you can also change cab or direction. However, if you return to autopilot mode, you must be on the train's path; other cases are not managed. When in player driven mode you can also switch to manual, but before returning to autopilot mode you must first return to auto mode.

Station stops, waiting points and reverse points are synchronized as far as possible in the two modes.

Cars can also be uncoupled in autopilot mode (but check that the train will stop in enough time, otherwise it is better to change to player driven mode). A static consist can also be coupled in autopilot mode.

The Request to Clear signal (`<Tab>` key) works in the sense that the signal opens. However in autopilot mode at the moment that the train stops you must switch to player driven mode to pass the signal and then you can return to autopilot mode.

Note that if you run with Advanced Adhesion enabled, you may have wheelslip when switching from autopilot mode to player driven mode.

The jerky movements of the levers in autopilot mode are the result of the way that OR pilots the train.

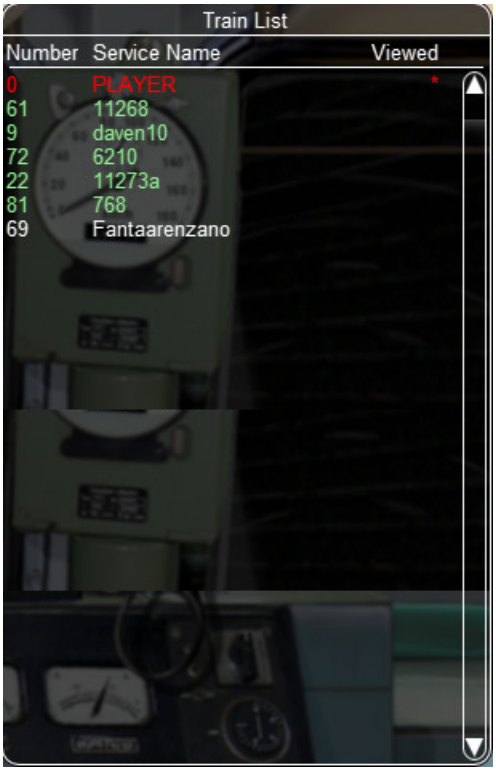
7.10 Changing the Train Driven by the Player

7.10.1 General

This function only works in activity mode, and allows the player to select another (existing) train from a list and to start driving it.

This function can be called more than once. A new information window has been created to support this function: the `Train List` window (opened with Alt+F9). It contains a list of all of the AI trains and of the static trains equipped with a locomotive with cab, plus the player train.

Here an example of an initial situation:



The current player train is shown in red. The star at the end of the line indicates that the cameras (cab camera is managed differently) are currently linked to that train.

AI trains whose loco(s) have at least a cab are shown in green. They are eligible for player train switching.

Static trains with loco and cab are shown in yellow.

Other AI trains are shown in white.

By left-clicking in the list for the first time on an AI train, the cameras become linked to that train. A red star appears at the end of the line. This is partially equivalent to clicking on <Alt+9>, but in this method the desired train is immediately selected and may become the player train.

Here is the intermediate situation:



By left-clicking a second time on the AI train (usually when it has completely appeared on the screen - if it is far away from the player train this can require several seconds to load the *world* around the train) the switch of control occurs.

The AI train string now becomes red and is moved to the first position.The train can be driven, or set to autopilot mode. The former player train becomes an AI train.

Here is the final situation:



If the second left-click was performed with the Shift key down, the former player train still becomes an

AI train, but it is put in a suspended mode (only if its speed is 0). It won't move until it becomes a player train again. A suspended train is shown in orange color on the Train List window.

The new player train can can be switched to manual mode, can also request to pass signals at danger with the <Tab> command, and can be moved outside of its original path. However before switching control to still another train, the new player train must be returned to the original path or put in suspend mode; or else it will disappear, as occurs for AI trains running outside their path.

The sequence may be restarted to switch to a new train or to switch back to the initial player train.

Train switching also works in activity mode together with multiplayer mode, in the sense that the dispatcher player can switch its played train, and the related information is sent to the client players.

The Train List window is also available in *Timetable mode*. In this case the names of all trains except the player train are shown in white (they can't be driven), however with a single click on a train in the window the external view cameras become linked to that train, as occurs with the Alt-9 command described *further below*.

7.10.2 Switching to a static train

In the Train List window the drivable static consists (that is the ones that have at least an engine provided with a cab) are also listed (in yellow color).

To ease recognition static consists are named STATIC plus the ID number as present in the .act file (e.g. STATIC - 32768).

The procedure to select a static consist in order to drive it is similar to that used to drive another non-static train train: with the first click on the static consist line in the Train List window the camera (if there wasn't the Cab camera active) moves to the static consist. With the second click the game enters into the cab of the static consist. If the second click occurs with the Shift key pressed, the old player train goes into a suspended state (else it enters autopilot mode, autonomously moving itself along its path).

The static consist becomes a standard train without a path - a pathless train. It runs in manual mode, and so it can be managed with all the thrills and frills available for manual mode. Signals can be cleared in the dispatcher window or alternatively requests for permission can be issued, switches can be moved, direction can be changed, cars can be coupled and uncoupled. If the train goes out of control (e.g. because of SPAD), CTRL+M has to be pressed first to exit emergency braking.

With stopped pathless train, if a new player train is selected in the Train List window, the pathless train returns to being a STATIC consist.

The pathless train can also couple to another train (e.g. an AI train or the initial player train). The coupled train becomes incorporated into the pathless train, but now more possibilities are available:

- The pathless train incorporating the AI train continues to be driven as a pathless train; later on the run it could uncouple the incorporated train, which would continue autonomously if it is still on its path.
- By clicking once on the incorporated AI train line in the Train List window it is the pathless train that becomes absorbed into the AI train, which now can operate on its path in autopilot mode or in player driven mode.
- Once the pathless train has coupled to the AI train, an uncouple operation can be performed with the F9 window (between any couple of cars). The pathless train can be driven further (with modified composition) and also the AI train can run further, provided both retain at least one locomotive.

7.10.3 Waiting point considerations

A waiting point icon showing a hand has been added for the *Track Monitor*, that is shown when WPs (waiting points) for new player trains are met in the path. This because the player should know that his train (when run as AI train) would stop at a point for a certain time. The WP is red when approaching it. When the train stops at it, it becomes yellow, and disappears when the time to depart is reached. When the new player train is run in autopilot mode, the train automatically stops for the required time at the WP.

If the activity foresees that the new player train has to execute an Extended AI Shunting function, OR allows this function to be executed. When the train runs in autopilot mode such functions are executed automatically; when it runs in player driven mode, the player must act to uncouple cars; in this case pop-up messages based on the activity events window appear to help the player.

Care has been taken when the player is driving a train that was foreseen to disappear due to an Extended AI Shunting function, as e.g. when it merges into another train or when it is part of a join-and-split function and is incorporated within another train. In these cases, when the coupling occurs, the player is automatically moved to the train that remains alive.

7.11 Changing the View

Open Rails provides all of the MSTS views plus additional view options:

- A 3D interior cabview option (where a 3D cabview file is available);
- Control of the view direction using the mouse (with the right-hand button pressed);
- The exterior views (keys 2,3,4,6) and the interior view (key 5) can be attached to any train in the simulation by the Alt+9 key as described below;
- The <Alt+F9> key shows the *Train List* window, which not only allows attaching the exterior views to any train, but also, in Activity mode, to move to the Cab and *drive any train in the simulation*;
- when in passenger view (key 5) it is possible to toggle the viewpoint from one side to the other of the wagon, and to jump to other viewpoints if defined, as described below;
- a "special viewpoint" trackside view camera is available, as described below.

All of the required key presses are shown by the F1 Help key in the game. Note that some of the key combinations are different in Open Rails than in MSTS. For instance, in Open Rails the cab Headout views from the cab view are selected by the Home and End keys, and the view direction is manipulated by the four arrow keys, or the mouse with the right-hand button depressed.

The commands for each of the views are described below.

- Key <1> opens the driver's view from the interior of the controlling cab of the player locomotive.
  - In case the 2D view is selected, the 2D view can be cycled between the fixed left, front, and right views with the <Left> and <Right> arrow keys. The cab itself can be hidden with the <Shift+1> key. (The 2D view is constructed from three 2D images, so the actual camera position can only be modified by editing the contents of the .cvf file.) If there is a mismatch between the aspect ratio of the cab and the aspect ratio of the monitor, OR will clip the cab and show only the portion that fits within the display. This clip can be panned around to reveal the rest of the cab with the <Up>, <Down>, <Alt+Left>, and <Alt+Right> keys. Alternatively, if placed into letterboxing mode, by using the <Ctrl+1> key, OR will render the full cab without a clip and cover the remaining space with black bars.
  - In case the 3D view is selected, the camera position and view direction are fully player controllable.
- Key <Alt+1> switches between 2D and 3D cabs if both are available.
- The entire cab view can be moved to other cabs (if available) in the player train by successive presses of <Ctrl+E>; the train must be stopped and the direction switch in Neutral.

- The headout views (if available) are selected by <Home> (right hand side, looking forward) or <End> (left hand side, looking back) and the headout view direction is controlled by the mouse with the right button depressed. If there are multiple locomotives, <Alt+PgUp> and <Alt+PgDn> move the headout views.

Rotation of the camera view in any direction is controlled by the mouse with the right-hand button depressed (or alternatively by the four arrow keys). The camera's position is moved forward or backward along the train axis with the PageUp and PageDown keys, and moved left or right or up or down with <Alt> + the four arrow keys. The headout views (if available) are selected by <Home> (right hand side, looking forward) or <End> (left hand side, looking back) and the outside view direction is controlled by the mouse with the right button depressed.

- Keys <2> and <3> open exterior views that move with the active train; these views are centered on a particular *target* car in the train. The target car or locomotive can be changed by pressing <Alt+PgUp> to select a target closer to the head of the train and <Alt+PgDn> to select a target toward the rear. The 2-View selects the train's head end as the initial target, the 3-View the last car. Alt+Home resets the target to the front, <Alt+End> to the rear of the train. With commands <Shift+Alt+7(Numpad)> and <Shift+Alt+1(Numpad)> cameras 2 or 3 move gently forward or backward along a stopped or moving train. By pressing a second time the key sequence the motion stops.

The camera's position with respect to the target car is manipulated by the four arrow keys – left or right arrows rotate the camera's position left or right, up or down arrows rotate the camera's position up or down while remaining at a constant distance from the target. The distance from the camera to the target is changed by zooming with the <PgUp> and <PgDn> keys. Rotation of the camera view direction about the camera's position is controlled by holding down the <Alt> key while using the arrow buttons, or by moving the mouse with the right mouse button depressed. The scroll wheel on the mouse zooms the screen image; the field of view is shown briefly. <Ctrl+8> resets the view angles to their default position relative to the current target car.

- Key <4> is a trackside view from a fixed camera position with limited player control - the height of the camera can be adjusted with the up and down arrow keys. Repeated pressing of the 4-key may change the position along the track.
- Key <Shift+4> is a special viewpoint trackside view: the camera locates itself at platforms, or looks at the train following a spawned road car along the road, or at a level crossing, when such special viewpoints are near to the train; else it behaves like the standard trackside view camera. When the camera follows a spawned road car, speed of the road car can be increased and decreased within certain limits with keys <9(NumKey)> and <3(NumKey)> to adjust the speed of the camera with that of the train. By pressing key <Shift+4> when the camera is on a special viewpoint, another one, if available, is selected.
- Key <5> is an interior view that is active if the active train has a *passenger view* declaration in any of its cars (or in the caboose). The view direction can be rotated by the arrow keys or the mouse with right button pressed. The camera position is moved forward or backward along the train axis with the PageUp and PageDown keys, and moved left or right or up or down with <Alt> + the four arrow keys. Successive presses of the <5> key will move the view to successive views (if they exist) within the active train. Note that the *active train* may be an AI train selected by <Ctrl+9>. By pressing <Shift+5> the viewpoint can be toggled to the other side of the wagon (if it was right side, it moves to left side and vice-versa). If more viewpoints are defined for such wagon as explained *here*, pressing <Shift+5> moves through the various viewpoints.
- Key <6> is the brakeman's view – the camera is assumed to be at either end of the train, selected by <Alt+Home> and <Alt+End>. Rotation is controlled by the arrow keys or mouse with right button depressed. There is no brakeman's view for a single locomotive.
- Key <8> is the free camera view; the camera starts from the current Key-2 or Key-3 view position, and moves forward (<PgUp> key) or back (<PgDn> key) along the view direction. The direction is controlled by the arrow keys or the mouse with right button depressed. The speed of motion is controlled by the <Shift> (increase) or <Ctrl> (decrease) keys. Open Rails saves the position of previous Key 8 views and can recall them by repeatedly pressing <Shift+8>.
- <Alt+9> is an ORTS feature: it controls the target train for the Key 2, 3, 4, 5 and 6 views during activities or timetable operations. If there is more than one active train or there are consists declared

in the activity for pickup, pressing this key combination will set the view to display each train or consist in turn. To return to the player train, press the <9> key. There may be a delay for each change of view as Open Rails calculates the new image. The cab view and data values in the F4 window always remain with the Player train. To directly select which train is to be shown either use the *Dispatcher Window* or the <Alt+F9> option described below. In the Dispatcher Window, locate the train that you wish to view, and click the mouse on it until the block representing it turns red; then click on the button `Show in game` in the Dispatcher Window and then return to the Open Rails window.

- <Alt+F9> is an enhancement of the <Alt+9> feature that displays the *Train List window* showing the names of all of the currently active trains. Click on the name of the desired train to move the exterior views to the selected train. In Activity mode, double-clicking on a train's name in this window transfers the Cabview and control of the selected train to the player. In Timetable mode, only the exterior views are selected.
- Key <9> resets the target train for the Key 2,3,4,5 and 6 views to the Player train.

Holding the <Shift> key with any motion command speeds up the movement, while holding the <Ctrl> key slows it.

Note that view direction control using the mouse with right button pressed differs slightly from using <Alt> + the arrow keys – the view direction can pass through the zenith or nadir, and the direction of vertical motion is then reversed. Passing back through the zenith or nadir restores normal behavior.

Whenever frame rates fall to unacceptable levels players are advised to adjust camera positions to cull some models from being in view and to adjust the camera again to include more models when frame rates are high.

Some camera views (among them 2Dcabview, 3Dcabview and passenger view) feature the <Ctrl+8> command, that resets the view position to the default one.

7.12 Toggling Between Windowed Mode and Full-screen

You can toggle at any time between windowed mode and full-screen by pressing <Alt+Enter>.

7.13 Modifying the Game Environment

7.13.1 Time of Day

In activity mode Open Rails software reads the StartTime from the MSTS .act file to determine what the game time is for the activity. In combination with the longitude and latitude of the route and the season, Open Rails computes the actual sun position in the sky. This provides an extremely realistic representation of the time of day selected for the activity. For example, 12 noon in the winter will have a lower sun position in the northern hemisphere than 12 noon in the summer. Open Rails game environment will accurately represent these differences.

Once the activity is started, Open Rails software allows the player to advance or reverse the environment *time of day* independently of the movement of trains. Thus, the player train may sit stationary while the time of day is moved ahead or backward. The keys to command this depend from the national settings of the keyboard, and can be derived from the key assignment list shown by pressing <F1>.

In addition, Open Rails offers functionality similar to the time acceleration switch for MSTS.

Use <Ctrl+Alt+PgUp(Numkey)> or <Ctrl+Alt+PgDn(Numkey)> keys to increase or decrease the speed of the game clock. <Ctrl+Alt+Home(Numkey)> resets the speed.

In a multiplayer session, all clients' time, weather and season selections are overridden by those set by the server.

7.13.2 Weather

When in activity mode Open Rails software determines the type of weather to display from the Weather parameter in the MSTS Activity file. In the other modes the weather can be selected in the start menu. A *Weather Change Activity Event* can be included in an activity that will modify the weather during the activity.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

7.13.3 Modifying Weather at Runtime

The following commands are available at runtime (keys not shown here can be found in the key assignment list obtained pressing F1):

- Overcast increase/decrease: increases and decreases the amount of clouds
- fog increase/decrease
- precipitation increase/decrease
- Precipitation "liquidity" (that is selection between rain and snow with intermediate states) increase/decrease.

These commands are active starting from any initial weather state (clear, rain, snow).

By selecting the desired precipitation liquidity before increasing precipitation, it is possible to decide whether to pass from clear to rain or from clear to snow weather.

Moreover, pressing `<Alt+P>` can abruptly change the weather from clear to raining to snowing and back to clear.

7.13.4 Randomized Weather in activities

By activating the related experimental option as described *here* the player may experience an initial weather that varies every time the activity is executed, and that varies in a random way during activity execution.

7.13.5 Season

In activity mode Open Rails software determines the season, and its related alternative textures to display from the Season parameter in the MSTS Activity file. In other modes the player can select the season in the start menu.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

7.14 Activity randomization

By activating the related experimental option as described *here* the player may experience slightly or significantly different activity behaviours in every different activity run. It must be stated that it is not guaranteed that every randomization leads to a realistic and/or manageable activity. However it must be considered that using features like *player train switching* and *manually setting switches and signals* many situations can be solved. This even contributes to generate a pleasant activity run.

Following activity features are randomized:

- diesel locomotive compressor blowdown: when this occurs a message is displayed, output power and force go to zero, and the smoke gets white (to have a diesel smoke colour change dieselsmoke.ace must be replaced with a better one; there is some freely available from the website of some payware trainset providers. Moreover the parameter of the third parameter line in the

Exhaust1 block within the .eng file of the diesel loco should have at least the value of 0.3, which by the way improves in general the lookout of the smoke). Wheh this event occurs, the train should be stopped as soon as possible, the defect loco should be put out of the MU chain and then switched off (these two operations can be done with the Car operations window). The defect loco is evidenced in red in the train operations window.

- diesel or electric locomotive bogie unpowered; when this occurs a message is displayed, and output power and force are halved. The defect loco is again evidenced in red in the train operations window. The total traction time is accumulated. In the first 30, 15, 10 traction minutes (for randomization levels 1, 2, 3) no locomotive failures occur. After that for each loco and at every simulator update (which has the same frequency as the FPS) a random number between 0 and 199999 is generated. If it is higher than 199998, 199992, 199899 for the three randomization levels the failure is generated. The failure may also occur on the player loco. No more than a faulty loco is possible on a train.
- freight car with brakes stuck: in this case the total braking time and the total countinuous braking time are accumulated. In this case the time with surely no failures varies from 20 to about 7 minutes for the total braking time and from 10 to about 3.5 minutes for the total continuous braking time. After such time for each car a random number between 0 and 199999 is generated at every simulator update. If the number is higher than 199996, 199992, and 199969 for the three randomization levels the failure is generated. The car will brake continuously, will be shown in red in the train operations window and will squeal if an .sms file named BrakesStuck.sms is present in the `<Train Simulator\Sound>` directory. Here an example of such file. Of course when this event occurs it is advisable to uncouple the wagon as soon as possible from the train. No more than a car will fail.

All these train failures occur only on the player train.

- AI train efficiency: the initial AI train efficiency (which determines max accelerations and decelerations and in some cases also max speed) is randomized, that is it may be increased or decreased around its preset value for a maximum of 20%, only in respectively 70% , 60% and 50% of cases when randomization level is 3, 2 or 1, and the increase and decrease is computed with a pseudonormal distribution curve, that is smaller changes are more likely than bigger changes. The same AI train efficiency randomization occurs after every station stop.
- station depart time: in the same 70% , 60% and 50% of cases the number of passengers boarding at a station are increased or decreased of a random amount that depends also from randomization level. Departure time therefore may be anticipated or, more often, delayed.
- waiting point delay: in the same 70% , 60% and 50% of cases a waiting point delay is introduced, that can have a maximum value of 25 seconds for the standard WPs and 5 minutes for the absolute WPs. Such maximum values depend also from randomization level.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

7.15 Screenshot - Print Screen

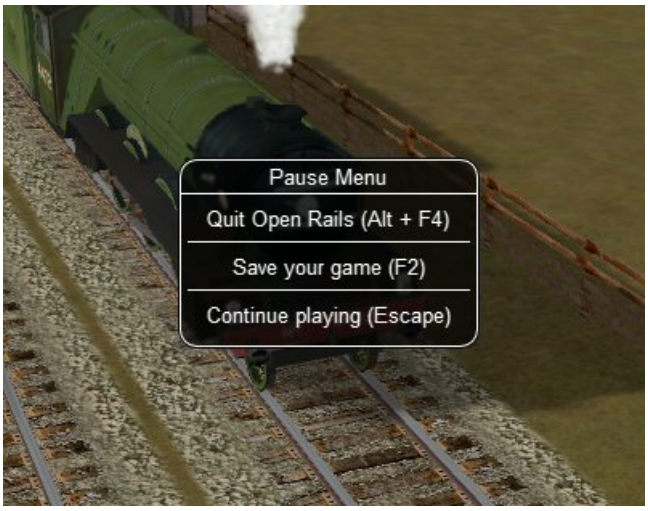
Press the keyboard `<PrintScreen>` key to capture an image of the game window. This will be saved by default in the file `C:\Users\<username>\Pictures\Open Rails\Open Rails <date and time>.png`

Although the image is taken immediately, there may be a short pause before the confirmation appears. If you hold down the Print Screen key, then OR takes multiple images as fast as it can.

The key to capture the current window – `<Alt+PrintScreen>` – is not intercepted by OR.

7.16 Suspending or Exiting the Game

You can suspend or exit the game by pressing the ESC key at any time. The window shown at the right will appear.



The window is self-explanatory.

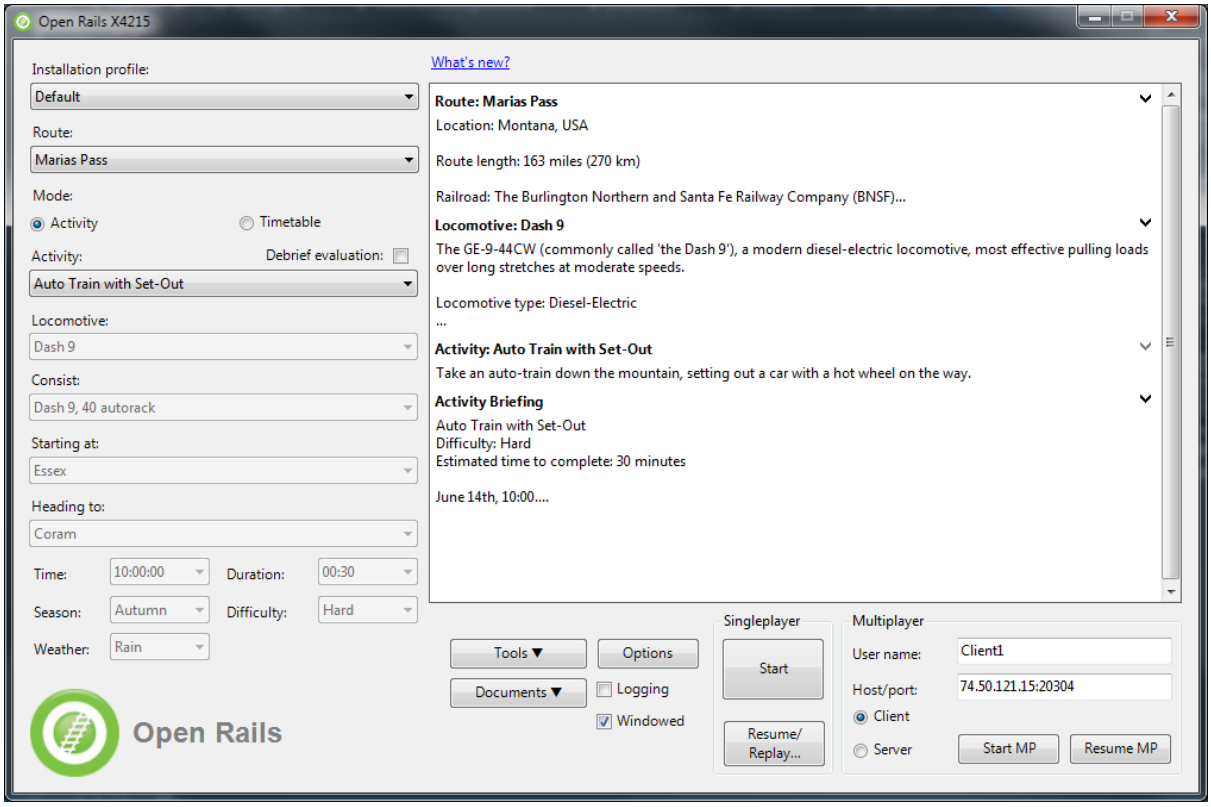
If you are running OR in a Window, you can also exit OR by simply clicking on the x on the right top of the OR window.

7.17 Save and Resume

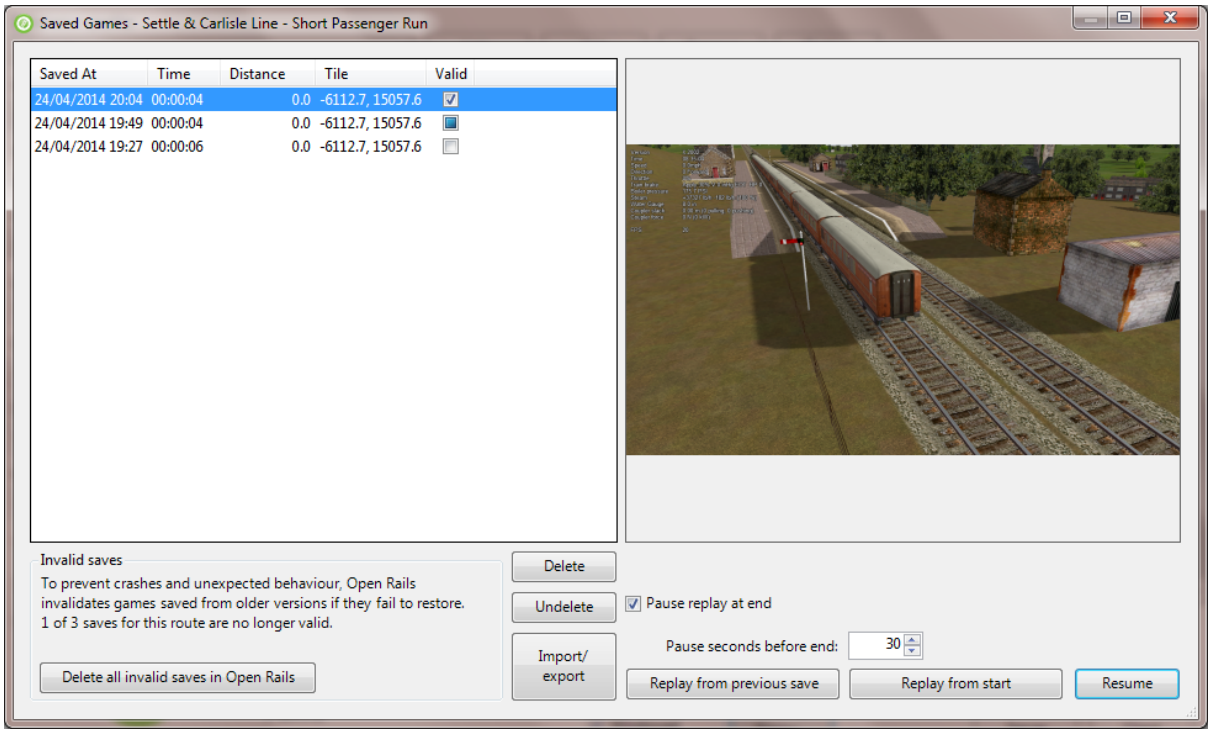
Open Rails provides Save and Resume facilities and keeps every save until you choose to delete it.

During the game you can save your session at any time by pressing <F2>.

You can view the saved sessions by choosing an activity and then pressing the `Resume/Replay...` button.



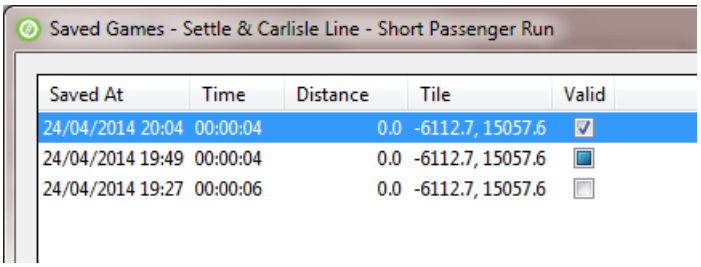
This will display the list of any Saves you made for this activity:



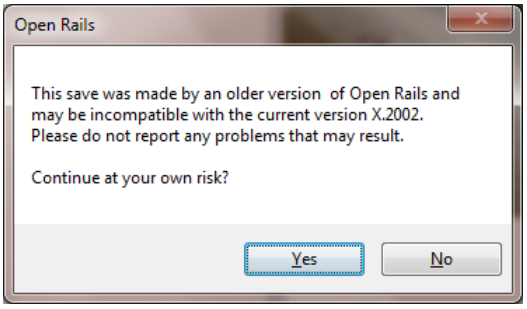
To help you identify a Save, the list provides a screenshot and date and also distance travelled in meters and the time and position of the player's train. This window can be widened to show the full width of the strings in the left panel.

7.17.1 Saves from Previous OR Versions

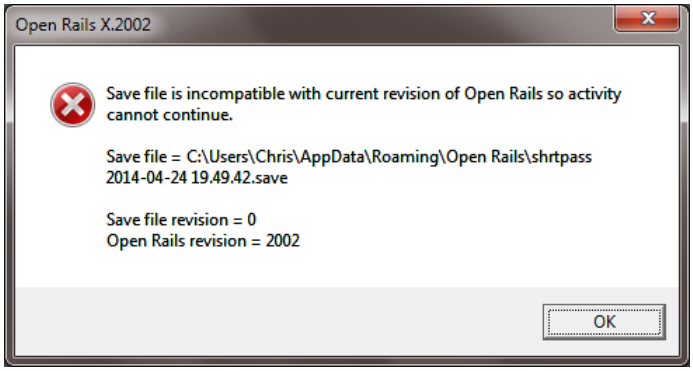
You should be aware that these Saves will only be useful in the short term as each new version of Open Rails will mark Saves from previous versions as potentially invalid (e.g. the second entry in the list below).



When you resume from such a Save, there will be a warning prompt.



The Save will be tested during the loading process. If a problem is detected, then you will be notified.



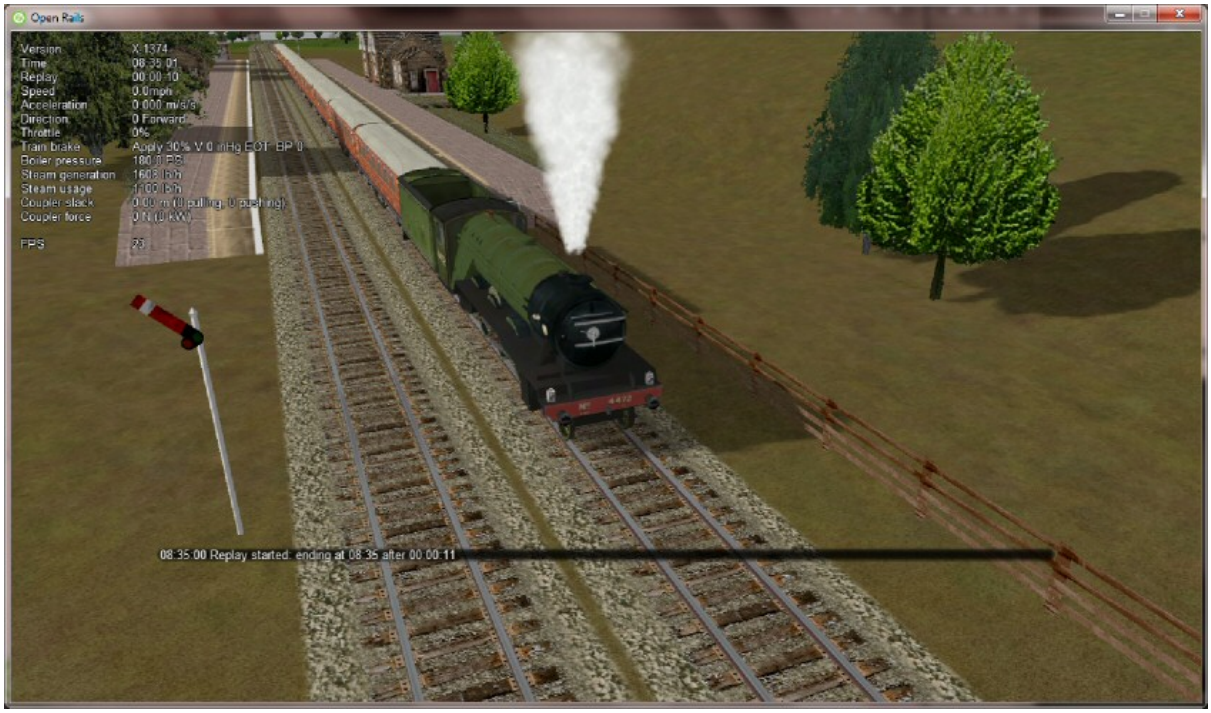
This Save and any Saves of the same age or older will be of no further value and will be marked as invalid automatically (e.g. the 3rd entry in the list). The button in the bottom left corner of the menu deletes all the invalid Saves for all activities in Open Rails.

7.18 Save and Replay

As well as resuming from a Save, you can also replay it just like a video. All the adjustments you made to the controls (e.g. opening the throttle) are repeated at the right moment to re-create the activity. As well as train controls, changes to the cameras are also repeated.

Just like a *black box flight recorder*, Open Rails is permanently in recording mode, so you can save a recording at any time just by pressing <F2> Save.

Normally, you would choose the replay option by `Menu` > `Resume` > `Replay` from start.



A second option `Menu` > `Resume` > `Replay` from previous save lets you play back a shortened recording. It resumes from the most recent Save it can find and replays from that point onwards. You might use it to play back a 5 minute segment which starts an hour into an activity.

A warning is given when the replay starts and a replay countdown appears in the Alt+F5 Head Up Display.

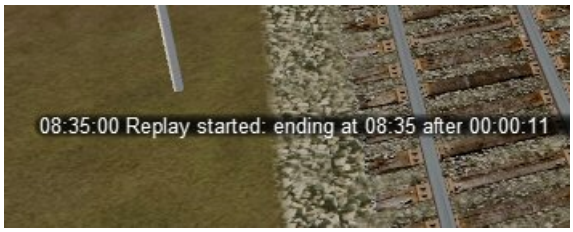


Fig. 1: Warning

By default, the simulation pauses when the replay is exhausted. Use Pause replay at end on the Saved Games window to change this.

Little can usefully be achieved by adjusting the train controls during replay, but the camera controls can be freely adjusted. If changes are made (e.g. switching to a different camera view or zooming out), then replay of the camera controls is suspended while replay of the train controls continues. The result is a bit like editing a video. To resume the replay of the camera controls, just press Esc to open the Pause Menu and then choose Continue playing.

A possible future development may be to edit the replay file to adjust times or to add messages to provide a commentary. This would allow you to build demonstrations and tutorials.

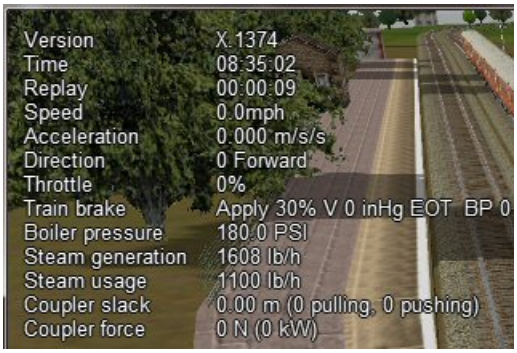
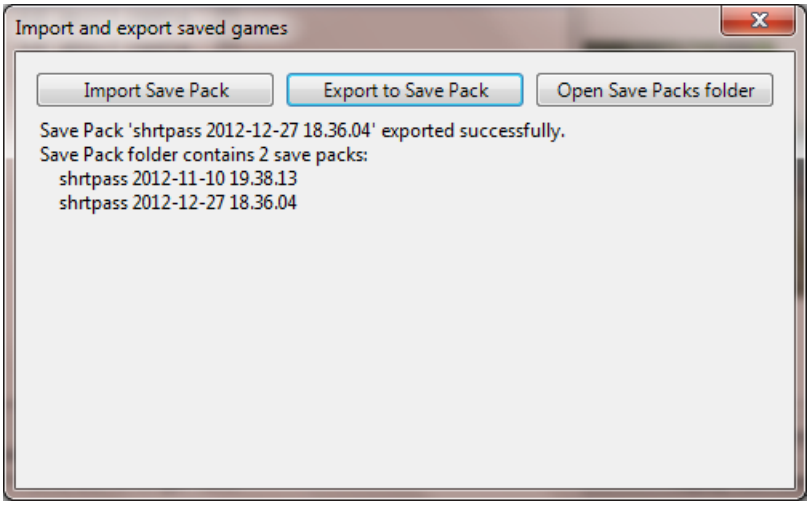


Fig. 2: Countdown

Replay is a feature which is unique to Open Rails. You can use it to make your own recordings and Open Rails provides a way to exchange them with other players.

7.18.1 Exporting and Importing Save Files

To export a Save file, use the command: `Menu` > `Options` > `Resume` > `Import/export saves` > `Export to Save Pack`



OR will pack the necessary files into a single archive file with the extension `ORSavePack` and place it in the folder `Open Rails\Save Packs`.

This `ORSavePack` file is a zip archive which contains the replay commands, a screenshot at the moment of saving, a Save file (so that Open Rails can offer its Resume option) and a log file. This arrangement means that the `ORSavePack` archive is ideal for attaching to a bug report.

You can use the `Import Save Pack` button on the same window to import and unpack a set of files from an ORSavePack archive. They will then appear in your `Saved Games` window.

7.19 Analysis Tools

The extended HUDs provide a rich amount of information for analysis, evaluation and to assist in troubleshooting.

You can move through the sequence of HUD displays by repeatedly pressing `<Shift+Alt+F5>`.

In the extended HUDs the trainsets (locos and cars) are identified by the trainset UiD as defined in the consist file, preceded by a train identification.

7.19.1 Extended HUD for Consist Information

This page shows in the first line data about the whole train. Under `Player` you will find the player locomotive UiD followed by an `F` if the forward cab is selected, and an `R` if the rear cab is selected.



`Tilted` is set at YES in case the consist name ends with `tilted` (e.g. `ETR460_tilted.con`), in which case it means that it is a tilting train.

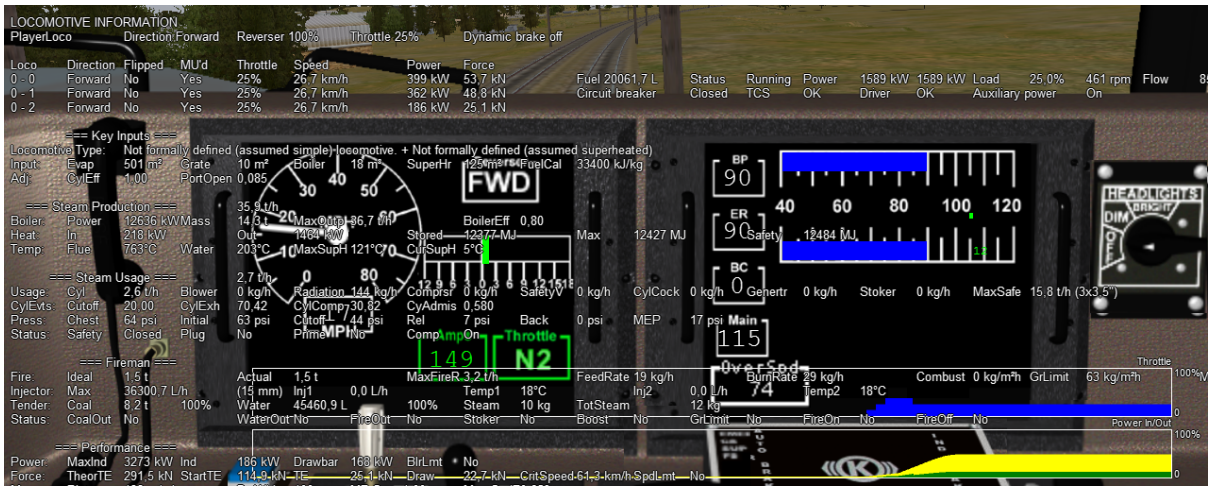
`Control mode` shows the actual control mode. Read more about this here.

Cab aspect shows the aspect of next signal.

In the other lines data about the train cars are shown. Data are mostly self-explanatory. Under `Drv/Cabs` a `D` appears if the car is drivable, and an `F` and/or a `R` appear if the car has a front and/or a rear cab.

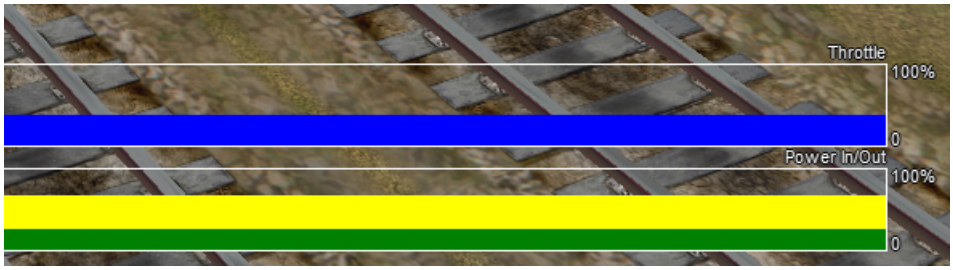
7.19.2 Extended HUD for Locomotive Information

The next extended HUD display shows locomotive information.

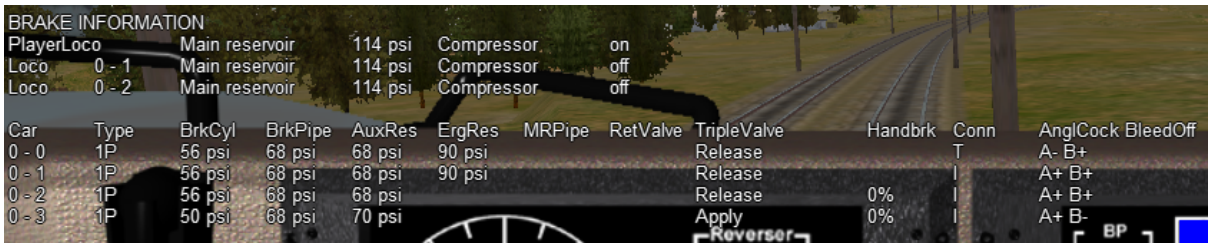


As can be seen from this screenshot related to a fictitious train with a diesel, an electric and a steam loco, information about diesel and electric locomotives is contained on a single line, while information about steam locomotives includes a large set of parameters, which shows the sophistication of OR's steam physics.

In the bottom part of this HUD two moving graphs show the evolution in time of the throttle value and of the power of the player locomotive (the one where the active cab resides).



7.19.3 Extended HUD for Brake Information



This extended HUD display includes all the information of the basic HUD plus Brake status information. In the first part specific information for locomotives is shown, while in the second one general information

is shown for all cars. After the car UiD the following alphanumeric string shows the brake system (`1P`: single-pipe system, `V`: vacuum etc.) and the current state of the air brakes on the unit. More information on this display can be found in *Open Rails Braking* and *F9 Train Operations Monitor*.

7.19.4 Extended HUD for Train Force Information

In the top part of this display some information related to the player locomotive is shown. The information format differs if *advanced adhesion* has been selected or not in the *Simulation Options*.

The middle line of information shown (provided that Wind Dependent Resistance is selected in the Options Menu) shows the wind speed and direction, the train direction, and the resulting train/wind vectors for speed and direction.

The table part below the above information, shows the relevant forces acting upon the locos/cars in the train.

The columns are as follows:

Car - the UiD of the car as defined in the car consist file.

Total - the total force acting on the car. This is the sum of the other forces after the signs are properly adjusted.

Motive - the motive force which should only be non-zero for locomotives, and that becomes negative during dynamic braking.

Brake - the brake force.

Friction - the friction (or resistance) force calculated from the Davis equation. This is in STILL air only.

Gravity - the force due to gravity.

Curve - the resistance forces due to the car being on a curve.

Tunnel - the resistance forces due to the car being in a tunnel.

Wind - the resistance forces due to the car being impacted by wind.

Coupler - the coupler force between this car and the next (negative is pull and positive is push). The **F** or **R** symbols indicate whether the coupler is a flexible or rigid coupler respectively.

Slack - indicates the amount of slack (distance due to coupler movement between the cars).

Mass - car mass in kg.

Gradient - gradient of the track underneath the car.

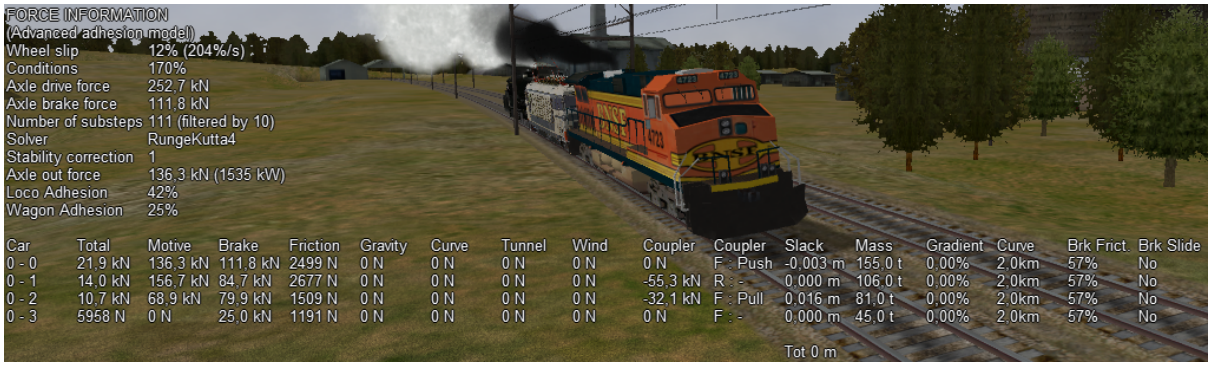
Curve - the radius of the curve.

Brk Frict - friction of the brakes on the car.

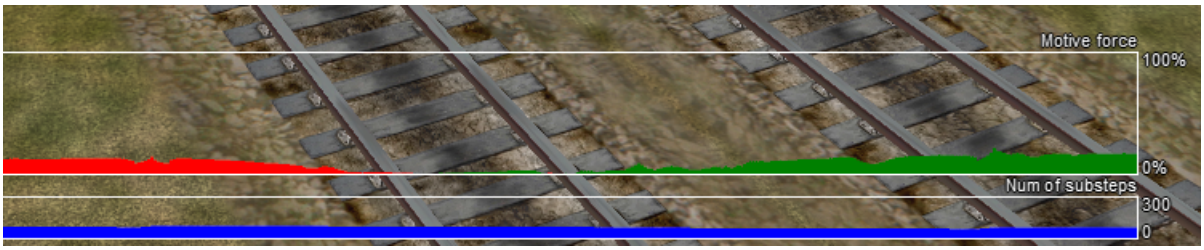
Brk Slide - indicates whether the car is skidding due to excessive brake application.

All of the force values will be in Newtons, or the UoM selected by the player.

Many of these values are relative to the orientation of the car, but some are relative to the train. If applicable, two further fields appear: the first is “True” if the car is flipped with respect to the train or `False` otherwise, while the second field signals coupler overload.



At the bottom of the picture two moving graphs are displayed.

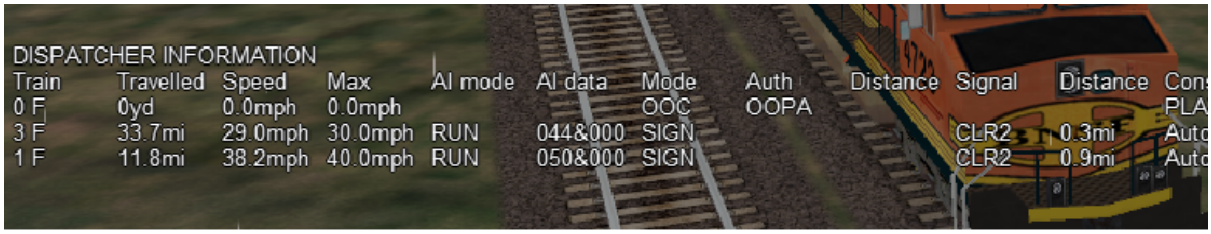


The upper graph displays the motive force in % of the player locomotive. Green colour means tractive force, red colour means dynamic brake force.

The lower graph refers – roughly speaking - to the level of refinement used to compute axle force.

7.19.5 Extended HUD for Dispatcher Information

The next extended HUD displays Dispatcher Information. It is very useful to troubleshoot activities or timetables. The player train and any AI trains will show in the Dispatcher Information, a line for each train.



A detailed explanation of the various columns follows:

- Train: Internal train number, with P=Passenger and F=Freight.
- Travelled: distance travelled. Gives an indication if all is well. If a train started an hour ago and 'travelled' is still 0.0, something's clearly wrong.
- Speed: present speed.
- Max: maximum allowed speed.
- AI Mode: gives an indication of what the AI train is 'doing'. Possible states:
  - INI: train is initializing. Normally you would not see this.
  - STP: train is stopped other than in a station. The reason for the stop is shown in `Authority`.
  - BRK: train is preparing to stop. Does not mean it is actually braking, but it 'knows' it has to stop, or at least reduce speed, soon. Reason and distance to the related position, are shown in `Authority` and `Distance`.
  - ACC: train is accelerating, either away from a stop or because of a raise in allowed speed.
  - RUN: train is running at allowed speed.

  - FOL: train is following another train in the same signal section. Its speed is now derived from the speed of the train ahead.
  - STA: train is stopped in station.
  - WTP: train is stopped at waiting point.
  - EOP: train is approaching end of path.
  - STC: train is Static train, or train is in Inactive mode if waiting for next action.
- AI data : shows throttle (first three digits) and brake (last three digits) positions when AI train is running, but shows departure time (booked) when train is stopped at station or waiting point, or shows activation time when train is in inactive mode (state STC).
- Mode:
  - SIGN (signal)
  - NODE
  - MAN: train is in manual mode (only player train, see *here*)
  - OOC: train is out of control
  - EXPL: train is in explorer mode (only player train) When relevant, this field also shows delay (in minutes), e.g. S+05 mean Signal mode, 5 minutes delay.
- Auth: End of "authorization" info - that is, the reason why the train is preparing to stop or slow down. Possible reasons are :
  - SPDL: speed limit imposed by speed sign.
  - SIGL: speed limit imposed by signal.
  - STOP: signal set at state "STOP".
  - REST: signal set at state "RESTRICTED" (train is to reduce speed at approaching this signal).
  - EOA: end of authority - generally only occurs in non-signaled routes or area, where authority is based on NODE mode and not SIGNAL mode.
  - STAT: station.
  - TRAH: train ahead.
  - EOR: end of train's route, or subroute in case the train approaches a reversal point.
  - AUX: all other authorization types, including auxiliary action authorizations (e.g. waiting points).

  When the control mode is `NODE` the column `Auth` can show following strings:
  - EOT: end of track
  - EOP: end of path
  - RSW: switch reserved by another train
  - LP: train is in loop
  - TAH: train ahead
  - MXD: free run for at least 5000 meters
  - NOP: no path reserved.

  When the control mode is `OOC` the column `Auth` can show following strings:
  - SPAD: passed signal at danger
  - RSPD: passed signal at danger running backwards
  - OOAU: passed authority limit

  - OOPA: out of path
  - SLPP: slipped into path
  - SLPT: slipped to end of track
  - OOTR: out of track
  - MASW: misaligned switch.
- Distance: distance to the authority location.
- Signal: aspect of next signal (if any).
- Distance: distance to this signal. Note that if signal state is STOP, and it is the next authority limit, there is a difference of about 30m between authority and signal distance. This is the 'safety margin' that AI trains keep to avoid accidentally passing a signal at danger.
- Consist: the first part of the train's service name. Only for the player, always the PLAYER string is displayed.
- Path: the state of the train's path. The figure left of the "=" sign is the train's present subpath counter : a train's path is split into subpaths when its path contains reversal points. The details between { and } are the actual subpath. Following the final } can be x<N>, this indicates that at the end of this subpath the train will move on to the subpath number N. Path details :
  - The path shows all track circuit sections which build this train's path. Track circuit sections are bounded by nodes, signals or cross-overs, or end-of-track. Each section is indicated by its type:
    * – is plain train section.
    * > is switch (no distinction is made for facing or trailing switch).
    * + is crossover.
    * [is end-of-track.
  - Following each section is the section state. Numbers in this state refer to the train numbers as shown at the start of each row. Below, <n> indicates such a number.
- <n> section is occupied by train <n>.
- (<n>) section is reserved for train <n>.
- # (either with <n> or on its own) section is claimed by a train which is waiting for a signal.
- & (always in combination with <n>) section is occupied by more than one train.
- **deadlock info (always linked to a switch node):**
  - * possible deadlock location - start of a single track section shared with a train running in opposite direction.
  - ^ active deadlock - train from opposite direction is occupying or has reserved at least part of the common single track section. Train will be stopped at this location – generally at the last signal ahead of this node.
  - ~ active deadlock at that location for other train - can be significant as this other train can block this train's path.

The dispatcher works by reserving track vector nodes for each train. An AI train will be allowed to move (or start) only if all of the nodes up to the next potential passing location are not reserved for another train. If this condition cannot be met, in Timetable Mode the AI train will not spawn.

There are other reasons why an AI train might not appear in Timetable Mode. The current dispatcher assumes that all routes are unsignaled. The dispatcher issues a track authority (which is similar to a track warrant) to all trains. For an AI train to start, the tracks it needs must not be already reserved for another train. The dispatcher compares the paths of the trains to identify possible passing points and then reserves tracks for a train up until a passing point. When a train gets near the next passing point the reservation is extended to the next one. The end result is that in Timetable Mode an AI train cannot be placed on a

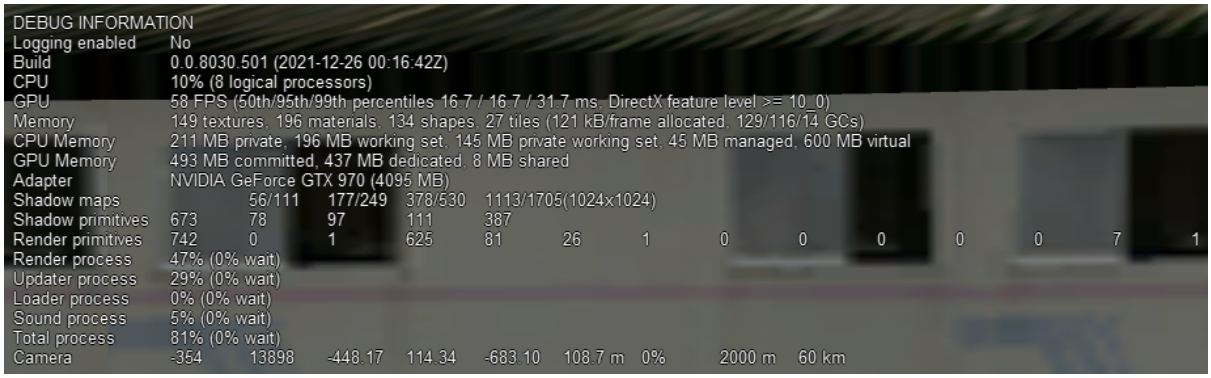
track if that section of track is already occupied by or reserved for another train. A section of track is any track bounded by either a switch or a signal.

Also, a train is not created if it would be partly or fully superimposed on an already existing train, or if its path is not long enough for it. This applies to both Timetable Mode and Activity Mode.

7.19.6 Extended HUD for Debug Information

The last extended HUD display shows Debug information containing:

- Logging enabled: *logging status*
- Build: date and time Open Rails was compiled
- CPU: processor utilization by Open Rails
- GPU: frame rate, frame time percentiles and graphics feature level
- Memory: number of core memory objects loaded (textures, materials, shapes, and tiles) plus garbage collection statistics
- CPU Memory:
  - Private: virtual memory allocated just for Open Rails (not shared with other applications)
  - Working set: physical memory in use by Open Rails (including any shared with other applications)
  - Private working set: physical memory in use just for Open Rails
  - Managed: virtual memory allocated by the .NET runtime (CLR)
  - Virtual: virtual memory allocated for any purpose (private + shared + others)
- GPU Memory:
  - Committed: all graphics memory allocated for Open Rails
  - Dedicated: physical graphics card memory in use by Open Rails
  - Shared: system memory shared with the graphics card in use by Open Rails
- Adapter: name and dedicated physical memory of graphics card
- Shadow maps: distance and size of each shadow map level (and texture size)
- Shadow primitives: total primitives (rendered items) and breakdown by shadow map level
- Render primitives: total primitives (rendered items) and breakdown by rendering sequence (`RenderPrimitiveSequence` in the code)
- Render/Updater/Loader/Sound process: percentage of time each process is active and waiting
- Camera: tile X, tile Z, X, Y, Z, altitude, LOD bias, effective viewing distance, effective distant mountain viewing distance

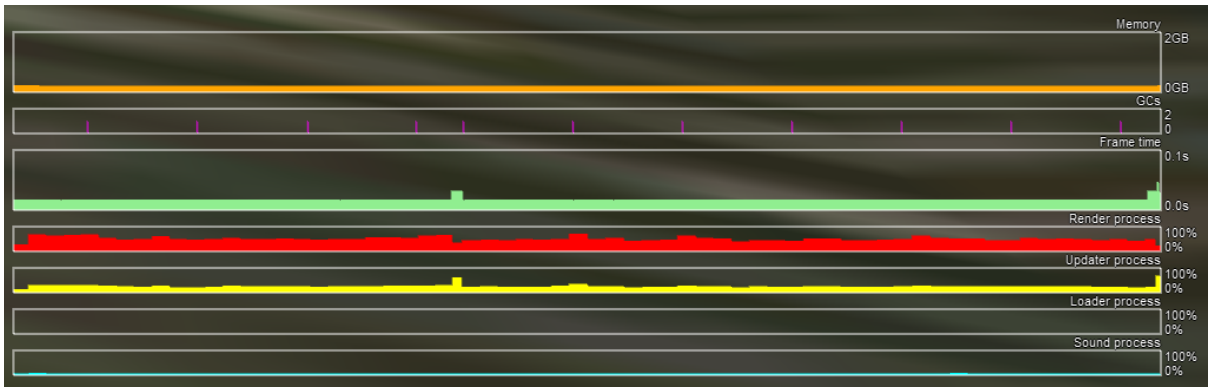


The primary measurements for comparing and analysing performance are the first column of values (some lines are skipped), so for the image above we have:

- CPU: 10% - *very low (warning at 75%)*
- GPU: 58 FPS - *good (warning at <55 FPS with 60 Hz vertical sync)*
- CPU Memory: 211 MB private - *very low (~3% of 8 GB)*
- GPU Memory: 493 MB committed - *low (~24% of 2 GB)*
- Highest process (Render): 47% - *moderate (warning at 75%)*

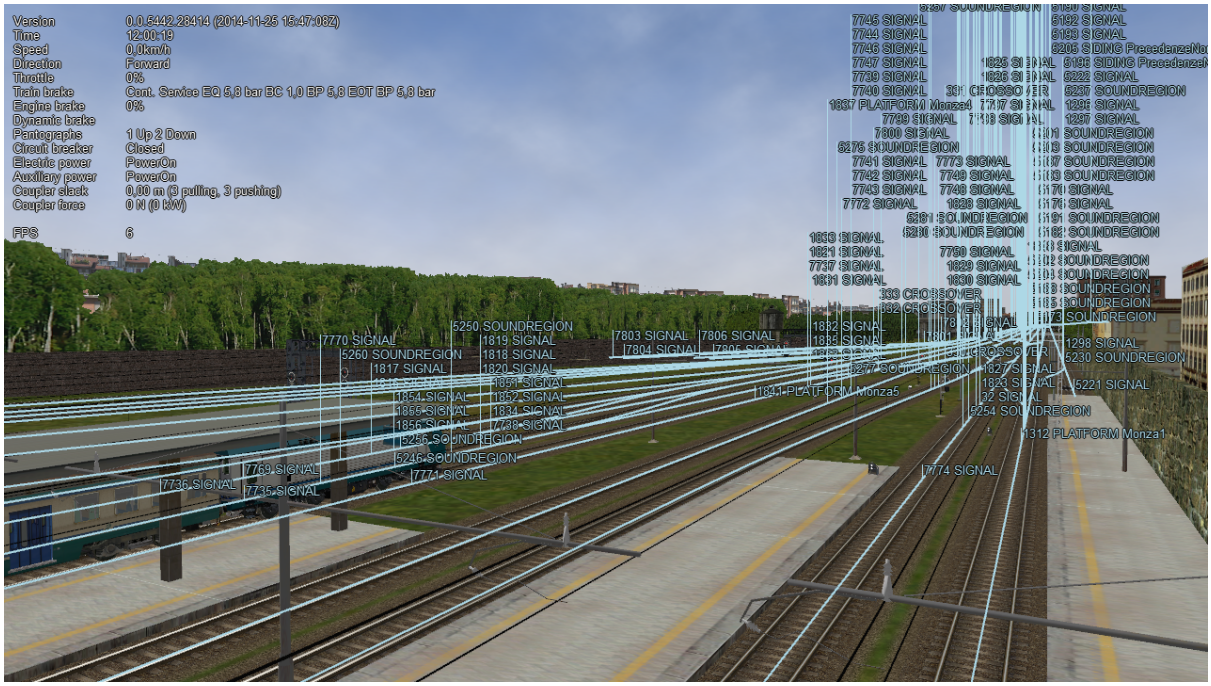
Also shown are the following graphs of:

- Memory: working set
- GCs: garbage collections
- Frame time: how long each frame takes to render
- Render/Updater/Loader/Sound process: same as textual display above



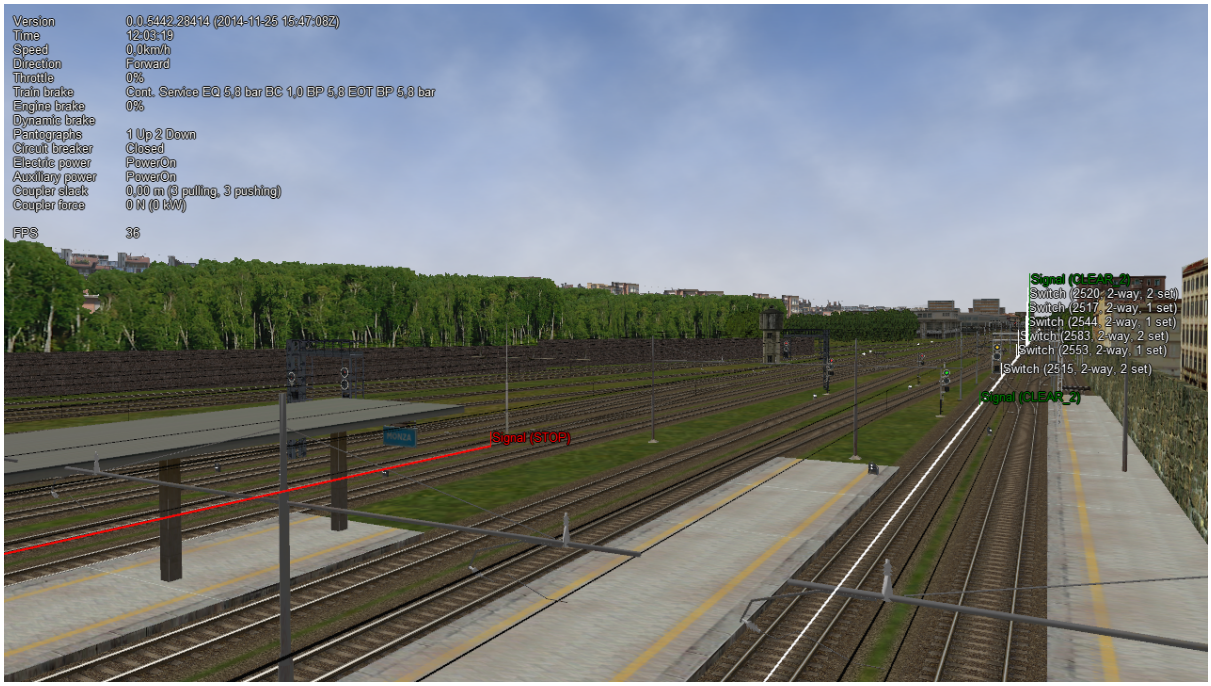
7.19.7 Viewing Interactive Track Items

By pressing `<Ctrl+Alt+F6>` at runtime you get a picture like this one that allows you to take note of the interactive IDs for debugging purposes.



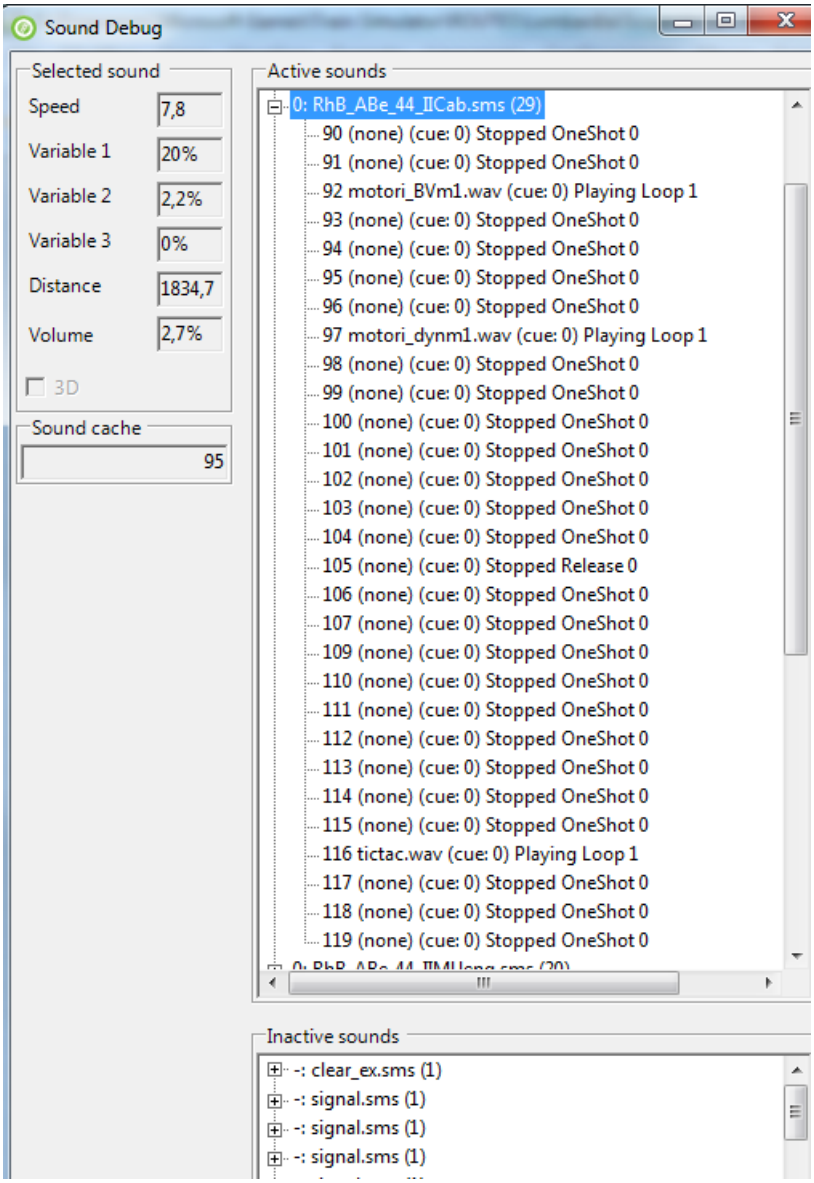
7.19.8 Viewing Signal State and Switches

By pressing <Ctrl+Alt+F11> you get a picture like the following that shows the state of the signals and switches on the path.



7.19.9 Sound Debug Window

By pressing <Alt+S> this window opens:



It shows in the upper part the list of all active .sms files (track sound apart); by expanding the detail of a specific .sms file, the list of all sound streams is displayed, as well as their state. On the left the value of the analog sound variables is displayed for the selected .sms file. The volume refers to the first stream of the selected sound file.

Active and inactive sounds toggle passing from internal to external views and vice-versa.

7.20 OpenRailsLog.txt Log file

When the `Logging` option in the main window is checked, a log file named OpenRailsLog.txt file is generated. This file contains rich information about the execution of the game session, allowing identification of critical problems. This file should always be attached to requests of support in case of problems.

The contents of the file are often self-explanatory, and therefore can be evaluated by the same contents developer. It includes reports of various errors in the MSTS files which are ignored by OR, including missing sound files, unrecognized terms in some files, etc. Selecting the Experimental Option *Show shape warnings* allows OR to report errors found in shape files in the log file. It includes also reports about malfunctions in the gaming session, such as trains passing red signals, as well as OR malfunctions.

7.21 Code-embedded Logging Options

OR source code is freely downloadable; check the http://www.OpenRails.org website for this. Within the code there are some debug options that, when activated, generate specific extended log files, e.g. for analysis of signal and of AI train behavior. Short specific info on this can be provided to people with programming skills.

7.22 Testing in Autopilot Mode

Autopilot mode is a powerful tool to help in testing activities.

CHAPTER 8

Open Rails Physics

Open Rails physics is in an advanced stage of development. The physics structure is divided into logical classes; more generic classes are parent classes, more specialized classes inherit properties and methods of their parent class. Therefore, the description for train cars physics is also valid for locomotives (because a locomotive is a special case of a train car). All parameters are defined within the .wag or .eng file. The definition is based on MSTS file format and some additional ORTS based parameters. To avoid possible conflicts in MSTS, the *ORTS* prefix is added to every OpenRails specific parameter (such as `ORTSMaxTractiveForceCurves`).

The .wag or .eng file may be placed as in MSTS in the `TRAINS\TRAINSET\TrainCar\` folder (where TrainCar is the name of the train car folder). If OR-specific parameters are used, or if different .wag or .eng files are used for MSTS and OR, the preferred solution is to place the OR-specific .wag or .eng file in a created folder `TRAINS\TRAINSET\TrainCar\OpenRails\` (see *here* for more).

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

8.1 Train Cars (WAG, or Wagon Part of ENG file)

The behavior of a train car is mainly defined by a resistance / resistive force (a force needed to pull a car). Train car physics also includes coupler slack and braking. In the description below, the Wagon section of the WAG / ENG file is discussed.

8.1.1 Resistive Forces

Open Rails physics calculates resistance based on real world physics: gravity, mass, rolling resistance and optionally curve resistance. This is calculated individually for each car in the train. The program calculates rolling resistance, or friction, based on the Friction parameters in the Wagon section of .wag/.eng file. Open Rails identifies whether the .wag file uses the *FCalc* utility or other friction data. If *FCalc* was used to determine the Friction variables within the .wag file, Open Rails compares that data to the Open Rails Davis equations to identify the closest match with the Open Rails Davis equation. If no-FCalc Friction parameters are used in the .wag file, Open Rails ignores those values, substituting its actual Davis equation values for the train car.

A basic (simplified) Davis formula is used in the following form:

F_{res} = ORTSDavis_A + speedMpS * (ORTSDavis_B + ORTSDavis_C * speedMpS^2)

Where F_{res} is the friction force of the car. The rolling resistance can be defined either by *FCalc* or ORTS-Davis_A, _B and _C components. If one of the *ORTSDavis* components is zero, *FCalc* is used. Therefore, e.g. if the data doesn't contain the B part of the Davis formula, a very small number should be used instead of zero.

When a train is initially started, additional force is needed to overcome the initial higher bearing torque (forces) and track resistance. Starting resistance is calculated automatically by Open Rails based upon empirical prototypical data at low speeds. By selecting different values for `ORTSBearingType` different values of starting resistance will be applied. The Open Rails calculation for starting resistance takes into account different conditions, such as weather (for example, snowing or clear), wagon (axle) load, wheel bearing temperature and wheel diameter. Hence when using the OR calculation the correct values should be inserted in `ORTSNumberAxles` parameter in the wagon section, and `ORTSNumberDriveAxles` in the engine section. The `WheelRadius` value should also be inserted in both sections as appropriate.

Alternatively the low-speed friction force can be manually specified by the user by setting `ORTSStandstillFriction` and `ORTSMergeSpeed`.

When running on a curve and if the *Curve dependent resistance* option is enabled, additional resistance is calculated, based on the curve radius, rigid wheel base, track gauge and super elevation. The curve resistance has its lowest value at the curve's optimal speed. Running at higher or lower speed causes higher curve resistance. The worst situation is starting a train from zero speed. The track gauge value can be set by `ORTSTrackGauge` parameter, otherwise 1435 mm is used. The rigid wheel base can be also set by `ORTSRigidWheelBase`, otherwise the value is estimated. Further details are discussed later.

When running on a slope (uphill or downhill), additional resistance is calculated based on the car mass taking into account the elevation of the car itself. Interaction with the `car vibration feature` is a known issue (if the car vibrates the resistance value oscillate).

8.1.2 Coupler Slack

Slack action for couplers is introduced and calculated the same way as in MSTS.

8.1.3 Hot Wheel Bearings

Open Rails (OR) has instead used a representative bearing heat model to simulate the typical outcomes for bearing temperature heating or cooling effects.

- Bearing heats up and cools down as the train moves and stops.
- Bearing resistance in cold weather is significantly higher then when the bearing is at its 'normal' operating temperature. Typically railway companies elected to reduce loads for trains in cold conditions. The OR model will reduce the car resistance as the bearing heats up, and it will increase resistance as the bearing cools down.
- OR has a built in temperature model to determine the ambient temperature. The ambient temperature is calculated based upon a world model of the average temperatures at various latitudes. OR will use the latitude of the route to calculate the ambient temperature. As ambient temperature also decreases with height above sea level, OR takes this into account as well, and varies the temperature accordingly.
- Depending upon the `ActivityRandomizationLevel` setting in the Option menu, an overheating bearing (hotbox) may be randomly initialized on any trailing car in the train (locomotives and tenders are excepted from overheating bearings). The Hotbox will be activated randomly within the first 66% of the activity duration. So for example, in an activity with a 20 minute duration, a hotbox will only be activiated in the first 12 minutes of the activity, if it has been initialised.

A special smoke effect, `BearingHotboxFX`, can be added adjacent to the wagon hot box. This will be triggered if the bearing overheats.

8.1.4 Derailment Coefficient

The derailment coefficient indicates the likelihood that a car or wagon will derail, and is the ratio of the lateral force to vertical force acting on the wagon. This concept was first proposed by Nadal.

The higher the coefficient the higher the risk that a derailment will occur. Most railway companies tend to operate at a coefficient value of less then 0.8 as this gives a desireable safety margin for the car.

The OR calculated derailment coefficient is displayed in the Force Information HuD. The coefficient value will change colour to indicate the likelihood of the car derailing. White indicates normal operation, yellow provides a warning indication, whilst red indicates that derailment is extremely likely.

Open Rails uses some standard defaults that it uses to calculate the derailment coefficient, however if the modeler desires greater accuracy the following parameters can be added to the WAG/ENG file in the wagon section:

`ORTSLengthBogieCentre` - length between bogie centres.

`ORTSLengthCarBody` - Length between car ends (typically measured between the coupler pivot points).

`ORTSLengthCouplerFace` - length between coupler faces.

`ORTSNumberAxles` - number of axles on the car.

`ORTSNumberDriveAxles` - number of driven axles on the locomotive. NB: Total axles on locomotive will be `ORTSNumberAxles` + `ORTSNumberDriveAxles`.

`ORTSNumberBogies` - number of bogies on the car.

8.1.5 Adhesion of Locomotives – Settings Within the Wagon Section of ENG files

MSTS calculates the adhesion parameters based on a very strange set of parameters filled with an even stranger range of values. Since ORTS is not able to mimic the MSTS calculation, a standard method based on the adhesion theory is used with some known issues in use with MSTS content.

MSTS `Adheasion` (sic!) parameters are not used in ORTS. Instead, a new set of parameters is used, which must be inserted within the `Wagon` section of the .ENG file:

```
ORTSAdhesion (
    ORTSCurtius_Kniffler (A B C D )
)
```

The A, B and C values are coefficients of a standard form of various empirical formulas, e.g. Curtius-Kniffler or Kother. The D parameter is used in the advanced adhesion model described later.

From A, B and C a coefficient CK is computed, and the adhesion force limit is then calculated by multiplication of CK by the car mass and the acceleration of gravity (9.81), as better explained later.

The adhesion limit is only considered in the adhesion model of locomotives.

The adhesion model is calculated in two possible ways. The first one – the simple adhesion model – is based on a very simple threshold condition and works similarly to the MSTS adhesion model. The second one – the advanced adhesion model – is a dynamic model simulating the real world conditions on a wheel-to-rail contact and will be described later. The advanced adhesion model uses some additional parameters such as:

```
ORTSAdhesion (
    ORTSSlipWarningThreshold ( T )
)
```

where T is the wheelslip percentage considered as a warning value to be displayed to the driver; and:

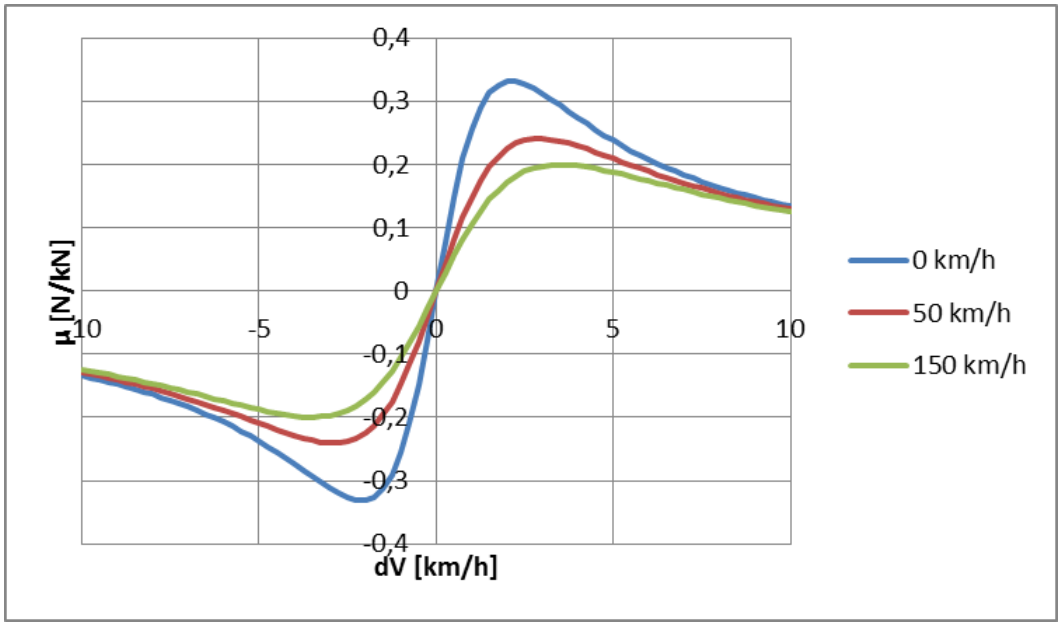
```
ORTSAdhesion(
    Wheelset (
        Axle (
            ORTSInertia (
                Inertia
            )
        )
    )
)
```

where Inertia is the model inertia in kg.m2 and can be set to adjust the advanced adhesion model dynamics. The value considers the inertia of all the axles and traction drives. If not set, the value is estimated from the locomotive mass and maximal power.

The first model – simple adhesion model – is a simple tractive force condition-based computation. If the tractive force reaches its actual maximum, the wheel slip is indicated in HUD view and the tractive force falls to 10% of the previous value. By reducing the throttle setting adherence is regained. This is called the simple adhesion model.

The second adhesion model (advanced adhesion model) is based on a simplified dynamic adhesion theory. Very briefly, there is always some speed difference between the wheel speed of the locomotive and the longitudinal train speed when the tractive force is different from zero. This difference is called *wheel slip / wheel creep*. The adhesion status is indicated in the HUD *Force Information* view by the *Wheel Slip* parameter and as a warning in the general area of the HUD view. For simplicity, only one axle model is computed (and animated). A tilting feature and the independent axle adhesion model will be introduced in the future.

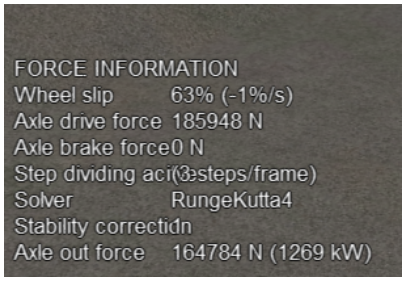
The heart of the model is the slip characteristics (picture below).

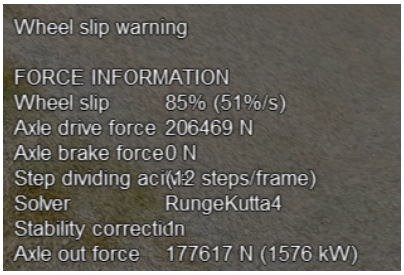


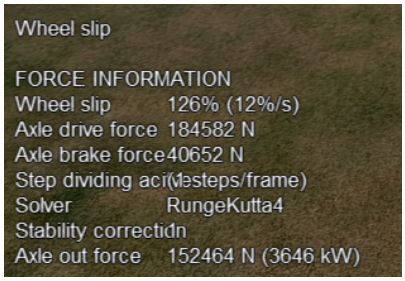
The *wheel creep* describes the stable area of the characteristics and is used in the most of the operation time. When the tractive force reaches the actual maximum of the slip characteristics, force transition falls down and more power is used to speed up the wheels, so called *wheel slip*.

To avoid the loss of the tractive force, use the throttle in combination with sanding to return to the stable area (wheel creep area). A possible sequence of the wheel slip development is shown on the pictures below. The *Wheel slip* value is displayed as a value relative to the best adhesion conditions for actual speed and weather. The value of 63% means very good force transition. For values higher than `( ORTSadhesion ( ORTSSlipWarningThreshold ) )` or 70% by default, the *Wheel slip* warning is displayed, but the force transition is still very good. This indication should warn you to use the throttle very carefully. Exceeding 100%, the *Wheel slip* message is displayed and the wheels are starting to speed up, which can be seen

on the speedometer or in external view 2. To reduce the wheel slip, use *throttle down*, sanding or the locomotive brake.







The *actual maximum* of the tractive force is based on the Curtius-Kniffler adhesion theory and can be adjusted by the aforementioned `ORTSCurtius_Kniffler ( A B C D )` parameters, where A, B, C are coefficients of Curtius-Kniffler, Kother or similar formula. By default, Curtius-Kniffler is used.

$$F_{adhMAX} = W \cdot m\,[\mathrm{kg}] \cdot 9.81\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] \cdot \left(\frac{A}{B + v\left[\frac{\mathrm{km}}{\mathrm{h}}\right]} + C\right)$$

Where `W` is the weather coefficient. This means that the maximum is related to the speed of the train, or to the weather conditions.

The `D` parameter is used in an advanced adhesion model and should always be 0.7.

There are some additional parameters in the *Force Information* HUD view. The axle/wheel is driven by the *Axle drive force* and braked by the *Axle brake force*. The *Axle out force* is the output force of the adhesion model (used to pull the train). To compute the model correctly the FPS rate needs to be divided by a *Solver dividing* value in a range from 1 to 50. By default, the Runge-Kutta4 solver is used to obtain the best results.

In some cases when the CPU load is high, the time step for the computation may become very high and the simulation may start to oscillate (the *Wheel slip* rate of change (in the brackets) becomes very high). You can use the `DebugResetWheelSlip` (`<Ctrl+X>` keys by default) command to reset the adhesion model. If you experience such behavior most of time, use the basic adhesion model instead by pressing `DebugToggleAdvancedAdhesion` (`<Ctrl+Alt+X>` keys by default).

To match some of the real world features, the *Wheel slip* event can cause automatic zero throttle setting. Use the `Engine (ORTS (ORTSWheelSlipCausesThrottleDown))` Boolean value of the ENG file.

Modern locomotives have slip control systems which automatically adjust power, providing an optimal tractive effort avoiding wheel slip. The `ORTSSlipControlSystem ( Full )` parameter can be inserted into the engine section of the .eng file to indicate the presence of such system.

8.2 Engine – Classes of Motive Power

Open Rails software provides for different classes of engines: diesel, electric, steam, control and default. If needed, additional classes can be created with unique performance characteristics.

8.2.1 Diesel Locomotives

Diesel Locomotives in General

The diesel locomotive model in ORTS simulates the behavior of two basic types of diesel engine driven locomotives– diesel-electric and diesel-mechanical. The diesel engine model is the same for both types, but acts differently because of the different type of load. Basic controls (direction, throttle, and brakes) are common across all classes of engines. Diesel engines can be started or stopped by pressing the START/STOP key (`<Shift+Y>` in English keyboards). The starting and stopping sequence is driven by a *starter* logic, which can be customized, or is estimated by the engine parameters.

The diesel electric locomotive uses a diesel prime mover to generate electricity (using generators naturally) and this electricity is then used to drive traction motors to turn the wheels. The other types of diesel locomotives are similar from the perspective that they have a diesel prime mover, and then some form of transmission mechanism to transfer the power output of the prime mover to the locomotive wheels.

In configuring the locomitve correctly it is important to use the correct power/force values. The key values required in the ENG file for a diesel locomotive (regardless of transmission type) are as follows:

`ORTSDieselEngineMaxPower ==>` sets the maximum power output at the shaft of the diesel engine (or prime mover).

`MaxPower ==>` sets the maximum power at the rail (provided to the wheels).

`MaxForce ==>` sets the force that the locomotive is able to apply to the wheels when starting.

`MaxContinuousForce ==>` is the maximum force that the locomotive can continuously supply to the wheels without exceeding the design specifications. Typically this is linked to a particular speed (see next parameter).

`ORTSSpeedOfMaxContinuousForce ==>` is the speed at which the maximum force will be applied.

`MaxVelocity ==>` is the maximum rated design speed of the locomotive. Some locomotives had a speed alarm which applied the brakes, or set the throttle to a lower value. This can be modelled using the OverspeedMonitor function.

`ORTSUnloadingSpeed ==>` is the locomotive speed when the generator reaches its maximum voltage, and due to the speed of the train, the engine starts to 'unload'. Typically beyond this speed, power output of the locomotive will decrease.

If using power/force Tables, then some of the above values will not be required, see the sections below for details.

Starting the Diesel Engine

To start the engine, simply press the START/STOP key once. The direction controller must be in the neutral position (otherwise, a warning message pops up). The engine RPM (revolutions per minute) will increase according to its speed curve parameters (described later). When the RPM reaches 90% of StartingRPM (67% of IdleRPM by default), the fuel starts to flow and the exhaust emission starts as well. RPM continues to increase up to StartingConfirmationRPM (110% of IdleRPM by default) and the demanded RPM is set to idle. The engine is now started and ready to operate.

Stopping the Diesel Engine

To stop the engine, press the START/STOP key once. The direction controller must be in the neutral position (otherwise, a warning message pops up). The fuel flow is cut off and the RPM will start to decrease according to its speed curve parameters. The engine is considered as fully stopped when RPM is zero. The engine can be restarted even while it is stopping (RPM is not zero).

Starting or Stopping Helper Diesel Engines

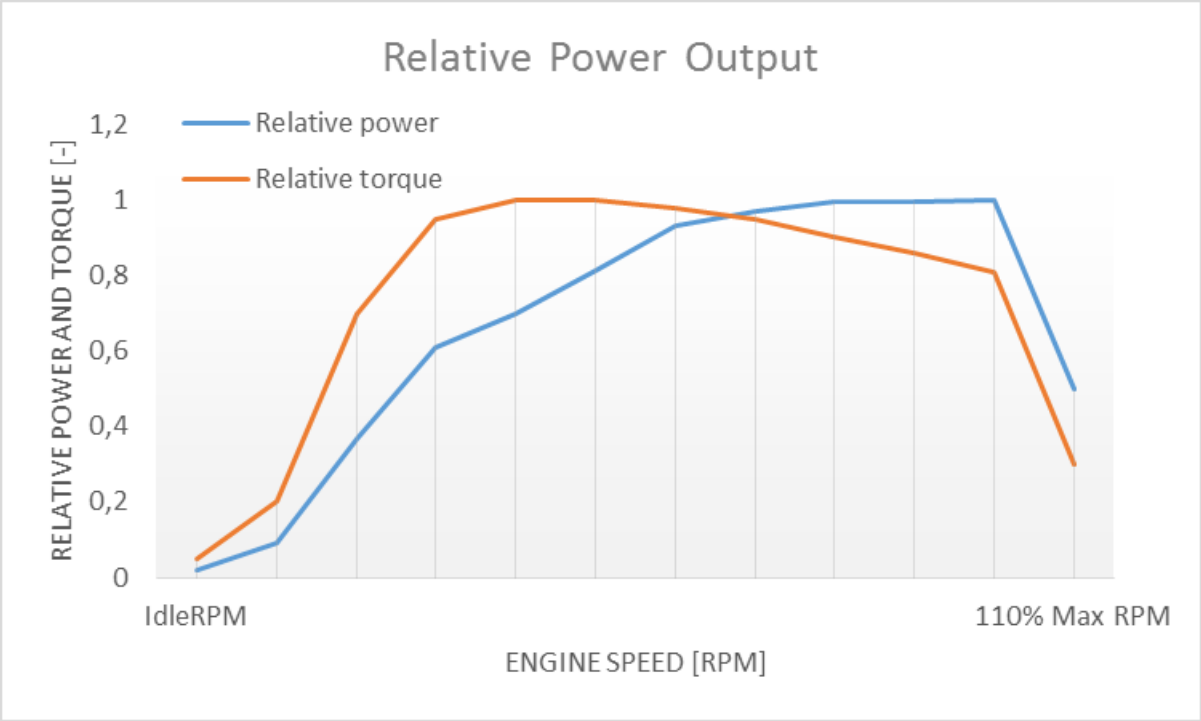
By pressing the Diesel helper START/STOP key (`<Ctrl+Y>` on English keyboards), the diesel engines of helper locomotives can be started or stopped. Also consider disconnecting the unit from the multiple-unit (MU) signals instead of stopping the engine (see *here*, *Toggle MU connection*).

It is also possible to operate a locomotive with the own engine off and the helper's engine on.

ORTS Specific Diesel Engine Definition

If no ORTS specific definition is found, a single diesel engine definition is created based on the MSTS settings. Since MSTS introduces a model without any data crosscheck, the behavior of MSTS and ORTS diesel locomotives can be very different. In MSTS, MaxPower is not considered in the same way and you can get much *better* performance than expected. In ORTS, diesel engines cannot be overloaded.

No matter which engine definition is used, the diesel engine is defined by its load characteristics (maximum output power vs. speed) for optimal fuel flow and/or mechanical characteristics (output torque vs. speed) for maximum fuel flow. The model computes output power / torque according to these characteristics and the throttle settings. If the characteristics are not defined (as they are in the example below), they are calculated based on the MSTS data and common normalized characteristics.



In many cases the throttle vs. speed curve is customized because power vs. speed is not linear. A default linear throttle vs. speed characteristics is built in to avoid engine overloading at lower throttle settings. Nevertheless, it is recommended to adjust the table below to get more realistic behavior.

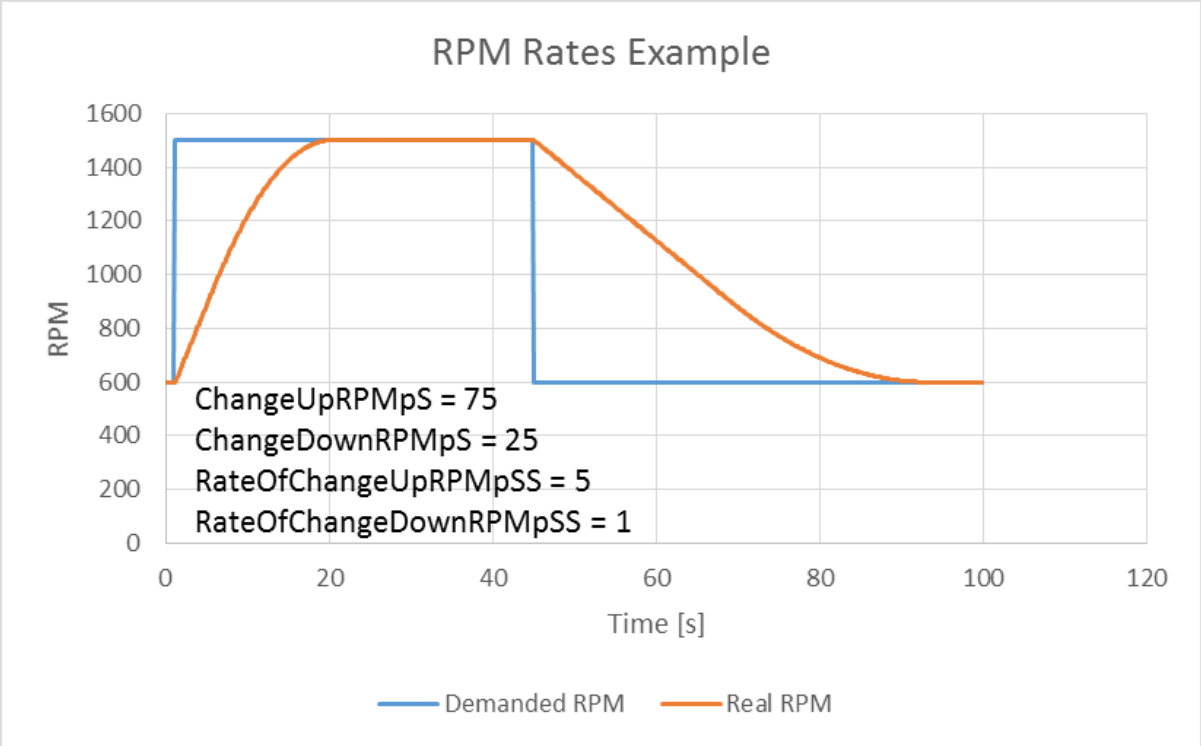
In ORTS, single or multiple engines can be set for one locomotive. In case there is more than one engine, other engines act like *helper* engines (start/stop control for helpers is `<Ctrl+Y>` by default). The power of each active engine is added to the locomotive power. The number of such diesel engines is not limited.

If the ORTS specific definition is used, each parameter is tracked and if one is missing (except in the case of those marked with *Optional*), the simulation falls back to use MSTS parameters.

```
Engine(                                          Engine section in eng file
...
ORTSDieselEngines ( 2                            Number of engines
  Diesel (
    IdleRPM ( 510 )                              Idle RPM
    MaxRPM ( 1250 )                              Maximal RPM
    StartingRPM ( 400 )                          Starting RPM
    StartingConfirmRPM ( 570 )                   Starting confirmation RPM
    ChangeUpRPMpS ( 50 )                         Increasing change rate RPM/s
    ChangeDownRPMpS ( 20 )                       Decreasing change rate RPM/s
    RateOfChangeUpRPMpSS ( 5 )                   Jerk of ChangeUpRPMpS RPM/s^2
    RateOfChangeDownRPMpSS ( 5 )                 Jerk of ChangeDownRPMpS RPM/s^2
    MaximalPower ( 300kW )                       Maximal output power
    IdleExhaust ( 5 )                            Num of exhaust particles at IdleRPM
    MaxExhaust ( 50 )                            Num of exhaust particles at MaxRPM
    ExhaustDynamics ( 10 )                       Exhaust particle mult. at transient
    ExhaustDynamicsDown (10)                     Mult. for down transient (Optional)
    ExhaustColor ( 00 fe )                       Exhaust color at steady state
    ExhaustTransientColor(                       Exhaust color at RPM changing
        00 00 00 00)
    DieselPowerTab (                             Diesel engine power table
        0       0                                   RPM        Power in Watts
        510     2000
        520     5000
        600     2000
        800     70000
        1000    100000
        1100    200000
        1200    280000
        1250    300000
    )
    DieselConsumptionTab (                       Diesel fuel consumption table
        0       0                                 RPM   Vs consumption l/h/rpm
        510     10
        1250    245
    )
    ThrottleRPMTab (                             Eengine RPM vs. throttle table
        0   510                                     Throttle %      Demanded RPM
        5   520
        10  600
        20  700
        50  1000
        75  1200
        100 1250
    )
    DieselTorqueTab (                            Diesel engine RPM vs. torque table
        0       0                                   RPM           Force in Newtons
        510     25000
        1250    200000
    )
    MinOilPressure ( 40 )                        Min oil pressure PSI
    MaxOilPressure ( 90 )                        Max oil pressure PSI
    MaxTemperature ( 120 )                       Maximal temperature Celsius
    Cooling ( 3 )                                Cooling 0=No cooling, 1=Mechanical,
                                                 2= Hysteresis, 3=Proportional
    TempTimeConstant ( 720 )                     Rate of temperature change
    OptTemperature ( 90 )                        Normal temperature Celsius
    IdleTemperature ( 70 )                       Idle temperature Celsius
  Diesel ( ... )                                 The same as above, or different
```

Diesel Engine Speed Behavior

The engine speed is calculated based on the RPM rate of change and its rate of change. The usual setting and the corresponding result is shown below. `ChangeUpRPMpS` means the slope of RPM, `RateOfChangeUpRPMpSS` means how fast the RPM approaches the demanded RPM.



Fuel Consumption

Following the MSTS model, ORTS computes the diesel engine fuel consumption based on .eng file parameters. The fuel flow and level are indicated by the HUD view. Final fuel consumption is adjusted according to the current diesel power output (load).

Diesel Exhaust

The diesel engine exhaust feature can be modified as needed. The main idea of this feature is based on the general combustion engine exhaust. When operating in a steady state, the color of the exhaust is given by the new ENG parameter `engine (ORTS (Diesel (ExhaustColor)))`.

The amount of particles emitted is given by a linear interpolation of the values of `engine(ORTS (Diesel (IdleExhaust)))` and `engine(ORTS (Diesel (MaxExhaust)))` in the range from 1 to 50. In a transient state, the amount of the fuel increases but the combustion is not optimal. Thus, the quantity of particles is temporarily higher: e.g. multiplied by the value of

`engine(ORTS (Diesel (ExhaustDynamics)))` and displayed with the color given by `engine(ORTS(Diesel(ExhaustTransientColor)))`.

The format of the *color* value is (aarrggbb) where:

- aa = intensity of light;
- rr = red color component;
- gg = green color component;
- bb = blue color component;

and each component is in HEX number format (00 to ff).

Cooling System

ORTS introduces a simple cooling and oil system within the diesel engine model. The engine temperature is based on the output power and the cooling system output. A maximum value of 100°C can be reached with no impact on performance. It is just an indicator, but the impact on the engine's performance will be implemented later. The oil pressure feature is simplified and the value is proportional to the RPM. There will be further improvements of the system later.

Diesel-Electric Locomotives

Diesel-electric locomotives are driven by electric traction motors supplied by a diesel-generator set. The gen-set is the only power source available, thus the diesel engine power also supplies auxiliaries and other loads. Therefore, the output power will always be lower than the diesel engine rated power.

In ORTS, the diesel-electric locomotive can use `ORTSTractionCharacteristics` or tables of `ORTSMaxTractiveForceCurves` to provide a better approximation to real world performance. If a table is not used, the tractive force is limited by MaxForce, MaxPower and MaxVelocity. The throttle setting is passed to the ThrottleRPMTab, where the RPM demand is selected. The output force increases with the Throttle setting, but the power follows maximal output power available (RPM dependent).

Diesel-Hydraulic Locomotives

Diesel-hydraulic locomotives are not implemented in ORTS. However, by using either `ORTSTractionCharacteristics` or `ORTSMaxTractiveForceCurves` tables, the desired performance can be achieved, when no gearbox is in use and the `DieselEngineType` is *electric*.

Diesel-Mechanical Locomotives

ORTS features a mechanical gearbox feature that mimics MSTS behavior, including automatic or manual shifting. Some features not well described in MSTS are not yet implemented, such as `GearBoxBackLoadForce`, `GearBoxCoastingForce` and `GearBoxEngineBraking`.

Output performance is very different compared with MSTS. The output force is computed using the diesel engine torque characteristics to get results that are more precise.

To indicate that the diesel is a mechanical transmission, `ORTSDieselTransmissionType` needs to be set to "Mechanic".

Two ORTS mechanical gearbox configurations can be set up.

These two gearboxes can be selected by the use of the following parameter:

`ORTSGearBoxType ( A )` - represents a semi-automatic pre-selector gearbox that gives a continuous power output that is not interrupted when changing gears.

`ORTSGearBoxType ( B )` - represents a semi-automatic pre-selector type gear box where although there is a break in tractive effort when changing from one gear to another, the engine speed is reduced by a shaft brake if needed, so that there is no need for the driver to adjust the throttle.

One of three possible types of main clutch are selectable for each of the above gear box types, as follows:

`ORTSMainClutchType ( Friction )` - represents a mechanical friction clutch.

`ORTSMainClutchType ( Fluid )` - represents a fluid coupling. Where a transmission includes both a friction clutch and a fluid coupling then ORTSMainClutchType ("Fluid") should be used in the eng file.

`ORTSMainClutchType ( Scoop )` - represents a fluid coupling that includes a scoop device to disconnect the engine from the transmission at idle speed.

`ORTSGearBoxFreeWheel` - indicates whether a freewheel mechanism is included in the transmission. (0) - should be used for transmissions that do not include a freewheel. This option will allow 'engine braking' to occur when appropriate. (1) - should be used for transmissions that include a freewheel. This option will allow the train to coast with the engine in gear.

`GearBoxNumberOfGears` - The number of gears available in the gear box.

Currently only a BASIC model configuration is available (ie no user defined traction curves or diesel engine curves are supported). OR calculates the tractive force curves for each gear based on the "inbuilt" torque curve of a typical diesel engine.

`GearBoxMaxSpeedForGears` - sets the maximum speed for each gear, corresponding to maximum engine rpm and maximum power . As an example, the values for a typical British Railways first generation dmu are:

GearBoxMaxSpeedForGears(15.3 27 41 65.5) The default values are in mph, although other units can be entered. In the above case the maximum permitted speed of the train is 70 mph; a small amount of 'overspeed' being allowed

> in top gear. The fourth gear speed of 65.5 mph corresponds to the maximum engine rpm set in the eng file by `DieselEngineMaxRPM`. The diesel engine may continue to 'runaway' above its normal 'maximum speed' until it reaches the maximum governed speed or 'redline' speed at which the engine governor will cut off the fuel supply until the engine speed is reduced. This speed can be set in basic Open Rails eng files using `ORTSDieselEngineGovenorRpM`. In the case of the above train, then these would be

DieselEngineMaxRPM(1800) ORTSDieselEngineGovenorRpM (2000)

If under any circumstances the engine reaches `ORTSDieselEngineGovenorRpM` then the diesel engine will automatically be shut down.

"ORTSGearBoxTractiveForceAtSpeed" - The tractive force available in each gear at the speed indicated in GearBoxMaxSpeedForGears. Units by default are in N, however lbf, N or kN. Published values for tractive effort of geared locomotives and multiple units are generally those at the maximum speed for each gear.

Hence a typical gear configuration for a diesel mechanic locomotive might look like the following:

> ORTSDieselTransmissionType (Mechanic)
>
> ORTSGearBoxType (B) ORTSMainClutchType ("Friction") ORTSGearBoxFreeWheel (0)
>
> GearBoxOperation(Manual) GearBoxNumberOfGears(6) GearBoxMaxSpeedForGears(4.5mph 6mph 9mph 14.5mph 21mph 33mph) ORTSGearBoxTractiveForceatSpeed(35400lbf 26600lbf 17700lbf 11200lbf 7600lbf 4830lbf)

Traction cut-off relay

The traction cut-off relay of all locomotives in a consist can be controlled by *Control Traction Cut-Off Relay Closing Order*, *Control Traction Cut-Off Relay Opening Order* and *Control Traction Cut-Off Relay Closing Authorization* commands (`<O>`, `<I>` and `<Shift+O>` by default). The status of the traction cut-off relay is indicated by the *Traction cut-off relay* value in the HUD view.

The traction cut-off relay is also opened if the *Train Control System* triggers an emergency braking.

Two default behaviours are available:

- By default, the traction cut-off relay of the train closes as soon as power is available on the engines.
- The traction cut-off relay can also be controlled manually by the driver. To get this behaviour, put the parameter `ORTSTractionCutOffRelay( Manual )` in the Engine section of the ENG file.

In order to model a different behaviour of the traction cut-off relay, a *scripting interface* is available. The script can be loaded with the parameter `ORTSTractionCutOffRelay( <name of the file> )`.

In real life, the traction cut-off relay does not close instantly, so you can add a delay with the optional parameter `ORTSTractionCutOffRelayClosingDelay( )` (by default in seconds).

Power supply

The power status is indicated by the *Power* value in the HUD view.

The power-on sequence time delay can be adjusted by the optional `ORTSPowerOnDelay( )` value (for example: `ORTSPowerOnDelay( 5s )`) within the Engine section of the .eng file (value in seconds). The same delay for auxiliary systems can be adjusted by the optional parameter `ORTSAuxPowerOnDelay( )` (by default in seconds).

A *scripting interface* to customize the behavior of the power supply is also available.

8.2.2 Electric Locomotives

At the present time, diesel and electric locomotive physics calculations use the default engine physics. Default engine physics simply uses the MaxPower and MaxForce parameters to determine the pulling power of the engine, modified by the Reverser and Throttle positions. The locomotive physics can be replaced by traction characteristics (speed in mps vs. force in Newtons) as described below.

Some OR-specific parameters are available in order to improve the realism of the electric system.

Pantographs

The pantographs of all locomotives in a consist are triggered by *Control Pantograph First* and *Control Pantograph Second* commands (`<P>` and `<Shift+P>` by default). The status of the pantographs is indicated by the *Pantographs* value in the HUD view.

Since the simulator does not know whether the pantograph in the 3D model is up or down, you can set some additional parameters in order to add a delay between the time when the command to raise the pantograph is given and when the pantograph is actually up.

In order to do this, you can write in the Wagon section of your .eng file or .wag file (since the pantograph may be on a wagon) this optional structure:

```
ORTSPantographs(
    Pantograph(         << This is going to be your first pantograph.
        Delay( 5s )     << Example : a delay of 5 seconds
    )
    Pantograph(
        ... parameters for the second pantograph ...
    )
)
```

Other parameters will be added to this structure later, such as power limitations or speed restrictions.

3rd and 4th Pantograph

Open Rails supports up to 4 pantographs per locomotive. If three or four pantographs are present, the above ORTSPantographs() block is mandatory, and must contain a number of Pantograph() blocks equal to the number of pantographs in the locomotive. The animation names of the 3rd and 4th pantograph follow the same rules valid for Pantograph 2 (replacing 2 with 3 and 4). The third panto is moved with Ctrl-P, while the fourth panto is moved with Ctrl-Shift-P. The cabview controls must be named ORTS_PANTOGRAPH3 and ORTS_PANTOGRAPH4.

Circuit breaker

The circuit breaker of all locomotives in a consist can be controlled by *Control Circuit Breaker Closing Order*, *Control Circuit Breaker Opening Order* and *Control Circuit Breaker Closing Authorization* commands (<O>, <I> and <Shift+O> by default). The status of the circuit breaker is indicated by the *Circuit breaker* value in the HUD view.

The circuit breaker is also opened if the *Train Control System* triggers an emergency braking.

Two default behaviours are available:

- By default, the circuit breaker of the train closes as soon as power is available on the pantograph.
- The circuit breaker can also be controlled manually by the driver. To get this behaviour, put the parameter `ORTSCircuitBreaker( Manual )` in the Engine section of the ENG file.

In order to model a different behaviour of the circuit breaker, a *scripting interface* is available. The script can be loaded with the parameter `ORTSCircuitBreaker( <name of the file> )`.

In real life, the circuit breaker does not close instantly, so you can add a delay with the optional parameter `ORTSCircuitBreakerClosingDelay( )` (by default in seconds).

Power supply

The power status is indicated by the *Power* value in the HUD view.

The power-on sequence time delay can be adjusted by the optional `ORTSPowerOnDelay( )` value (for example: `ORTSPowerOnDelay( 5s )`) within the Engine section of the .eng file (value in seconds). The same delay for auxiliary systems can be adjusted by the optional parameter `ORTSAuxPowerOnDelay( )` (by default in seconds).

A *scripting interface* to customize the behavior of the power supply is also available.

Traction motor type

There are different types of electric motors: series DC motors, asynchronous/synchronous AC motors, etc. Currently a simple AC induction motor has been implemented, and can be selected with the ``ORTSTractionMotorType ( AC ) `` parameter, to be inserted in the Engine section of the ENG file. The use of this motor will have an impact on wheel slip, because the wheel speed never exceeds the frequency of the rotating magnetic field.

8.2.3 Steam Locomotives

General Introduction to Steam Locomotives

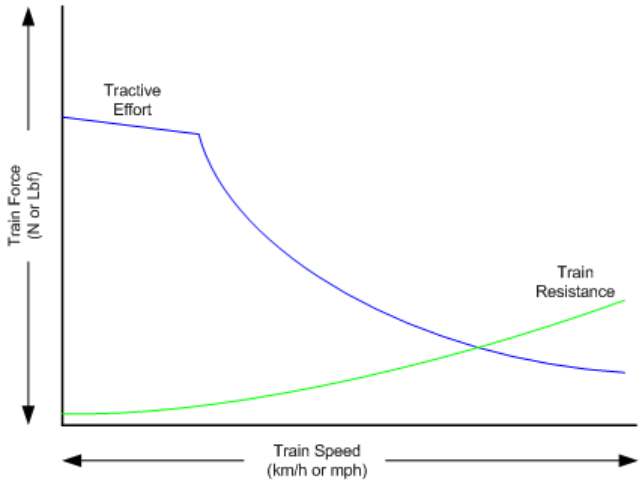
Principles of Train Movement

Key Points to Remember:

- Steam locomotive tractive effort must be greater than the train resistance forces.
- Train resistance is impacted by the train itself, curves, gradients, tunnels, etc.
- Tractive effort reduces with speed, and will reach a point where it *equals* the train resistance, and thus the train will not be able to go any faster.
- This point will vary as the train resistance varies due to changing track conditions.
- Theoretical tractive effort is determined by the boiler pressure, cylinder size, drive wheel diameters, and will vary between locomotives.
- Low Factors of Adhesion will cause the locomotive's driving wheels to slip.

Forces Impacting Train Movement

The steam locomotive is a heat engine which converts *heat* energy generated through the burning of fuel, such as coal, into heat and ultimately steam. The steam is then used to do *work* by injecting the steam into the cylinders to drive the wheels around and move the locomotive forward. To understand how a train will move forward, it is necessary to understand the principal mechanical forces acting on the train. The diagram below shows the two key forces affecting the ability of a train to move.



The first force is the tractive effort produced by the locomotive, whilst the second force is the resistance presented by the train. Whenever the tractive effort is greater than the train resistance the train will continue to move forward; once the resistance exceeds the tractive effort, then the train will start to slow down, and eventually will stop moving forward.

The sections below describe in more detail the forces of tractive effort and train resistance.

Train Resistance

The movement of the train is opposed by a number of different forces which are collectively grouped together to form the *train resistance*.

The main resistive forces are as follows (the first two values of resistance are modelled through the Davis formulas, and only apply on straight level track):

- Journal or Bearing resistance (or friction)
- Air resistance
- Gradient resistance – trains travelling up hills will experience greater resistive forces then those operating on level track.
- *Curve resistance* – applies when the train is traveling around a curve, and will be impacted by the curve radius, speed, and fixed wheel base of the rolling stock.
- *Tunnel resistance* – applies when a train is travelling through a tunnel.

Tractive Effort

Tractive Effort is created by the action of the steam against the pistons, which, through the media of rods, crossheads, etc., cause the wheels to revolve and the engine to advance.

Tractive Effort is a function of mean effective pressure of the steam cylinder and is expressed by following formula for a simple locomotive. Geared and compound locomotives will have slightly different formula:

```
TE = Cyl/2 x (M.E.P. x d2 x s) / D
```

Where:

- Cyl = number of cylinders
- TE = Tractive Effort (lbf)
- M.E.P. = mean effective pressure of cylinder (psi)
- D = diameter of cylinder (in)
- S = stroke of cylinder piston (in)
- D = diameter of drive wheels (in)

Theoretical Tractive Effort

To allow the comparison of different locomotives, as well as determining their relative pulling ability, a theoretical approximate value of tractive effort is calculated using the boiler gauge pressure and includes a factor to reduce the value of M.E.P.

Thus our formula from above becomes:

```
TE = Cyl/2 x (C x BP x d2 x s) / D
```

Where:

- BP = Boiler Pressure (gauge pressure - psi)
- C = factor to account for losses in the engine, typically values between 0.7 and 0.85 were used by different manufacturers and railway companies. Default is set @ 0.85. User can change by adding the `ORTSTractiveEffortFactor` parameter to the ENG file.

Factor of Adhesion

The factor of adhesion describes the likelihood of the locomotive slipping when force is applied to the wheels and rails, and is the ratio of the starting Tractive Effort to the weight on the driving wheels of the locomotive:

```
FoA = Wd / TE
```

Where:

- FoA = Factor of Adhesion
- TE = Tractive Effort (lbs)
- Wd = Weight on Driving Wheels (lbs)

Typically the Factor of Adhesion should ideally be between 4.0 & 5.0 for steam locomotives. Values below this range will typically result in slippage on the rail.

Indicated HorsePower (IHP)

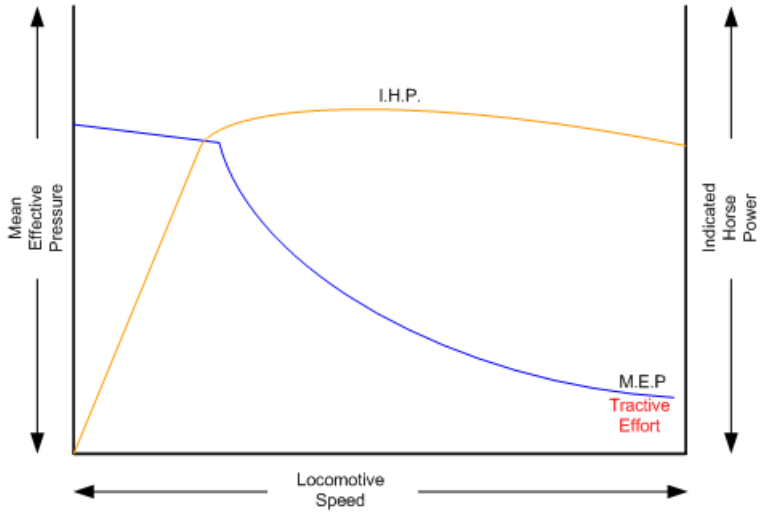
Indicated Horsepower is the theoretical power produced by a steam locomotive. The generally accepted formula for Indicated Horsepower is:

```
I.H.P. = Cyl/2 x (M.E.P. x L x A x N) / 33000
```

Where:

- IHP = Indicated Horsepower (hp)
- Cyl = number of cylinders
- M.E.P. = mean effective pressure of cylinder (psi)
- L = stroke of cylinder piston (ft)
- A = area of cylinder (sq in)
- N = number of cylinder piston strokes per min (NB: two piston strokes for every wheel revolution)

As shown in the diagram below, IHP increases with speed, until it reaches a maximum value. This value is determined by the cylinder's ability to maintain an efficient throughput of steam, as well as for the boiler's ability to maintain sufficient steam generation to match the steam usage by the cylinders.



Hauling Capacity of Locomotives

Thus it can be seen that the hauling capacity is determined by the summation of the tractive effort and the train resistance.

Different locomotives were designed to produce different values of tractive effort, and therefore the loads that they were able to haul would be determined by the track conditions, principally the ruling gradient for the section, and the load or train weight. Therefore most railway companies and locomotive manufacturers developed load tables for the different locomotives depending upon their theoretical tractive efforts.

The table below is a sample showing the hauling capacity of an American (4-4-0) locomotive from the Baldwin Locomotive Company catalogue, listing the relative loads on level track and other grades as the cylinder size, drive wheel diameter, and weight of the locomotive is varied.

CODE WORD	Class	Cylinders Diam. Stroke Inches	Diameter Driving Wheels Inches	Boiler Pressure Pounds per Square Inch	Rated Tractive Force Pounds	Weight in Working Order Pounds		Wheel Base		Capacity Tender for Water 8½ lb. gallons	Load in Tons (2000 Pounds) of Cars and Lading							
						On all Driving Wheels	Total	Of Driving Wheels	Total		On a Level	On a Grade per Mile of						
												26.4 ft. or ½ %	52.8 ft. or 1 %	79.2 ft. or 1½ %	105.6 ft. or 2 %	158.4 ft. or 3 %	211.2 ft. or 4 %	264.0 ft. or 5 %
Matchlock.....	8-28 C	17 x 24	66	180	16,080	70,000	102,000	7′ 6″	21′ 5″	3500	1720	805	485	335	245	145	90	60
Matelas.....	8-30 C	18 x 24	66	180	18,020	77,000	115,000	7′ 6″	21′ 6″	4000	1930	900	545	375	275	165	105	65
Matelasser....	8-32 C	19 x 24	66	180	20,070	85,000	128,000	8′ 0″	22′ 11″	4500	2140	1000	605	415	305	185	115	75
Matellarum....	8-34 C	20 x 24	66	180	22,260	92,000	136,000	8′ 6″	23′ 4″	5000	2315	1085	655	450	330	200	125	80

Typically the ruling gradient is defined as the maximum uphill grade facing a train in a particular section of the route, and this grade would typically determine the maximum permissible load that the train could haul in this section. The permissible load would vary depending upon the direction of travel of the train.

Elements of Steam Locomotive Operation

A steam locomotive is a very complex piece of machinery that has many component parts, each of which will influence the performance of the locomotive in different ways. Even at the peak of its development in the middle of the 20th century, the locomotive designer had at their disposal only a series of factors and simple formulae to describe its performance. Once designed and built, the performance of the locomotive was measured and adjusted by empirical means, i.e. by testing and experimentation on the locomotive. Even locomotives within the same class could exhibit differences in performance.

A simplified description of a steam locomotive is provided below to help understand some of the key basics of its operation.

As indicated above, the steam locomotive is a heat engine which converts fuel (coal, wood, oil, etc.) to heat; this is then used to do work by driving the pistons to turn the wheels. The operation of a steam locomotive can be thought of in terms of the following broadly defined components:

- Boiler and Fire (Heat conversion)
- Cylinder (Work done)

Boiler and Fire (Heat conversion)

The amount of work that a locomotive can do will be determined by the amount of steam that can be produced (evaporated) by the boiler.

Boiler steam production is typically dependent upon the Grate Area, and the Boiler Evaporation Area.

- *Grate Area* – the amount of heat energy released by the burning of the fuel is dependent upon the size of the grate area, draught of air flowing across the grate to support fuel combustion, fuel calorific value, and the amount of fuel that can be fed to the fire (a human fireman can only shovel so much coal in an hour). Some locomotives may have had good sized grate areas, but were 'poor steamers' because they had small draught capabilities.
- *Boiler Evaporation Area* – consisted of the part of the firebox in contact with the boiler and the heat tubes running through the boiler. This area determined the amount of heat that could be transferred to the water in the boiler. As a rule of thumb a boiler could produce approximately 12-15 lbs/h of steam per ft^2 of evaporation area.
- *Boiler Superheater Area* – Typically modern steam locomotives are superheated, whereas older locomotives used only saturated steam. Superheating is the process of putting more heat into the steam without changing the pressure. This provided more energy in the steam and allowed the locomotive to produce more work, but with a reduction in steam and fuel usage. In other words a superheated locomotive tended to be more efficient then a saturated locomotive.

Cylinder (Work done)

To drive the locomotive forward, steam was injected into the cylinder which pushed the piston backwards and forwards, and this in turn rotated the drive wheels of the locomotive. Typically the larger the drive wheels, the faster the locomotive was able to travel.

The faster the locomotive travelled the more steam that was needed to drive the cylinders. The steam able to be produced by the boiler was typically limited to a finite value depending upon the design of the boiler. In addition the ability to inject and exhaust steam from the cylinder also tended to reach finite limits as well. These factors typically combined to place limits on the power of a locomotive depending upon the design factors used.

Locomotive Types

During the course of their development, many different types of locomotives were developed, some of the more common categories are as follows:

- Simple – simple locomotives had only a single expansion cycle in the cylinder
- Compound – locomotives had multiple steam expansion cycles and typically had a high and low pressure cylinder.
- Saturated – steam was heated to only just above the boiling point of water.
- Superheated – steam was heated well above the boiling point of water, and therefore was able to generate more work in the locomotive.
- Geared – locomotives were geared to increase the tractive effort produced by the locomotive, this however reduced the speed of operation of the locomotive.

Superheated Locomotives

In the early 1900s, superheaters were fitted to some locomotives. As the name was implied a superheater was designed to raise the steam temperature well above the normal saturated steam temperature. This had a number of benefits for locomotive engineers in that it eliminated condensation of the steam in the cylinder, thus reducing the amount of steam required to produce the same amount of work in the cylinders. This resulted in reduced water and coal consumption in the locomotive, and generally improved the efficiency of the locomotive.

Superheating was achieved by installing a superheater element that effectively increased the heating area of the locomotive.

Geared Locomotives

In industrial type railways, such as those used in the logging industry, spurs to coal mines were often built to very cheap standards. As a consequence, depending upon the terrain, they were often laid with sharp curves and steep gradients compared to normal *main line standards*.

Typical *main line* rod type locomotives couldn't be used on these lines due to their long fixed wheelbase (coupled wheels) and their relatively low tractive effort was no match for the steep gradients. Thus geared locomotives found their niche in railway practice.

Geared locomotives typically used bogie wheelsets, which allowed the rigid wheelbase to be reduced compared to that of rod type locomotives, thus allowing the negotiation of tight curves. In addition the gearing allowed an increase of their tractive effort to handle the steeper gradients compared to main line tracks.

Whilst the gearing allowed more tractive effort to be produced, it also meant that the *maximum* piston speed was reached at a lower track speed.

As suggested above, the maximum track speed would depend upon loads and track conditions. As these types of lines were lightly laid, excessive speeds could result in derailments, etc.

The three principal types of geared locomotives used were:

- Shay Locomotives
- Climax
- Heisler

Steam Locomotive Operation

To successfully drive a steam locomotive it is necessary to consider the performance of the following elements:

- Boiler and Fire (Heat conversion)
- Cylinder (Work done)

For more details on these elements, refer to the “Elements of Steam Locomotive Operation”

Summary of Driving Tips

- Wherever possible, when running normally, have the regulator at 100%, and use the reverser to adjust steam usage and speed.
- Avoid jerky movements when starting or running the locomotive, thus reducing the chances of breaking couplers.
- When starting always have the reverser fully wound up, and open the regulator slowly and smoothly, without slipping the wheels.

Open Rails Steam Functionality (Fireman)

The Open Rails Steam locomotive functionality provides two operational options:

- Automatic Fireman (Computer Controlled): In Automatic or Computer Controlled Fireman mode all locomotive firing and boiler management is done by Open Rails, leaving the player to concentrate on driving the locomotive. Only the basic controls such as the regulator and throttle are available to the player.
- Manual Fireman: In Manual Fireman mode all locomotive firing and boiler management must be done by the player. All of the boiler management and firing controls, such as blower, injector, fuel rate, are available to the player, and can be adjusted accordingly.

Use the keys `<Crtl+F>` to switch between Manual and Automatic firing modes.

A full listing of the keyboard controls for use when in manual mode is provided on the *Keyboard* tab of the Open Rails *Options* panel.

Boiler Management

In Open Rails, the safe operating range for the boiler water level is 75-90% and this is maintained automatically by the AI Fireman. (Note: this is not the reading of the boiler water glass gauge but the %age full of the boiler.)

In manual mode, you must keep the boiler water level below 90%. A level of 91% or more drags water into the steam pipes and, being incompressible, the water will damage the cylinders. Open Rails does not model the damage but issues confirmation messages: “Boiler overfull and priming” and “Boiler no longer priming” on rising to 91% and falling below 90%

In manual mode, you must keep the boiler water level above 70%. A level below 70% uncovers the firebox crown. In real life, this is a catastrophic failure which melts the fusible plugs in the crown and that releases steam into the firebox and from there onto the footplate.

Open Rails does not model the steam but drops the boiler pressure and the fire and issues a confirmation message: "Water level dropped too far. Plug has fused and loco has failed." Basically the loco is coasting thereafter and nothing can be done to recover.

Hot or Cold Start

The locomotive can be started either in a hot or cold mode. Hot mode simulates a locomotive which has a full head of steam and is ready for duty.

Cold mode simulates a locomotive that has only just had the fire raised, and still needs to build up to full boiler pressure, before having full power available.

This function can be selected through the Open Rails options menu on the *Simulation* tab.

Main Steam Locomotive Controls

This section will describe the control and management of the steam locomotive based upon the assumption that the Automatic fireman is engaged. The following controls are those typically used by the driver in this mode of operation:

- Cylinder Cocks – allows water condensation to be exhausted from the cylinders. (Open Rails Keys: toggle <C>)
- Regulator – controls the pressure of the steam injected into the cylinders. (Open Rails Keys: <D> = increase, <A> = decrease)
- Reverser – controls the valve gear and when the steam is "cutoff". Typically it is expressed as a fraction of the cylinder stroke. (Open Rails Keys: <W> = increase, <S> = decrease). Continued operation of the W or S key will eventually reverse the direction of travel for the locomotive.
- Brake – controls the operation of the brakes. (Open Rails Keys: <'> = increase, <;> = decrease)

Recommended Option Settings

For added realism of the performance of the steam locomotive, it is suggested that the following settings be considered for selection in the Open Rails options menu:

- Break couplers
- Curve speed dependent
- Curve resistance speed
- Hot start
- Tunnel resistance dependent

NB: Refer to the relevant sections of the manual for more detailed description of these functions.

Locomotive Starting

Open the cylinder cocks. They are to remain open until the engine has traversed a distance of about an average train length, consistent with safety.

The locomotive should always be started in full gear (reverser up as high as possible), according to the direction of travel, and kept there for the first few turns of the driving wheels, before adjusting the reverser.

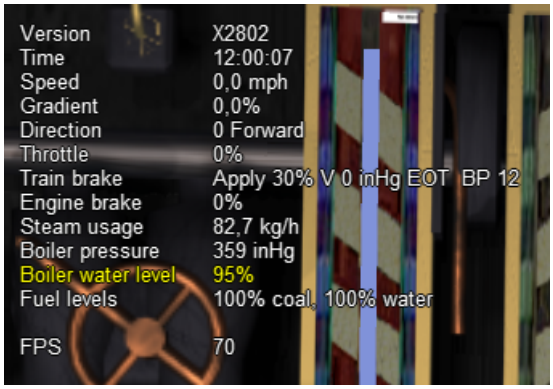
After ensuring that all brakes are released, open the regulator sufficiently to move the train, care should be exercised to prevent slipping; do not open the regulator too much before the locomotive has gathered speed. Severe slipping causes excessive wear and tear on the locomotive, disturbance of the fire bed and blanketing of the spark arrestor. If slipping does occur, the regulator should be closed as appropriate, and if necessary sand applied.

Also, when starting, a slow even increase of power will allow the couplers all along the train to be gradually extended, and therefore reduce the risk of coupler breakages.

Locomotive Running

Theoretically, when running, the regulator should always be fully open and the speed of the locomotive controlled, as desired, by the reverser. For economical use of steam, it is also desirable to operate at the lowest cut-off values as possible, so the reverser should be operated at low values, especially running at high speeds.

When running a steam locomotive keep an eye on the following key parameters in the Heads up Display (HUD – <F5>) as they will give the driver an indication of the current status and performance of the locomotive with regard to the heat conversion (Boiler and Fire) and work done (Cylinder) processes. Also bear in mind the above driving tips.



- Direction – indicates the setting on the reverser and the direction of travel. The value is in per cent, so for example a value of 50 indicates that the cylinder is cutting off at 0.5 of the stroke.
- Throttle – indicates the setting of the regulator in per cent.
- Steam usage – these values represent the current steam usage per hour.
- Boiler Pressure – this should be maintained close to the maximum working pressure of the locomotive.
- Boiler water level – indicates the level of water in the boiler. Under operation in Automatic Fireman mode, the fireman should manage this.
- Fuel levels – indicate the coal and water levels of the locomotive.

For information on the other parameters, such as the brakes, refer to the relevant sections in the manual.

For the driver of the locomotive the first two steam parameters are the key ones to focus on, as operating the locomotive for extended periods of time with steam usage in excess of the steam generation value

will result in declining boiler pressure. If this is allowed to continue the locomotive will ultimately lose boiler pressure, and will no longer be able to continue to pull its load.

Steam usage will increase with the speed of the locomotive, so the driver will need to adjust the regulator, reverser, and speed of the locomotive to ensure that optimal steam pressure is maintained. However, a point will finally be reached where the locomotive cannot go any faster without the steam usage exceeding the steam generation. This point determines the maximum speed of the locomotive and will vary depending upon load and track conditions

The AI Fireman in Open Rails is not proactive, ie it cannot look ahead for gradients, etc, and therefore will only add fuel to the fire once the train is on the gradient. This reactive approach can result in a boiler pressure drop whilst the fire is building up. Similarly if the steam usage is dropped (due to a throttle decrease, such as approaching a station) then the fire takes time to reduce in heat, thus the boiler pressure can become excessive.

To give the player a little bit more control over this, and to facilitate the maintaining of the boiler pressure the following key controls have been added to the AI Fireman function:

AIFireOn - (`<Alt+H>`) - Forces the AI fireman to start building the fire up (increases boiler heat & pressure, etc) - typically used just before leaving a station to maintain pressure as steam consumption increases. This function will be turned off if AIFireOff, AIFireReset are triggered or if boiler pressure or BoilerHeat exceeds the boiler limit.

AIFireOff - (`<Ctrl+H>`) - Forces the AI fireman to stop adding to the fire (allows boiler heat to decrease as fire drops) - typically used approaching a station to allow the fire heat to decrease, and thus stopping boiler pressure from exceeding the maximum. This function will be turned off if AIFireOn, AIFireReset are triggered or if boiler pressure or BoilerHeat drops too low.

AIFireReset - (`<Ctrl+Alt+H>`) - turns off both of the above functions when desired.

If theses controls are not used, then the AI fireman operates in the same fashion as previously.

Steam Boiler Heat Radiation Loss

A certain amount of heat is lost from the boiler of a steam locomotive. An uninsulated boiler could lose a lot of heat and this impacts on the performance of the locomotive, hence boilers were insulated to reduce the heat losses.

The amount of heat lost will be dependent upon the exposed surface area of the boiler, the difference in temperature between the boiler and the ambient temperature. The amount of heat lost will also increase as the speed of the locomotive increases.

OR models the heat loss from a boiler with some standard default settings, however the model can be customised to suit the locomotive by adjusting the following settings.

- `ORTSBoilerSurfaceArea` - Surface area of the boiler / fire box that impacts heat loss. Default UoM - (ft^2)
- `ORTSFractionBoilerInsulated` - Fraction of boiler surface area covered by insulation (less then 1)
- `ORTSHeatCoefficientInsulation` - Thermal conduction coefficient. Default UoM - (BTU / (ft^2 / hr.) / (1 (in. / F))

Steam Boiler Blowdown

Over time as steam is evaporated from the boiler a concentration of impurities will build up in the boiler. The boiler blowdown valve was used to remove these sediments from the boiler which could impact its efficiency. Depending upon the quality of the feed water used in the boiler, blowdown could be needed regularly when the locomotive was in operation.

The blowdown valve can be operated by toggling the `<Shft+C>` keys onn and off. Alternatively a cab control can be set up by using the `<ORTS_BLOWDOWN_VALVE ( x, y, z )>`.

A special steam effect can also be added. See the section on steam effects.

Steam Locomotive Carriage Steam Heat Modelling

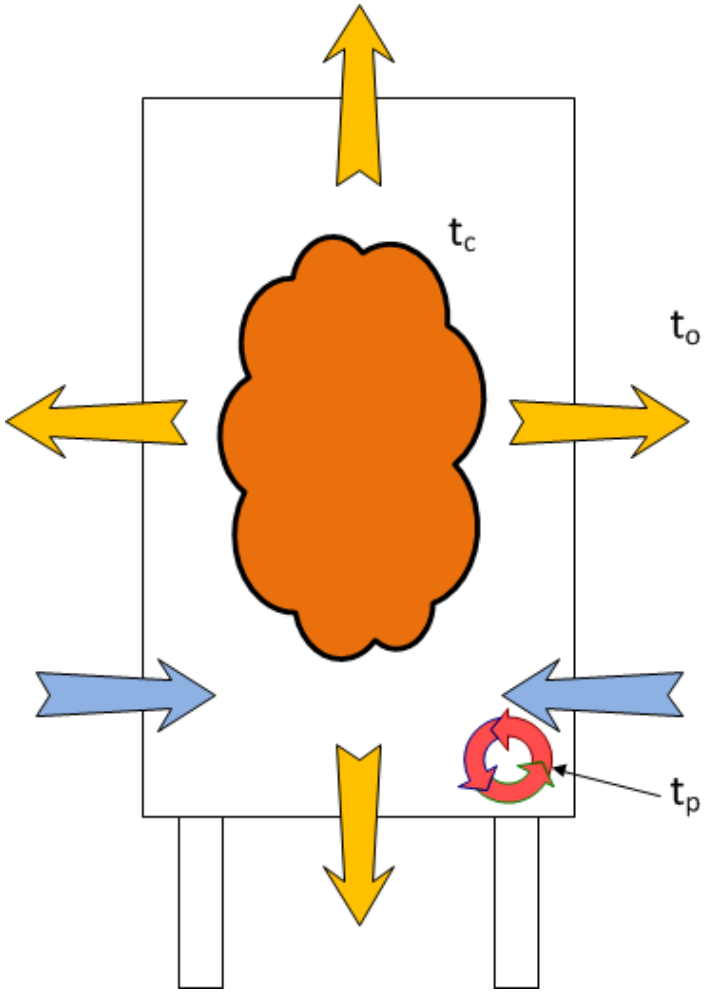
Overview

In the early days of steam, passenger carriages were heated by fire burnt in stoves within the carriage, but this type of heating proved to be dangerous, as on a number of occasions the carriages actually caught fire and burnt.

A number of alternative heating systems were adopted as a safer replacement.

The Open Rails Model is based upon a direct steam model, ie one that has steam pipes installed in each carriage, and pumps steam into each car to raise the internal temperature in each car.

The heat model in each car is represented by Figure 1 below. The key parameters influencing the operation of the model are the values of tc, to, tp, which represent the temperature within the carriage, ambient temperature outside the carriage, and the temperature of the steam pipe due to steam passing through it.



Heat Model for Passenger Car

As shown in the figure the heat model has a number of different elements as follows:

i. *Internal heat mass* – the air mass in the carriage (represented by cloud) is heated to temperature that is comfortable to the passengers. The energy required to maintain the temperature will be determined the volume of the air in the carriage.

ii. *Heat Loss – Transmission* – over time heat will be lost through the walls, roof, and floors of the carriage (represented by outgoing orange arrows), this heat loss will reduce the temperature of the internal air mass.

iii. *Heat Loss – Infiltration* – also over time as carriage doors are opened

and closed at station stops, some cooler air will enter the carriage (represented by ingoing blue arrows), and reduce the temperature of the internal air mass.

iv. *Steam Heating* – to offset the above heat losses, steam was piped through each of the carriages (represented by circular red arrows). Depending upon the heat input from the steam pipe, the temperature would be balanced by offsetting the steam heating against the heat losses.

Carriage Heating Implementation in Open Rails

Steam heating can be set up on steam locomotives, or on diesels with steam heating boilers, or alternatively with special cars that had steam heating boilers installed in them.

To enable steam heating to work in Open Rails the following parameter must be included in the engine section of the steam locomotive ENG File:

```
MaxSteamHeatingPressure( x )
```

Where: x = maximum steam pressure in the heating pipe – should not exceed 100 psi

If the above parameter is added to the locomotive, then an extra line will appear in the extended HUD to show the temperature in the train, and the steam heating pipe pressure, etc.

Steam heating will only work if there are passenger cars attached to the locomotive, or cars that have been set as requiring heating.

Warning messages will be displayed if the temperature inside the carriage drops below the temperature limits.

The player can control the train temperature by using the following controls:

- `<Alt+U>` – increase steam pipe pressure (and hence train temperature)
- `<Alt+D>` – decrease steam pipe pressure (and hence train temperature)

The steam heating control valve can be configured by adding an engine controller called `ORTSSteamHeat ( w, x, y, z)`. It should be configured as a standard 4 value controller.

The primary purpose of this model is to calculate steam usage for the heating, and in the case of a steam locomotive this will reduce available steam for the locomotive to use. Water and fuel usage in producing the heat will also result in the mass of the locomotive or steam heating van to be reduced.

It should be noted that the impact of steam heating will vary depending upon the season, length of train, etc.

A set of standard default parameters are included in Open Rails which will allow steam heating to work once the above changes have been implemented.

For those who would like to customise the steam heating the following parameters which can be inserted in the wagon file section can be adjusted as follows.

The passenger (or other heated cars) can be adjusted with the following parameters:

- `ORTSHeatingWindowDeratingFactor` - is the fraction of the car side that is occupied by windows.
- `ORTSHeatingCompartmentTemperatureSet` - is the temperature that the car thermostat is set to.

- `ORTSHeatingCompartmentPipeAreaFactor` - is a factor that adjusts the heating area of the steam heater in the passenger compartment.
- `ORTSHeatingTrainPipeOuterDiameter` - outer diameter of the main steam pipe that runs the length of the train.
- `ORTSHeatingTrainPipeInnerDiameter` - inner diameter of the main steam pipe that runs the length of the train.
- `ORTSHeatingConnectingHoseOuterDiameter` - outer diameter of the connecting hose between carriages.
- `ORTSHeatingConnectingHoseInnerDiameter` - inner diameter of the connecting hose between carriages.

For diesel locomotives or steam heating boiler vans the following parameters can be used to set the parameters of the steam heating boiler:

- `ORTSWagonSpecialType` - can be used to indicate whether the car is a boiler van (set = HeatingBoiler), or if the car is heated (set = Heated).
- `ORTSHeatingBoilerWaterUsage` - is the water usage for the steam heating boiler, and is a table with a series of x and y parameters, where x = steam usage (lb/hr) and y = water usage (g-uk/hr).
- `ORTSHeatingBoilerFuelUsage` - is the fuel usage for the steam heating boiler, and is a table with a series of x and y parameters, where x = steam usage (lb/hr) and y = fuel usage (g-uk/hr).
- `ORTSHeatingBoilerWaterTankCapacity` - is the feed water tank capacity for the steam boiler.
- `ORTSHeatingBoilerFuelTankCapacity` - is the fuel tank capacity for the steam boiler. Applies to steam heating boiler cars only.

Special effects can also be added to support the steam heating model, see the section *Special Visual Effects for Locomotives or Wagons* for more information.

Steam Locomotives – Physics Parameters for Optimal Operation

Required Input ENG and WAG File Parameters

The OR Steam Locomotive Model (SLM) should work with default MSTS files; however optimal performance will only be achieved if the following settings are applied within the ENG file. **The following list only describes the parameters associated with the SLM, other parameters such as brakes, lights, etc. still need to be included in the file.** As always, make sure that you keep a backup of the original MSTS file.

Open Rails has been designed to do most of the calculations for the modeler, and typically only the key parameters are required to be included in the ENG or WAG file. The parameters shown in the *Locomotive performance Adjustments* section should be included only where a specific performance outcome is required, since *default* parameters should provide a satisfactory result.

When creating and adjusting ENG or WAG files, a series of tests should be undertaken to ensure that the performance matches the actual real-world locomotive as closely as possible. For further information on testing, as well as some suggested test tools, go to this site.

NB: These parameters are subject to change as Open Rails continues to develop.

Notes:

- New – parameter names starting with *ORTS* means added as part of OpenRails development
- Existing – parameter names not starting with *ORTS* are original in MSTS or added through MSTS BIN

Possible Locomotive Reference Info:

i. Steam Locomotive Data

ii. Example Wiki Locomotive Data

iii. Testing Resources for Open Rails Steam Locomotives

Parameter	Description	Recommended Units	Typical Examples
General Information (Engine section)			
ORTSSteam-LocomotiveType (x)	Describes the type of locomotive	Simple, Compound, Geared	(Simple) (Compound) (Geared)
WheelRadius (x)	Radius of drive wheels	Distance	(0.648m) (36in)
MaxSteamHeating-Pressure (x)	Max pressure in steam heating system for passenger carriages	Pressure, NB: normally < 100 psi	(80psi)
Boiler Parameters (Engine section)			
ORTSSteamBoilerType (x)	Describes the type of boiler	Saturated, Superheated	(Saturated) (Superheated)
BoilerVolume (x)	Volume of boiler. This parameter is not overly critical.	Volume, where an act. value is n/a, use approx. EvapArea / 8.3	(“220*(ft^3)”) (“110*(m^3)”)
ORTSEvaporationArea (x)	Boiler evaporation area	Area	(“2198*(ft^2)”) (“194*(m^2)”)
MaxBoilerPressure (x)	Max boiler working pressure (gauge)	Pressure	(200psi) (200kPa)
ORTSSuperheatArea (x)	Superheating heating area	Area	(“2198*(ft^2)”) (“194*(m^2)”)
Locomotive Tender Info (Wagon section - will override Engine section values)			
ORTSTenderWagon-WaterMass (x)	Water in tender	Mass	(36500lb) (16000kg)
ORTSTenderWagon-CoalMass (x)	Coal in tender	Mass	(13440lb) (6000kg)
Locomotive Tender Info (Engine section)			
MaxTenderWaterMass (x)	Water in tender	Mass	(36500lb) (16000kg)
MaxTenderCoalMass (x)	Coal in tender	Mass	(13440lb) (6000kg)

continues on next page

Table 1 – continued from previous page

Parameter	Description	Recommended Units	Typical Examples
IsTenderRequired (x)	Locomotive Requires a tender	0 = No, 1 = Yes	(0) (1)
Fire (Engine section)			
ORTSGrateArea (x)	Locomotive fire grate area	Area	(“2198*(ft^2)”) (“194*(m^2)”)
ORTSFuelCalorific (x)	Calorific value of fuel	For coal use 13700 btu/lb	(13700btu/lb) (33400kj/kg)
ORTSSteamFireman-MaxPossibleFiringRate (x)	Maximum fuel rate that fireman can shovel in an hour. (Mass Flow)	Use as def: UK:3000lb/h US:5000lb/h AU:4200lb/h	(4200lb/h) (2000kg/h)
SteamFiremanIs-MechanicalStoker (x)	Mechanical stoker = large rate of coal feed	Boolean, 0=no-stoker 1=stoker	(1)
Steam Cylinder (Engine section)			
NumCylinders (x)	Number of steam cylinders	Boolean	(2)
CylinderStroke (x)	Length of cylinder stroke	Distance	(26in) (0.8m)
CylinderDiameter (x)	Cylinder diameter	Distance	(21in) (0.6m)
LPNumCylinders (x)	Number of steam LP cylinders (compound locomotive only)	Boolean	(2)
LPCylinderStroke (x)	LP cylinder stroke length (compound locomotive only)	Distance	(26in) (0.8m)
LPCylinderDiameter (x)	Diameter of LP cylinder (compound locomotive only)	Distance	(21in) (0.6m)
Friction (Wagon section)			
ORTSDavis_A (x)	Journal or roller bearing + mechanical friction	N, lbf. Use FCalc to calculate	(502.8N) (502.8lb)
ORTSDavis_B (x)	Flange friction	Nm/s, lbf/mph. Use FCalc	(1.5465Nm/s) (1.5465lbf/mph)
ORTSDavis_C (x)	Air resistance friction	Nm/s^2, lbf/mph^2 Use FCalc	(1.43Nm/s^2) (1.43lbf/mph^2)
ORTSBearingType (x)	Bearing type, defaults to Friction	Roller, Friction, Low	(Roller)
Friction (Engine section)			

continues on next page

Table 1 – continued from previous page

Parameter	Description	Recommended Units	Typical Examples
ORTSDriveWheel-Weight (x)	Total weight on the locomotive driving wheels	Mass, Leave out if unknown	(2.12t)
Curve Speed Limit (Wagon section)			
ORTSUnbalanced-SuperElevation (x)	Determines the amount of Cant Deficiency applied to carriage	Distance, Leave out if unknown	(3in) (0.075m)
ORTSTrackGauge (x)	Track gauge	Distance, Leave out if unknown	(4ft 8.5in) (1.435m) (4.708ft)
CentreOfGravity (x, y, z)	Defines the centre of gravity of a locomotive or wagon	Distance, Leave out if unknown	(0m, 1.8m, 0m) (0ft, 5.0ft, 0ft)
Curve Friction (Wagon section)			
ORTSRigidWheelBase (x)	Rigid wheel base of vehicle	Distance, Leave out if unknown	(5ft 6in) (3.37m)
Locomotive Gearing (Engine section – Only required if locomotive is geared)			
ORTSSteamGearRatio (a, b)	Ratio of gears	Numeric	(2.55, 0.0)
ORTSSteamMaxGear-PistonRate (x)	Max speed of piston	ft/min	(650)
ORTSSteamGearType (x)	Fixed gearing or selectable gearing	Fixed, Select	(Fixed) (Select)
ORTSGearedTractive-EffortFactor (x)	**Factor to include** in TE calculation	Fixed	(Fixed)
Locomotive Performance Adjustments (Engine section – Optional, for experienced modellers)			
ORTSBoiler-EvaporationRate (x)	Multipl. factor for adjusting maximum boiler steam output	Be tween 10–15, Leave out if not used	(15.0)
ORTSBurnRate (x, y)	Tabular input: Coal combusted (y) to steam generated (x)	x – lbs, y – kg, series of x & y values. Leave out if unused	
ORTSCylinder-EfficiencyRate (x)	Multipl. factor for steam cylinder (force) output	Un limited, Leave out if unused	(1.0)
ORTSBoilerEfficiency (x, y)	Tabular input: boiler efficiency (y) to coal combustion (x)	x – lbs/ft2/h, series of x & y values. Leave out if unused	
ORTSCylinderPort-Opening (x)	Size of cylinder port opening	Between 0.05–0.12, Leave out if unused	(0.085)
ORTSCylinderInitial-PressureDrop (x, y)	Tabular input: wheel speed (x) to pressure drop factor (y)	x – rpm, series of x & y values. Leave out if unused	

continues on next page

Table 1 – continued from previous page

Parameter	Description	Recommended Units	Typical Examples
ORTSCylinderBack-Pressure (x, y)	Tabular input: Loco indicated power (x) to backpressure (y)	x – hp, y – psi(g), series of x & y values. Leave out if unused	

Special Visual Effects for Locomotives or Wagons

Steam exhausts on a steam locomotive, and other special visual effects can be modelled in OR by defining appropriate visual effects in the `SteamSpecialEffects` section of the steam locomotive ENG file, the `DieselSpecialEffects` section of the diesel locomotive ENG file, or the `SpecialEffects` section of a relevant wagon (including diesel, steam or electric locomotives.

OR supports the following special visual effects in a steam locomotive:

- Steam cylinders (named `CylindersFX` and `Cylinders2FX`) – two effects are provided which will represent the steam exhausted when the steam cylinder cocks are opened. Two effects are provided to represent the steam exhausted at the front and rear of each piston stroke. These effects will appear whenever the cylinder cocks are opened, and there is sufficient steam pressure at the cylinder to cause the steam to exhaust, typically the regulator is open (> 0%).
- Stack (named `StackFX`) – represents the smoke stack emissions. This effect will appear all the time in different forms depending upon the firing and steaming conditions of the locomotive.
- Compressor (named `CompressorFX`) – represents a steam leak from the air compressor. Will only appear when the compressor is operating.
- Generator (named `GeneratorFX`) – represents the emission from the turbo-generator of the locomotive. This effect operates continually. If a turbo-generator is not fitted to the locomotive it is recommended that this effect is left out of the effects section which will ensure that it is not displayed in OR.
- Safety valves (named `SafetyValvesFX`) – represents the discharge of the steam valves if the maximum boiler pressure is exceeded. It will appear whenever the safety valve operates.
- Whistle (named `WhistleFX`) – represents the steam discharge from the whistle.
- Injectors (named `Injectors1FX` and `Injectors2FX`) – represents the steam discharge from the steam overflow pipe of the injectors. They will appear whenever the respective injectors operate.
- Ejectors (named `SmallEjectorFX` and `LargeEjectorFX`) – represents the steam discharge from the steam ejectors associated with vacuum braking. They will appear whenever the respective ejectors operate.
- Boiler blowdown valves (named `BlowdownFX`) – represents the discharge of the steam boiler blowdown valve. It will appear whenever the blowdown valve operates.

OR supports the following special visual effects in a diesel locomotive:

- Exhaust (named `Exhaustnumber`) – is a diesel exhaust. Multiple exhausts can be defined, simply by adjusting the numerical value of the number after the key word exhaust.

OR supports the following special visual effects in a wagon (also the wagon section of an ENG file):

- Steam Heating Boiler (named `HeatingSteamBoilerFX`) – represents the exhaust for a steam heating boiler. Typically this will be set up on a diesel or electric train as steam heating was provided directly from a steam locomotive.
- Wagon Generator (named `WagonGeneratorFX`) – represents the exhaust for a generator. This generator was used to provide additional auxiliary power for the train, and could have been used for air conditioning, heating lighting, etc.
- Wagon Smoke (named `WagonSmokeFX`) – represents the smoke coming from say a wood fire. This might have been a heating unit located in the guards van of the train.

- Heating Hose (named `HeatingHoseFX`) – represents the steam escaping from a steam pipe connection between wagons.
- Heating Compartment Steam Trap (named `HeatingCompartmentSteamTrapFX`) – represents the steam escaping from the steam trap under a passenger compartment.
- Heating Main Pipe Steam Trap (named `HeatingMainPipeSteamTrapFX`) – represents the steam escaping from a steam trap in the main steam pipe running under the passenger car.

NB: If a steam effect is not defined in the `SteamSpecialEffects`, `DieselSpecialEffects`, or the `SpecialEffects` section of an ENG/WAG file, then it will not be displayed in the simulation. Similarly if any of the co-ordinates are zero, then the effect will not be displayed.

Each effect is defined by inserting a code block into the ENG/WAG file similar to the one shown below:

```
CylindersFX (
    -1.0485 1.0 2.8
    -1  0  0
    0.1
)
```

The code block consists of the following elements:

- Effect name – as described above,
- Effect location on the locomotive (given as an x, y, z offset in metres from the origin of the wagon shape)
- Effect direction of emission (given as a normal x, y and z)
- Effect nozzle width (in metres)

Auxiliary Water Tenders

To increase the water carrying capacity of a steam locomotive, an *auxiliary tender* (or as known in Australia as a water gin) would sometimes be coupled to the locomotive. This auxiliary tender would provide additional water to the locomotive tender via connecting pipes.

Typically, if the connecting pipes were opened between the locomotive tender and the auxiliary tender, the water level in the two vehicles would equalise at the same height.

To implement this feature in Open Rails, a suitable water carrying vehicle needs to have the following parameter included in the WAG file.

`ORTSAuxTenderWaterMass ( 70000lb )` The units of measure are in mass.

When the auxiliary tender is coupled to the locomotive the *tender* line in the LOCOMOTIVE INFORMATION HUD will show the two tenders and the water capacity of each. Water (C) is the combined water capacity of the two tenders, whilst Water (T) shows the water capacity of the locomotive tender, and Water (A) the capacity of the auxiliary tender (as shown below).



To allow the auxiliary tender to be filled at a water fuelling point, a water freight animation will be need to be added to the WAG file as well. (Refer to *Freight Animations* for more details).

8.2.4 Unpowered Control Car

This module simulates the control cab of a DMU set of cars. The cab typically would be located in an unpowered trailer car which is attached to a powered car for the provision of its motive force to drive the train forward.

Apart from producing motive force the car (and cabin controls) should behave exactly the same as a locomotive.

To set a control car up it is necessary to produce an ENG file which has the `Type ( Control )` parameter set in the engine section of the ENG file.

The Control car uses most of the same parameters for its configuration as a powered locomotive. The major items that can be left out are those parameters associated with power, motive force, diesel engines, some braking items, such as the compressor and main air reservoir, and some of the diesel effects (as it has no diesel engine).

Some of the cab monitoring gauges provide visibility of what is happening on the powered car. To do this OR searches for the "closest" powered car near the Control car and uses its information.

8.3 Multiple Units of Locomotives in Same Consist

In an OR player train one locomotive is controlled by the player, while the other units are controlled by default by the train's MU (multiple unit) signals for braking and throttle position, etc. The player-controlled locomotive generates the MU signals which are passed along to every unit in the train.

8.3.1 Distributed Power

This is applicable only to trains hauled by diesel locomotives equipped with dynamic brakes.

More locomotive groups may be present in American long freight trains; a locomotive group is defined as a set of locomotives that have no wagons in between. Groups different from the group including the lead locomotive are called remote groups.

Remote groups can be controlled in two modes: *synchronous* or *asynchronous*. Locomotives in the player train can be arranged to be part of one of the two above control groups. So each locomotive group (except the lead one, which always belongs to the *sync* control group) can be either part the *sync* or of the *async* control group. However, if a locomotive group is part of the *async* control group, all locomotive groups behind it must also be part of the *async* control group.

The arrangement can be changed during run, which in real life is performed by using the locomotive's onboard computer (e.g. Locotrol).

This functionality is necessary in American long-train freight operations. An example use case is when a train finished climbing a mountain pass. At this point the lead locomotives have to start dynamic braking driving downhill, while the trailing units still need to keep pushing the train's end uphill. The locomotive driver has the possibility to build the *fence*, i.e. move the last locomotive group to the *async* control group.

While locomotives in *sync* control group always copy the traction and dynamic brake settings of the lead (man-controlled) locomotive, for locomotives in *async* control group these can be adjusted independently. Other controls, like reverser and air brake are always synchronized throughout the whole train, and changing reverser setting or applying air brakes will force *async* controlled locomotives to fall back to idle.

The *fence* between *sync* and *async* control groups can be moved back and forth along the train. This is useful when there are one or more middle-train locomotive groups in the consist. However it is not possible to move the fence to separate two directly interconnected locomotives: such locomotives can be rearranged only together.

If the driver dismantles the *fence*, by moving it after the last locomotive group, all locomotive groups are moved back to *sync* control group, and therefore all locomotives will work in sync with the leading one. All-sync operation is also the default, when the game starts.

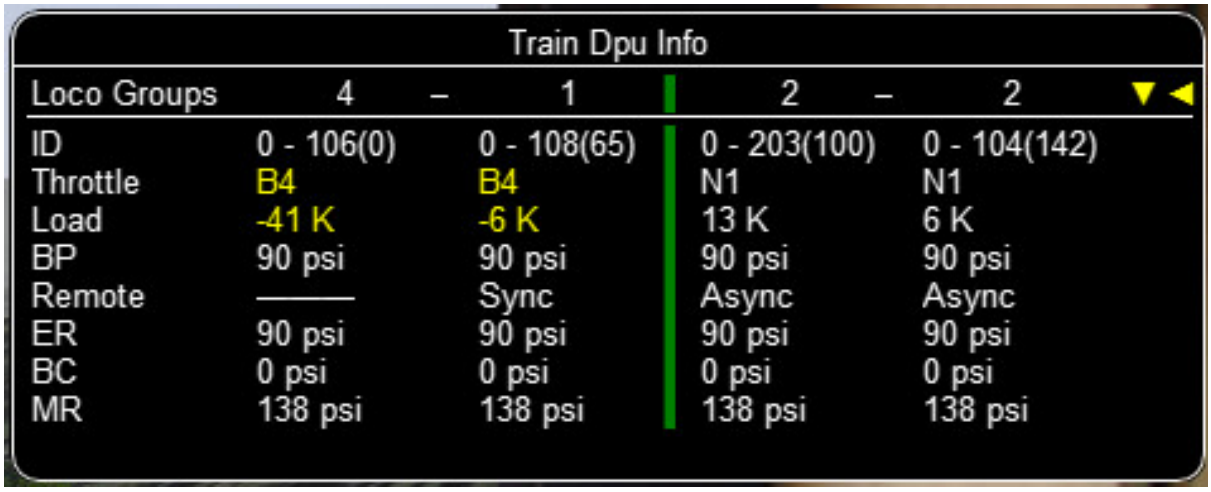
Traction and dynamic brake settings for *sync* group can be controlled by the usual keys: <A> and <D>. The following additional controls are available for controlling the *async* group settings:

- Move To Back – <Ctrl+Shift+O>: Move one locomotive group to back (*async*) control group (*fence* is moved towards the front of the train).
- Move To Front – <Ctrl+O>: Move one locomotive group to front (*sync*) control group (*fence* is moved towards the back of the train).
- Traction – <Ctrl+L>: Switch *async* group to traction mode.
- Idle – <Ctrl+Shift+L>: Switch *async* group to idle state.
- Brake – <Ctrl+'> (key two positions at the right of L): Switch *async* group to dynamic braking mode.
- More – <Ctrl+U>: Increase *async* group traction or dynamic brake by a notch, depending on its mode setting.
- Less – <Ctrl+Shift+U>: Decrease *async* group traction or dynamic brake by a notch, depending on its mode setting.

HUD shows the sync–async configuration in line *Multiple Units* on main page. When it reads e.g. "2–2 | 1", then it means that front and middle-train double-unit locomotives are controlled in *sync* with the leading unit, while the trailing pushing unit is controlled *async* independently. The actual set value of traction or dynamic brake of *async* group is shown in lines *Throttle* and *Dynamic Brake*, respectively, in brackets, e.g.: Throttle: 0% (50%).

Distributed power info and commands can also be displayed and operated through cabview controls, as explained *here*

The complete distributed power configuration is displayed in the Distributed Power Info extended HUD page, where the state of all locomotives in the train are shown, as well as in the *Train DPU Info* window, which is displayed after pressing <Shift+F9>, and which shows only the state of the first locomotive of each locomotive group, as occurs also in displays of real locomotives.



8.3.2 Engines of AI Trains

For AI trains, the AI software directly generates the remote control signals, i.e. there is no player-controlled locomotive. In this way, all engines use the same physics code for power and friction.

This software model will ensure that non-player controlled engines will behave exactly the same way as player controlled ones.

8.4 Open Rails Braking

Open Rails software has implemented its own braking physics in the current release. It is based on the Westinghouse 26C and 26F air brake and controller system. Open Rails braking will parse the type of braking from the .eng file to determine if the braking physics uses passenger or freight standards, self-lapping or not.

There are two different features regarding graduated release of brakes. If the train brake controller has a self-lapping notch that provides graduated release, then the amount of brake pressure can be adjusted up or down by changing the control in this notch. If the notch does not provide graduated release, then the brakes can only be increased in this notch and one of the release positions is required to release the brakes. The list of notches that have graduated release can be found *here*.

To achieve a graduated release, the brake valves in the train cars must have this capability. If the BrakeEquipmentType() parameter in the Wagon() section contains "Graduated_release_triple_valve" or "Distributor", then the brake cylinder pressure is regulated to keep it proportional to the difference between the emergency reservoir pressure and the brake pipe pressure. If the brake valve is a "Triple_valve" instead, when the brake pipe pressure rises above the auxiliary reservoir pressure, the brake cylinder pressure is released completely at a rate determined by the retainer setting.

Selecting *Graduated Release Air Brakes* in *Menu* > *Options* will force self-lapping notches in the brake controller to have graduated release. It will also force graduated release of brakes in triple valves. This option should be unchecked, except for compatibility problems with old MSTS stock.

The following brake types are implemented in OR:

- Vacuum single
- Air single-pipe
- Air twin-pipe
- EP (Electro-pneumatic)
- Single-transfer-pipe (air and vacuum)

The operation of air single-pipe brakes is described in general below.

The auxiliary reservoir needs to be charged by the brake pipe and, depending on the WAG file parameters setting, this can delay the brake release. The auxiliary reservoir is also charged by the emergency reservoir (until both are equal and then both are charged from the pipe).

Increasing the *Brake Pipe Charging Rate* (psi/s) value controls the charging rate. Increasing the value will reduce the time required to recharge the train; while decreasing the value will slow the charging rate. However, this might be limited by the train brake controller parameter settings in the ENG file. The brake pipe pressure cannot go up faster than that of the equalization reservoir.

The default value, 21, should cause the recharge time from a full set to be about 1 minute for every 12 cars. If the *Brake Pipe Charging Rate* (psi/s) value is set to 1000, the pipe pressure gradient features will be disabled and will also disable some but not all of the other new brake features.

Brake system charging time depends on the train length as it should, but at the moment there is no modeling of main reservoirs and compressors.

8.4.1 Train Brake Controller Positions

The following notch positions can be defined for the train brake at `Engine(EngineControllers(Brake_Train`:

Table 2: Brake Controller Tokens

OR Brake Token	Description	Brake Systems	Operation
RELEASE and RUNNING tokens			
TrainBrakes-Controller-OverchargeStart	RELEASE or OVERCHARGE Rapidly releases air brakes and charges air reservoirs. Train brake pipe may be overcharged (up to ORTSTrainBrakesControllerMax-OverchargePressureand will gradually return to normal working pressure when the controller is moved to a release position (the rate will be determined by ORTSTrain BrakesControllerOvercharge. Elimination-Rate	Air single pipe Air twin pipe EP	Air
TrainBrakes-ControllerFull-QuickRelease-Start	RELEASE or QUICK RELEASE Air brakes: Rapidly releases air brakes and charges air reservoirs, without overcharging the train pipe. EP brakes: Rapidly releases EP brakes. Vacuum brakes – diesel and electric loco: Operates exhauster at fast speed. Rapidly releases vacuum brakes and charges vacuum reservoirs. Vacuum brakes – steam with combination ejector: Operates large ejector at full power. Rapidly releases vacuum brakes and charges vacuum reservoirs. Vacuum brakes – steam with separate ejector: Connects brake pipe to ejector(s) and/or vacuum pump. Brakes may be released by operating large or small ejector.	Air single pipe Air twin pipe EP Vacuum single pipe	Air EP Vacuum
TrainBrakes-Controller-ReleaseStart	RUNNING or RELEASE Air brakes: Maintains working pressure in train pipe. Slowly releases brakes. EP brakes: Releases brakes. Vacuum brakes – diesel and electric loco: Connects brake pipe to exhauster. Maintains vacuum in train pipe. Slowly releases brakes. Vacuum brakes – steam with combination ejector: Operates large ejector at full power. Rapidly releases vacuum brakes and charges vacuum reservoirs. Vacuum brakes – steam with separate ejector: Connects brake pipe to ejector(s) and/or vacuum pump. Brakes may be released by operating large or small ejector.	Air single pipe Air twin pipe EP Vacuum single pipe	Air EP Vacuum
LAP, HOLDING and NEUTRAL tokens			
TrainBrakes-Controller-RunningStart	LAP or RUNNING Air brakes: Train pipe pressure is held at any pressure with compensation for leakage. EP brakes: Brake application is held at any value. Vacuum brakes – diesel and electric loco: Train pipe vacuum is held at any value with compensation for leakage. Vacuum brakes – steam with combination ejector: Connects brake pipe to small ejector and/or vacuum pump. Maintains vacuum. Brakes may be released by operating small ejector. (Vacuum brakes – steam with separate ejector: Connects brake pipe only to small ejector and/or vacuum pump.)	Air single pipe Air twin pipe EP Vacuum single pipe	Air EP Vacuum

8.4.2 Brake Shoe Adhesion

The braking of a train is impacted by the following two types of adhesion (friction coefficients):

- **Brakeshoe** – the coefficient of friction of the brakeshoe varies due to the type of brake shoe, and the speed of the wheel increases. Typically older cast iron brake shoes had lower friction coefficients then more modern composite brakeshoes.
- **Wheel** – the adhesion or friction coefficient between the wheel and the rail will also vary with different conditions, such as whether the track was dry or wet, and will also vary with the speed of rotation of the wheel.

Thus a train traveling at high speed will have lower brake shoe adhesion, which means that the train will take a longer time to stop (or alternatively more force needs to be applied to the brakeshoe to achieve the same slowing effect of the wheel, as at slower speeds). Traveling at high speeds may also result in insufficient force being available to stop the train, and therefore under some circumstances the train may become uncontrollable (unstoppable) or *runaway* on steep falling gradients.

Conversely if too much force is applied to the brakeshoe, then the wheel could *lock up*, and this could result in the wheel slipping along the rail once the *adhesive force* (wagon weight x coefficient of friction) of the wagon is exceeded by the braking force. In this instance the static friction between the wheel and the track will change to dynamic friction, which is significantly lower than the static friction, and thus the train will not be stopped in the desired time and distance.

When designing the braking forces railway engineers need to ensure that the maximum braking force applied to the wheels takes into account the above adhesion factors.

Implementation in Open Rails

Open Rails models the aspects described above, and operates within one of the following modes:

- Advanced Adhesion NOT selected - brake force operates as per previous OR functionality, i.e. - constant brake force regardless of speed.
- Advanced Adhesion SELECTED and legacy WAG files, or NO additional user friction data defined in WAG file - OR assumes the users assigned friction coefficient have been set at 20% friction coefficient for cast iron brakes, and reverse engineers the braking force, and then applies the default friction curve as the speed varies.
- Advanced Adhesion SELECTED and additional user friction data HAS been defined in WAG file - OR applies the user defined friction/speed curve.

It should be noted that the MaxBrakeForce parameter in the WAG file is the actual force applied to the wheel after reduction by the friction coefficient.

Option iii) above is the ideal recommended method of operating, and naturally will require include files, or variations to the WAG file.

To setup the WAG file, the following values need to be set:

- use the OR parameter `ORTSBrakeShoeFriction ( x, y )` to define an appropriate friction/speed curve, where x = speed in kph, and y = brakeshoe friction. This parameter needs to be included in the WAG file near the section defining the brakes. This parameter allows the user to customise to any brake type.
- Define the `MaxBrakeForce` value with a friction value equal to the zero speed value of the above curve, i.e. in the case of the curve below this woyuld be 0.49.

For example, a sample curve definition for a COBRA (COmposition BRAkes) brakeshoe might be as follows:

```
ORTSBrakeShoeFriction ( 0.0 0.49 8.0 ..............  80.5 0.298 88.5 0.295 96.6 0.289 104.6
0.288 )
```

The debug FORCES INFORMATION HUD has been modified by the addition of two extra columns:

- Brk. Frict. - Column shows the current friction value of the brakeshoe and will vary according to the speed. (Applies to modes ii) and iii) above). In mode i) it will show friction constant at 100%, which indicates that the MaxBrakeForce defined in the WAG file is being used without alteration, ie it is constant regardless of the speed.
- Brk. Slide - indicates that the vehicle wheels are sliding along the track under brake application. (Ref to *Wheel Skidding due to Excessive Brake Force*)

It should be noted that the *Adhesion factor correction* slider in the options menu will vary the brakeshoe coefficient above and below 100% (or unity). It is recommended that this is set @ the default value of 100%.

These changes introduce an extra challenge to train braking, but provide a more realistic train operation.

For example, in a lot of normal Westinghouse brake systems, a minimum pressure reduction was applied by moving the brake controller to the LAP position. Typically Westinghouse recommended values of between 7 and 10 psi.

8.4.3 Train Brake Pipe Losses

The train brake pipe on a train is subject to air losses through leakage at joints, etc. Typically when the brake controller is in the RUNNING position, air pressure is maintained in the pipe from the reservoir. However on some brake systems, especially older ones such as the A6-ET, when the brake controller is in the LAP position the train brake pipe is isolated from the air reservoir, and hence over time the pipe will suffer pressure drops due to leakages. This will result in the brakes being gradually applied.

Some brake systems allow small leakage to happen without applying brakes. This can be regulated with the parameter `ORTSBrakeInsensitivity` in the WAG file. UIC vehicles are usually insensitive to pressure drop rates lower than 0.3 bar/min. This parameter also prevents brakes being applied while an overcharge in the brake pipe is being eliminated.

More modern systems have a self lapping feature which compensates for train brake pipe leakage regardless of the position that the brake controller is in.

Open Rails models this feature whenever the `TrainPipeLeakRate` parameter is defined in the engine section of the ENG file. Typically most railway companies accepted leakage rates of around 5 psi/min in the train brake pipe before some remedial action needed to be undertaken.

If this parameter is left out of the ENG file, then no leakage will occur.

8.4.4 Wheel Skidding due to Excessive Brake Force

The application of excessive braking force onto a wheel can cause it to lock up and then start to slip along the rails. This occurs where the wagon braking force exceeds the adhesive weight force of the wagon wheel, i.e. the wheel to rail friction is overcome, and the wheel no longer *grips* the rails.

Typically this happens with lightly loaded vehicles at lower speeds, and hence the need to ensure that braking forces are applied to design standards. Skidding will be more likely to occur when the adhesion between the wheel and track is low, so for example skidding is more likely in wet weather then dry weather. The value *Wag Adhesion* in the FORCES INFORMATION HUD indicates this adhesion value, and will vary with the relevant weather conditions.

When a vehicle experiences wheel skid, an indication is provided in the FORCES INFORMATION HUD. To correct the problem the brakes must be released, and then applied slowly to ensure that the wheels are not *locked* up. Wheel skid will only occur if ADVANCED adhesion is selected in the options menu.

(Ref to *Wheel Skidding due to Excessive Brake Force* for additional information)

8.4.5 Using the F5 HUD Expanded Braking Information

This helps users of Open Rails to understand the status of braking within the game and assists in realistically coupling and uncoupling cars. Open Rails braking physics is more realistic than MSTS, as it models the connection, charging and exhaust of brake lines.

When coupling to a static consist, note that the brake line for the newly added cars normally does not have any pressure. This is because the train brake line/hose has not yet been connected. The last columns of each line shows the condition of the air brake hose connections of each unit in the consist.

BRAKE INFORMATION
Main reservoir 135 psi

Car	Type	BrkCyl	BrkPipe	AuxRes	ErgRes	MRPipe	RetValve	TripleValve	Handbrk	Conn	AnglCock BleedOff
0 - 0	1P	11 psi	90 psi	90 psi	90 psi			Release		T	A- B+
0 - 1	1P	11 psi	90 psi	90 psi	90 psi			Release		I	A+ B-
32884 - 0	1P	0 psi	0 psi	0 psi	0 psi			Emergency	100%	T	A+ B+
32884 - 1	1P	0 psi	0 psi	0 psi	0 psi			Emergency	100%	T	A+ B+
32884 - 2	1P	0 psi	0 psi	0 psi	0 psi			Emergency	100%	T	A+ B+

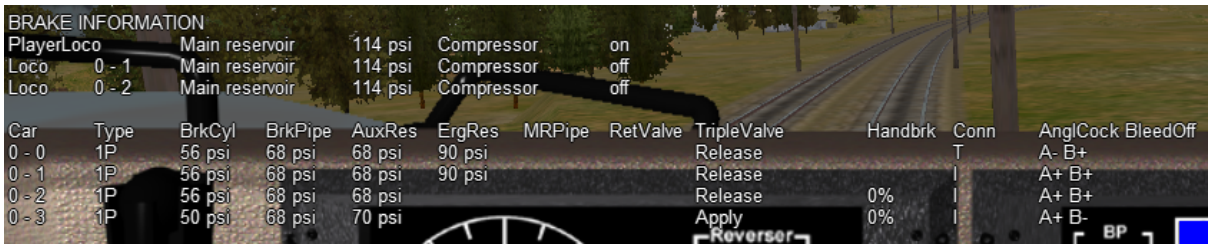
The columns under *AnglCock* describe the state of the *Angle Cock*, a manually operated valve in each of the brake hoses of a car: A is the cock at the front, B is the cock at the rear of the car. The symbol `+` indicates that the cock is open and the symbol `–` that it is closed. The column headed by `T` indicates if the hose on the locomotive or car is interconnected: `T` means that there is no connection, `I` means it is connected to the air pressure line. If the angle cocks of two consecutive cars are B+ and A+ respectively, they will pass the main air hose pressure between the two cars. In this example note that the locomotive air brake lines start with A- (closed) and end with B- (closed) before the air hoses are connected to the newly coupled cars. All of the newly coupled cars in this example have their angle cocks open, including those at the ends, so their brake pressures are zero. This will be reported as *Emergency* state.

Coupling Cars

Also note that, immediately after coupling, you may also find that the handbrakes of the newly added cars have their handbrakes set to 100% (see column headed *Handbrk*). Pressing `<Shift+;>` (Shift plus semicolon in English keyboards) will release all the handbrakes on the consist as shown below. Pressing `<Shift+'>` (Shift plus apostrophe on English keyboards) will set all of the handbrakes. Cars without handbrakes will not have an entry in the handbrake column.

If the newly coupled cars are to be moved without using their air brakes and parked nearby, the brake pressure in their air hose may be left at zero: i.e. their hoses are not connected to the train's air hose. Before the cars are uncoupled in their new location, their handbrakes should be set. The cars will continue to report *State Emergency* while coupled to the consist because their BC value is zero; they will not have any braking. The locomotive brakes must be used for braking. If the cars are uncoupled while in motion, they will continue coasting.

If the brakes of the newly connected cars are to be controlled by the train's air pressure as part of the consist, their hoses must be joined together and to the train's air hose and their angle cocks set correctly. Pressing the Backslash key <\>) (in English keyboards; please check the keyboard assignments for other keyboards) connects the brake hoses between all cars that have been coupled to the engine and sets the intermediate angle cocks to permit the air pressure to gradually approach the same pressure in the entire hose. This models the operations performed by the train crew. The HUD display changes to show the new condition of the brake hose connections and angle cocks:



All of the hoses are now connected; only the angle cocks on the lead locomotive and the last car are closed as indicated by the –. The rest of the cocks are open (+) and the air hoses are joined together (all `I`) to connect to the air supply on the lead locomotive.

Upon connection of the hoses of the new cars, recharging of the train brake line commences. Open Rails uses a default charging rate of about 1 minute per every 12 cars. The HUD display may report that the consist is in *Emergency* state; this is because the air pressure dropped when the empty car brake systems were connected. Ultimately the brake pressures reach their stable values:

BRAKE INFORMATION
Main reservoir 140 psi

Car	Type	BrkCyl	BrkPipe	AuxRes	ErgRes	MRPipe	RetValve	TripleValve	Handbrk	Conn	AnglCock	BleedOff
0 - 0	1P	10 psi	90 psi	90 psi	90 psi			Release		T	A-	B+
0 - 1	1P	10 psi	90 psi	90 psi	90 psi			Release		I	A+	B+
32884 - 0	1P	0 psi	90 psi	90 psi	90 psi			Release	0%	I	A+	B+
32884 - 1	1P	0 psi	90 psi	90 psi	90 psi			Release	0%	I	A+	B+
32884 - 2	1P	0 psi	90 psi	90 psi	90 psi			Release	0%	I	A+	B-

If you don't want to wait for the train brake line to charge, pressing <Shift+/> (in English keyboards) executes *Brakes Initialize* which will immediately fully charge the train brakes line to the final state. However, this action is not prototypical and also does not allow control of the brake retainers.

The state of the angle cocks, the hose connections and the air brake pressure of individual coupled cars can be manipulated by using the F9 Train Operations Monitor, described *here*. This will permit more realistic shunting of cars in freight yards.

Uncoupling Cars

When uncoupling cars from a consist, using the F5 HUD Expanded Brake Display in conjunction with the F9 Train Operations Monitor display allows the player to set the handbrakes on the cars to be uncoupled, and to uncouple them without losing the air pressure in the remaining cars. Before uncoupling, close the angle cock at the rear of the car ahead of the first car to be uncoupled so that the air pressure in the remaining consist is not lost when the air hoses to the uncoupled cars are disconnected. If this procedure is not followed, the train braking system will go into *Emergency* state and will require pressing the <\> (backslash) key to connect the air hoses correctly and then waiting for the brake pressure to stabilize again.

Setting Brake Retainers

If a long consist is to be taken down a long or steep grade the operator may choose to set the *Brake Retainers* on some or all of the cars to create a fixed braking force by those cars when the train brakes are released. (This requires that the retainer capability of the cars be enabled; either by the menu option *Retainer valve on all cars*, or by the inclusion of an appropriate keyword in the car's .wag file.) The train must be fully stopped and the main brakes must be applied so that there is adequate pressure in the brake cylinders. Pressing <Shift+]> controls how many cars in the consist have their retainers set, and the pressure value that is retained when the train brakes are released. The settings are described in *Brake Retainers* below. Pressing <Shift+[> cancels the settings and exhausts all of the air from the brake cylinders when the brakes are released. The F5 display shows the symbol *RV ZZ* for the state of the retainer valve in all cars, where ZZ is: *EX* for *Exhaust* or *LP* or *HP*. When the system brakes are released and there are no retainers set, the air in the brake cylinders in the cars is normally released to the air. The BC pressure for the cars with retainers set will not fall below the specified value. In order to change the retainer settings, the train must be fully stopped. A sample F5 view with 50% LP is shown below:

BRAKE INFORMATION
Main reservoir 140 psi

Car	Type	BrkCyl	BrkPipe	AuxRes	ErgRes	MRPipe	RetValve	TripleValve	Handbrk	Conn	AnglCock	BleedOff
0 - 0	1P	5 psi	90 psi	90 psi	90 psi			Release		T	A-	B+
0 - 1	1P	5 psi	90 psi	90 psi	90 psi			Release		I	A+	B+
32884 - 0	1P	0 psi	90 psi	90 psi	90 psi		EX	Release	0%	I	A+	B+
32884 - 1	1P	0 psi	90 psi	90 psi	90 psi		LP	Release	0%	I	A+	B+
32884 - 2	1P	0 psi	90 psi	90 psi	90 psi		EX	Release	0%	I	A+	B+
32884 - 3	1P	0 psi	90 psi	90 psi	90 psi		LP	Release	0%	I	A+	B+
32884 - 4	1P	0 psi	90 psi	90 psi	90 psi		EX	Release	0%	I	A+	B+
32884 - 5	1P	0 psi	90 psi	90 psi	90 psi		LP	Release	0%	I	A+	B+

8.4.6 Dynamic Brakes

Open Rails software supports dynamic braking for engines. To increase the Dynamic brakes press Period (.) and Comma (,) to decrease them. Dynamic brakes are usually off at train startup (this can be overridden by the related MSTS setting in the .eng file), the throttle works and there is no value shown in the dynamic brake line in the HUD. To turn on dynamic brakes set the throttle to zero and then press Period. Pressing Period successively increases the Dynamic braking forces. If the value n in the MSTS parameter DynamicBrakesDelayTimeBeforeEngaging (n) is greater than zero, the dynamic brake will engage only after n seconds. The throttle will not work when the Dynamic brakes are on.

The Dynamic brake force as a function of control setting and speed can be defined in a DynamicBrakeForceCurves table that works like the *MaxTractiveForceCurves table*. If there is no DynamicBrakeForceCurves defined in the ENG file, than one is created based on the MSTS parameter values.

8.4.7 Native Open Rails Braking Parameters

Open Rails has implemented additional specific braking parameters to deliver realism in braking performance in the simulation.

Following are a list of specific OR parameters and their default values. The default values are used in place of MSTS braking parameters; however, two MSTS parameters are used for the release state: MaxAuxilaryChargingRate and EmergencyResChargingRate.

- `Wagon(BrakePipeVolume` – Volume of car's brake pipe in cubic feet (default .5). This is dependent on the train length calculated from the ENG to the last car in the train. This aggregate factor is used to approximate the effects of train length on other factors. Strictly speaking this value should depend on the car length, but the Open Rails Development team doesn't believe it is worth the extra complication or CPU time that would be needed to calculate it in real time. We will let the community customize this effect by adjusting the brake servicetimefactor instead, but the Open Rails Development team doesn't believe this is worth the effort by the user for the added realism.
- `Wagon(ORTSEmergencyValveActuationRate` – Threshold rate for emergency brake actuation of the triple valve. If the pressure in the brake pipe decreases at a higher rate than specified, the triple valve will switch to emergency mode.
- `Wagon(ORTSMainResPipeAuxResCharging` – Boolean value that indicates, for twin pipe systems, if the main reservoir pipe is used for charging the auxiliary reservoirs. If set to false, the main reservoir pipe will not be used by the brake system.
- `Engine(ORTSMainResChargingRate` – Rate of main reservoir pressure change in psi per second when the compressor is on (default .4).
- `Engine(ORTSEngineBrakeReleaseRate` – Rate of engine brake pressure decrease in psi per second (default 12.5).
- `Engine(ORTSEngineBrakeApplicationRate` – Rate of engine brake pressure increase in psi per second (default 12.5).
- `Engine(ORTSBrakePipeChargingRate` – Rate of lead engine brake pipe pressure increase in PSI per second (default 21).
- `Engine(ORTSBrakePipeQuickChargingRate` – Rate of lead engine brake pipe pressure increase in PSI per second during a quick release (by default will be equal to ORTSBrakePipeChargingRate).
- `Engine(ORTSBrakeServiceTimeFactor` – Time in seconds for lead engine brake pipe pressure to drop to about 1/3 for service application (default 1.009).
- `Engine(ORTSBrakeEmergencyTimeFactor` – Time in seconds for lead engine brake pipe pressure to drop to about 1/3 in emergency (default .1).
- `Engine(ORTSBrakePipeTimeFactor` – Time in seconds for a difference in pipe pressure between adjacent cars to equalize to about 1/3 (default .003).

- `Engine(AirBrakeMaxMainResPipePressure` – Pressure in Main Reservoir Pipe for twin pipe braking systems (default = Main Reservoir Pressure).

8.4.8 Brake Retainers

The retainers of a car will only be available if either the General Option *Retainer valve on all cars* is checked, or the car's .wag file contains a retainer valve declaration. To declare a retainer the line `BrakeEquipmentType ( )` in the .wag file must include either the item `Retainer_4_Position` or the item `Retainer_3_Position`. A 4 position retainer includes four states: exhaust, low pressure (10 psi), high pressure (20 psi), and slow direct (gradual drop to zero). A 3 position retainer does not include the low pressure position. The use and display of the retainers is described in *Extended HUD for Brake Information*.

The setting of the retained pressure and the number of retainers is controlled using the Ctrl+[and Ctrl+] keys (Ctrl plus the left and right square bracket ([and]) keys on an English keyboard). The Ctrl+[key will reset the retainer on all cars in the consist to exhaust (the default position). Each time the Ctrl+] key is pressed the retainer settings are changed in a defined sequence. First the fraction of the cars set at a low pressure is selected (25%, 50% and then 100% of the cars), then the fraction of the cars at a high pressure is selected instead, then the fraction at slow direct. For the 25% setting the retainer is set on every fourth car starting at the rear of the train, 50% sets every other car and 100% sets every car. These changes can only be made when the train is stopped. When the retainer is set to exhaust, the ENG file release rate value is used, otherwise the pressures and release rates are hard coded based on some AB brake documentation used by the Open Rails development team.

8.4.9 Emergency Brake Application Key

The *Backspace* key is used, as in MSTS, to apply the train brakes in an emergency situation without requiring operation of the train brake lever. However in OR moving the brake lever back to the Release position will only cause OR to report *Apply Emergency Brake Push Button*. The Backspace key must be pressed again to cancel the emergency application, then normal operation can be resumed. When the button is active, the F5 HUD will display *Emergency Brake Push Button* in the *Train Brake* line.

8.4.10 Automatic Vacuum Brakes

Automatic Vacuum braking has been implemented in Open Rails in one of the two following forms:

- Direct Vacuum - in this form, while ever the Brake Pipe (BP) is connected to the ejectors or vacuum pump, depending upon the operating capacity of the ejectors, a vacuum will be maintained or created. Typically this will be when the brake controller is in a Brake Off position.
- Equalising Reservoir (EQ) - in this form a main vacuum reservoir is fitted to the locomotive, along with the equalising reservoir. Typically the main reservoir is maintained at a sufficiently high enough vacuum to create the vacuum in the BP to release the brakes. The BP vacuum will equalise at the vacuum set by the driver on the equalising reservoir.

As the altitude at which the train is operating increases, so the effectiveness of vacuum brakes decreases. For example, if a train is operating with a 21InHg system, based upon the following railway highest points in the countries indicated, it would be expect that only the following maximum possible vacuum levels would be achieveable:

UK = 350m = 20InHg Aus = 923m = 19InHg USA = 4,301m = 14InHg

In general, brakes (in particular a system with an equalising reservoir) will have three potential timings that impact the application or the releasing of the brakes.

i) In the equalising reservoir as the brake controller is varied

ii) In the train brake pipe as the vacuum is increased or decreased

iii) In the brake cylinder as it is applied or released.

In the case of brakes without an equalising reservoir only items ii) and iii) are valid in the above list.

The OR code attempts to model the above three items, however some compromises may need to be made, and it is suggested that the best outcome will be achieved when an overall timing approach is considered, rather than considering each of the individual components in isolation.

To enable the Equalising Reservoir option above BrakesTrainBrakeType must be set to `vacuum_single_pipe_eq` in the engine section of the ENG file.

Following is a list of specific OR parameters and their default values. The default values can be overwritten by including the following parameters into the relevant wagon section of the WAG or ENG file.

- `wagon(BrakePipeVolume` – Volume of brake pipe fitted to car in cubic feet (default calculated from car length, and assumption of 2in BP).
- `wagon(ORTSAuxilaryResCapacity` – Volume of auxiliary vacuum reservoir (coupled to brake cylinder) in cubic feet (default calculated on basis of 24in reservoir).
- `wagon(ORTSBrakeCylinderSize` – Size of brake cylinders fitted to wagon in inches (default assumes a 18in brake cylinder).
- `wagon(ORTSNumberBrakeCylinders` – Number of brake cylinders fitted to wagon, as an integer number (default 2).
- `wagon(ORTSDirectAdmissionValve` – Car has direct admission valves fitted, 0 = No, 1 = Yes (default No).
- `wagon(ORTSBrakeShoeFriction` – defines the friction curve for the brake shoe with speed (default curve for cast iron brake shoes included in OR).

Other standard brake parameters such as `MaxBrakeForce`, `MaxReleaseRate`, `MaxApplicationRate`, `BrakeCylinderPressureForMaxBrakeBrakeForce` can be used as well.

Additionaly the following are defined in the engine section of the ENG file:

- `engine(BrakeCylinderPressureForMaxBrakeBrakeForce` – sets the rate at which the brake pipe charges in InHg per second (default 0.32) This value should be calculated on the basis of feeding into a 200ft^3 brake system, as OR will adjust the value depending upon the connected volume of the brake cylinders and brake pipe.
- `engine(ORTSBrakeServiceTimeFactor` – Time for lead engine brake pipe pressure to drop in seconds (default 10.0)
- `engine(ORTSBrakeEmergencyTimeFactor` – Time for lead engine brake pipe pressure to drop under emergency conditions, in seconds (default 1.0)
- `engine(ORTSBrakePipeTimeFactor` – Controls propagation increase time along train pipe as vacuum increases, ie when brakes released, in seconds (default 0.02)
- `engine(TrainPipeLeakRate` – Rate at which the train brake pipe leaks at, in InHg per second (default no leakage)
- `engine(ORTSVacuumBrakesMainResVolume` – The volume of the main vacuum brake reservoir in cubic feet (default 110.0 , EQ operation only)
- `engine(ORTSVacuumBrakesMainResMaxVacuum` – The maximum vacuum in the main vacuum brake reservoir. When this pressure is reached the exhauster will automatically stop running, in InHg. (default 23 , EQ operation only)
- `engine(ORTSVacuumBrakesExhausterRestartVacuum` – pressure below which the exhauster will start to operate to recharge the main reservoir, in InHg (default 21 , EQ operation only)
- `engine(ORTSVacuumBrakesMainResChargingRate` – rate at which the main vacuum reservoir charges at, in InHg per second (default 0.2, EQ operation only)

Note: It is strongly recommended that UoM be used whenever units such as InHg, etc are specificed in the above parameters.

Other standard brake parameters such as VacuumBrakesHasVacuumPump, VacuumBrakesMinBoilerPressureMaxVacuum, VacuumBrakesSmallEjectorUsageRate, VacuumBrakesLargeEjectorUsageRate can be defined as well.

When defining the Brake Controllers for vacuum braked locomotives, only the following BrakesController tokens should be used - TrainBrakesControllerFullQuickReleaseStart, TrainBrakesControllerReleaseStart, TrainBrakesControllerRunningStart, TrainBrakesControllerApplyStart, TrainBrakesControllerHoldLappedStart, TrainBrakesControllerVacuumContinuousServiceStart, TrainBrakesControllerEmergencyStart, EngineBrakesControllerReleaseStart, EngineBrakesControllerRunningStart, EngineBrakesControllerApplyStart.

If TrainPipeLeakRate has been set in the ENG file, then the small ejector will be required to offset the leakage in the Brake Pipe. The *J* and *Shft-J* keys can be used to increase/decrease the level of operation of the small ejector.

An engine controller can be configured to customise the operation of the small ejector. This controller is called ORTSSmallEjector (w, x, y, z), and will be set up as a standard 4 value controller.

An engine controller can also be configured to customise the operation of the large ejector. This controller is called ORTSLargeEjector (w, x, y, z), and will be set up as a standard 4 value controller. The large ejector needs to be operated to release the brakes. The *Alt-J* and *Ctrl-J* keys can be used to decrease/increase the level of operation of the large ejector.

In diesel and electric locomotives, the Vacuum Exhauster preforms a similar function to the small and large ejector, but in an "automated" fashion. The *J* key can be used to run the vacuum exhauster at high speed to facilitate a quicker release of the brakes. An engine controller called ORTSFastVacuumExhauster (x y z), and will be set up as a standard 3 value controller.

If it is not desired to operate the large ejector, a simplified brake operation can be used by selecting the "Simple Contol and Physics" option in the options menu (Simulator TAB). This option can also be used if there is a "mismatch" between the locomotive and car brakes to set a standard default set of brakes.

Engine brakes can also be configured for locomotives as required. They will work in a similar fashion to those fitted to air braked locomotives.

8.4.11 Non Automatic Vacuum Brakes

Non automatic (or Straight) vacuum braking has been added to OR. This braking has been based upon the Eames and Hardy vacuum braking systems.

Straight brakes operate in the reverse way to "normal" vacuum brakes, ie the train brake pipe needs to have a vacuum created in it to apply the brakes, and air needs to be allowed into the brake pipe to release the brakes. The straight brake had the major disadvantage that if the brake pipe was interupted then the brakes would not be able to be applied and stop the train. Consequently they were only mostly applied to early trains and were replaced over time by automatic brakes.

To configure a car with manual braking then in the car brake section configure the fllowing two parameters:

```
BrakeEquipmentType( "Straight_Vacuum_Single_Pipe" )
```

Additional train controllers have been added to facilitate the operation of straight braked controlled cars.

TrainBrakesControllerStraightBrakingReleaseOffStart - closes the Eames release valve TrainBrakesControllerStraightBrakingReleaseOnStart - opens the Eames release valve to release the brakes. The normal large ejector controls need to be used to apply the brakes (use *Alt-J* and *Ctrl-J* keys)

TrainBrakesControllerStraightBrakingReleaseStart - Hardy brake - release TrainBrakesControllerStraightBrakingLapStart - Hardy brake - hold at current vacuum level TrainBrakesControllerStraightBrakingApplyStart - Hardy brake - twin ejector - use large ejector to apply brakes on train only TrainBrakesControllerStraightBrakingApplyAllStart - Hardy brake - single ejector type - use large ejector to apply brakes TrainBrakesControllerStraightBrakingEmergencyStart

- Hardy brake - twin ejector type - use large and small ejector to apply brakes on train, locomotive and tender

Similar timing parameters to those used in the Vacuum Brake above are used in this brake type.

8.4.12 Manual Brakes

Manual braking is provided in OR to facilitate cars with no brakes fitted (for example Stephenson's Rocket locomotive initially had no brakes fitted). Alternatively some trains used manually operated brakes controlled by a brakeman. This feature allows for the creation of braking on selected cars along the train that are operated by a brakeman (for example some trains had brakes only on the locomotive and the brakevans (caboose) which would be operated to control the stopping of the train.

An additional engine controller has been added to facilitate the operation of all brakeman (manual braked) controlled cars.

The controller can be added to an ENG file in a similar fashion to an air brake or vacuum controller by using the following brake controller parameter: `TrainBrakesControllerManualBrakingStart`

To configure a car with manual braking then in the car brake section configure the following two parameters:

`BrakeSystemType ( "Manual_Braking" )` `BrakeEquipmentType( "Manual_brake, Handbrake" )`

If the BrakeEquipmentType is left out, OR will assume that no braking is fitted to the car.

The following values, in the wagon section of the file need to be set for a manually braked car:

- `MaxBrakeForce`
- `MaxReleaseRate`
- `MaxApplicationRate`

The manual brake can be increased by pressing *Alt-]*, and decreased by pressing *Alt-[*.

8.4.13 Steam Brakes

Steam brakes can be applied to a locomotive, and its corresponding tender, by adding the following parameter to the ENG file:

`BrakesEngineBrakeType ( "Steam_brake" )`

The brake can be applied by pressing], and released by pressing the [keys.

To control the application and release rates on the brake use the `EngineBrakesControllerMaxApplicationRate` and `EngineBrakesControllerMaxReleaseRate` parameters.

The `SteamBrakeFX` special effect, if added to the wagon, will turn on and off with the brake operation and can be used to model steam leakage of the steam brake cylinder, etc.

8.4.14 Wheel Slide Protection

Open Rails supports the use of Wheel Slide Protection (WSP) on trains with air brakes. WSP operates as described below.

During braking wheelslide control is effected throughout the train by additional equipment on each vehicle. In the piping to each pair of brake cylinders are fitted electrically operated dump valves. When axle rotations which are sensed electrically, differ by a predetermined speed the dump valves are operated releasing brake cylinder pressure to both axles of the affected bogie.

Dump valve operation will cease when differences in axle rotations are within specified limits or the axle accelerates faster than a specified rate. The dump valve will only operate for a maximum period of seven

seconds after which time it will be de-energised and the dump valve will not re-operate until the train has stopped or the throttle operated.

Dump valve operation is prevented under the following conditions:

- When the Power Controller is open.
- When Brake Pipe Pressure has been reduced below 36 psi (250 kPa).

To enable WSP `ORTSWheelBrakeSlideProtection ( 1 )`. If it is desired that emergency braking should not be impacted by WSP, then use the `ORTSEmergencyBrakingDisablesWSP ( 1 )` parameter.

When WSP is active the brake cylinder pressure reading will go yellow in the extended HuD on the BRAKE INFORMATION screen.

8.4.15 SME (sometimes also called SEM) Brake System

SME braking is a straight air-brake system having an automatic emergency feature by means of which the simplicity of the straight air brake is retained for service operation, but it also has the additional protection afforded by the automatic application of the brake in case of a break-in-two or the bursting of a hose. SME braking is typically used on short DMU rail sets. SME braking is a form of electro-pneumatic (EP) brake system, however EP and SME equipped cars cannot be mixed together in the same consist.

To activate SME braking, set `BrakeSystemType ( SME )`.

The following brake tokens can be used with it: `TrainBrakesControllerSMEOnlyStart` `TrainBrakesControllerSMEFullServiceStart` `TrainBrakesControllerSMEHoldStart` `TrainBrakesControllerSMEReleaseStart`

8.5 Dynamically Evolving Tractive Force

The Open Rails development team has been experimenting with max/continuous tractive force, where it can be dynamically altered during game play using the `ORTSMaxTractiveForceCurves` parameter as shown earlier. The parameters were based on the Handbook of Railway Vehicle Dynamics. This says the increased traction motor heat increase resistance which decreases current and tractive force. We used a moving average of the actual tractive force to approximate the heat in the motors. Tractive force is allowed to be at the maximum per the ENG file, if the average heat calculation is near zero. If the average is near the continuous rating than the tractive force is de-rated to the continuous rating. There is a parameter called `ORTSContinuousForceTimeFactor` that roughly controls the time over which the tractive force is averaged. The default is 1800 seconds.

8.6 Curve Resistance - Theory

8.6.1 Introduction

When a train travels around a curve, due to the track resisting the direction of travel (i.e. the train wants to continue in a straight line), it experiences increased resistance as it is *pushed* around the curve. Over the years there has been much discussion about how to accurately calculate curve friction. The calculation methodology presented (and used in OR) is meant to be representative of the impacts that curve friction will have on rolling stock performance.

8.6.2 Factors Impacting Curve Friction

A number of factors impact upon the value of resistance that the curve presents to the trains movement, as follows:

- Curve radius – the smaller the curve radius the higher the higher the resistance to the train
- Rolling Stock Rigid Wheelbase – the longer the rigid wheelbase of the vehicle, the higher the resistance to the train. Modern bogie stock tends to have shorter rigid wheelbase values and is not as bad as the older style 4 wheel wagons.
- Speed – the speed of the train around the curve will impact upon the value of resistance, typically above and below the equilibrium speed (i.e. when all the wheels of the rolling stock are perfectly aligned between the tracks). See the section below *Impact of superelevation*.

The impact of wind resistance on curve friction is calculated in the general calculations for Wind Resistance.

8.6.3 Impact of Rigid Wheelbase

The length of the rigid wheelbase of rolling stock will impact the value of curve resistance. Typically rolling stock with longer rigid wheelbases will experience a higher degree of *rubbing* or frictional resistance on tight curves, compared to stock with smaller wheelbases.

Steam locomotives usually created the biggest problem in regard to this as their drive wheels tended to be in a single rigid wheelbase as shown in figure. In some instances on routes with tighter curve the *inside* wheels of the locomotive were sometimes made flangeless to allow them to *float* across the track head. Articulated locomotives, such as Shays, tended to have their drive wheels grouped in bogies similar to diesel locomotives and hence were favoured for routes with tight curves.

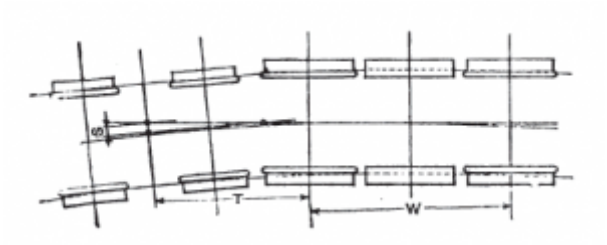


Fig. 1: Diagram Source: The Baldwin Locomotive Works – Locomotive Data – 1944 Example of Rigid Wheelbase in steam locomotive

The value used for the rigid wheelbase is shown as W in figure

8.6.4 Impact of Super Elevation

On any curve whose outer rail is super-elevated there is, for any car, one speed of operation at which the car trucks have no more tendency to run toward either rail than they have on straight track, where both rail-heads are at the same level (known as the equilibrium speed). At lower speeds the trucks tend constantly to run down against the inside rail of the curve, and thereby increase the flange friction; whilst at higher speeds they run toward the outer rail, with the same effect. This may be made clearer by reference to figure below, which represents the forces which operate on a car at its centre of gravity.

With the car at rest on the curve there is a component of the weight W which tends to move the car down toward the inner rail. When the car moves along the track centrifugal force `Fc` comes into play and the car action is controlled by the force `Fr` which is the resultant of `W` and `Fc`. The force `Fr` likewise has a component which, still tends to move the car toward the inner rail. This tendency persists until, with increasing speed, the value of `Fc` becomes great enough to cause the line of operation of `Fr` to coincide with the centre line of the track perpendicular to the plane of the rails. At this equilibrium speed there is no longer any tendency of the trucks to run toward either rail. If the speed be still further increased,

the component of `Fr` rises again, but now on the opposite side of the centre line of the track and is of opposite sense, causing the trucks to tend to move toward the outer instead of the inner rail, and thereby reviving the extra flange friction. It should be emphasized that the flange friction arising from the play of the forces here under discussion is distinct from and in excess of the flange friction which arises from the action of the flanges in forcing the truck to follow the track curvature. This excess being a variable element of curve resistance, we may expect to find that curve resistance reaches a minimum value when this excess reduces to zero, that is, when the car speed reaches the critical value referred to. This critical speed depends only on the super-elevation, the track gauge, and the radius of the track curvature. The resulting variation of curve resistance with speed is indicated in diagram below.

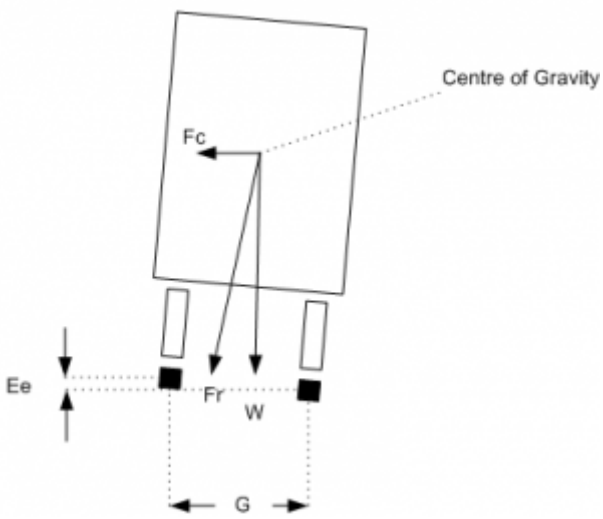


Fig. 2: Forces on rolling stock transitioning a curve

8.6.5 Calculation of Curve Resistance

R = W F (D + L) 2 r

Where:

- R = Curve resistance,
- W = vehicle weight,
- F = Coefficient of Friction,
  - 0.1 – 0.3 for wet rail
  - 0.5 for dry, smooth steel-to-steel
- D = track gauge,
- L = Rigid wheelbase,
- r = curve radius.

(Source: The Modern locomotive by C. Edgar Allen - 1912)

8.6.6 Calculation of Curve Speed Impact

The above value represents the least value amount of resistance, which occurs at the equilibrium speed, and as described above will increase as the train speed increases and decreases from the equilibrium speed. This concept is shown pictorially in the following graph. Open Rails uses the following formula to model the speed impact on curve resistance:

$$SpeedFactor = abs\left((v_{equilibrium} - v_{train}) \cdot v_{equilibrium}\right) \cdot ResistanceFactor_{start}$$

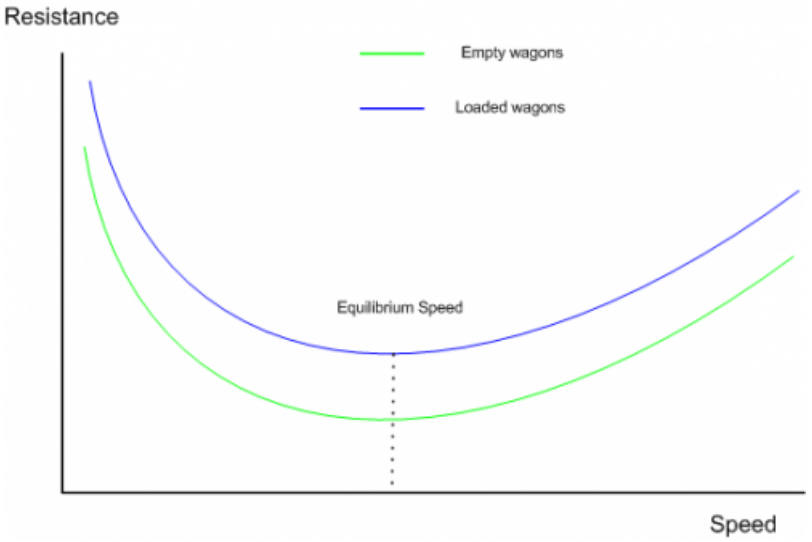


Fig. 3: Generalisation of Variation of Curve Resistance With Speed

8.6.7 Further background reading

http://en.wikipedia.org/wiki/Curve_resistance_(railroad)

8.7 Curve Resistance - Application in OR

Open Rails models this function, and the user may elect to specify the known wheelbase parameters, or the above *standard* default values will be used. OR calculates the equilibrium speed in the speed curve module, however it is not necessary to select both of these functions in the simulator options TAB. Only select the function desired. By studying the *Forces Information* table in the HUD, you will be able to observe the change in curve resistance as the speed, curve radius, etc. vary.

8.7.1 OR Parameter Values

Typical OR parameter values may be entered in the Wagon section of the .wag or .eng file, and are formatted as below.:

```
ORTSRigidWheelBase ( 3in )
ORTSTrackGauge ( 4ft 8.5in) // (also used in curve speed module)
```

8.7.2 OR Default Values

The above values can be entered into the relevant files, or alternatively if they are not present, then OR will use the default values described below.

Rigid Wheelbase – as a default OR uses the figures shown above in the *Typical Rigid Wheelbase Values* section. The starting curve resistance value has been assumed to be 200%, and has been built into the speed impact curves. OR calculates the curve resistance based upon the actual wheelbases provided by the player or the appropriate defaults. It will use this as the value at *Equilibrium Speed*, and then depending upon the actual calculated equilibrium speed (from the speed limit module) it will factor the resistance up as appropriate to the current train speed.

Steam locomotive wheelbase approximation – the following approximation is used to determine the default value for the fixed wheelbase of a steam locomotive.

$$WheelBase = 1.25 \cdot (axles - 1) \cdot DrvWheelDiameter$$

8.7.3 Typical Rigid Wheelbase Values

The following values are used as defaults where actual values are not provided by the player.

Rolling Stock Type	Typical value
Freight Bogie type stock (2 wheel bogie)	5’ 6” (1.6764m)
Passenger Bogie type stock (2 wheel bogie)	8’ (2.4384m)
Passenger Bogie type stock (3 wheel bogie)	12’ (3.6576m)
Typical 4 wheel rigid wagon	11’ 6” (3.5052m)
Typical 6 wheel rigid wagon	12’ (3.6576m)
Tender (6 wheel)	14’ 3” (4.3434m)
Diesel, Electric Locomotives	Similar to passenger stock
Steam locomotives	Dependent on drive wheels #. Can be up to 20’+, e.g. large 2–10–0 locomotives

Modern publications suggest an allowance of approximately 0.8 lb per ton (US) per degree of curvature for standard gauge tracks. At very slow speeds, say 1 or 2 mph, the curve resistance is closer to 1.0 lb (or 0.05% up grade) per ton per degree of curve.

8.8 Super Elevation (Curve Speed Limit) – Theory

8.8.1 Introduction

When a train rounds a curve, it tends to travel in a straight direction and the track must resist this movement, and force the train to move around the curve. The opposing movement of the train and the track result in a number of different forces being in play.

8.8.2 19th & 20th Century vs Modern Day Railway Design

In the early days of railway construction financial considerations were a big factor in route design and selection. Given that the speed of competing transport, such as horses and water transport was not very great, speed was not seen as a major factor in the design process. However as railway transportation became a more vital need for society, the need to increase the speed of trains became more and more important. This led to many improvements in railway practices and engineering. A number of factors, such as the design of the rolling stock, as well as the track design, ultimately influence the maximum speed of a train. Today's high speed railway routes are specifically designed for the speeds expected of the rolling stock.

8.8.3 Centrifugal Force

Railway locomotives, wagons and carriages, hereafter referred to as rolling stock, when rounding a curve come under the influence of centrifugal force. Centrifugal force is commonly defined as:

- The apparent force that is felt by an object moving in a curved path that acts outwardly away from the centre of rotation.
- An outward force on a body rotating about an axis, assumed equal and opposite to the centripetal force and postulated to account for the phenomena seen by an observer in the rotating body.

For this article the use of the phrase centrifugal force shall be understood to be an apparent force as defined above.

8.8.4 Effect of Centrifugal Force

When rolling stock rounds a curve, if the rails of the track are at the same elevation (i.e. the two tracks are at the same level) the combination of centrifugal force Fc and the weight of the rolling stock W will produce a resulting force Fr that does not coincide with the centre line of track, thus producing a downward force on the outside rail of the curve that is greater than the downward force on the inside rail (Refer to Figure 1). The greater the velocity and the smaller the radius of the curve (some railways have curve radius as low as 100m), the farther the resulting force Fr will move away from the centre line of track. Equilibrium velocity was the velocity at which a train could negotiate a curve with the rolling stock weight equally distributed across all the wheels.

If the position of the resulting force Fr approaches the outside rail, then the rolling stock is at risk of *falling* off the track or overturning. The following drawing, illustrates the basic concept described. Lateral displacement of the centre of gravity permitted by the suspension system of the rolling stock is not illustrated.

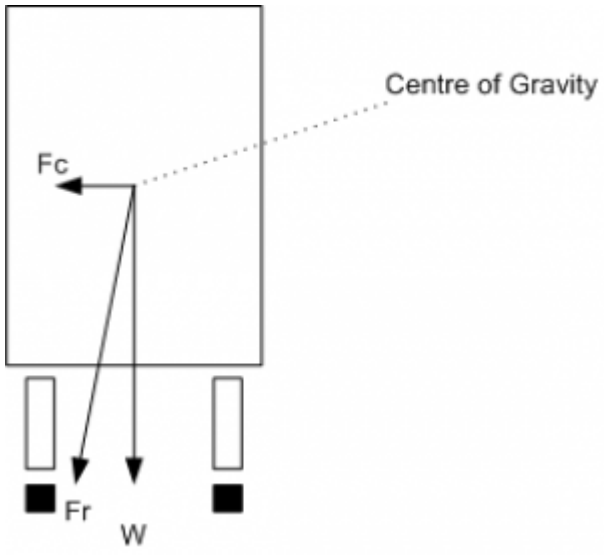


Fig. 4: Forces at work when a train rounds a curve

8.8.5 Use of Super Elevation

In order to counteract the effect of centrifugal force Fc the outside rail of the curve may be elevated above the inside rail, effectively moving the centre of gravity of the rolling stock laterally toward the inside rail.

This procedure is generally referred to as super elevation. If the combination of lateral displacement of the centre of gravity provided by the super elevation, velocity of the rolling stock and radius of curve is such that resulting force Fr becomes centred between and perpendicular to a line across the running rails the downward pressure on the outside and inside rails of the curve will be the same. The super elevation that produces this condition for a given velocity and radius of curve is known as the balanced or equilibrium elevation.

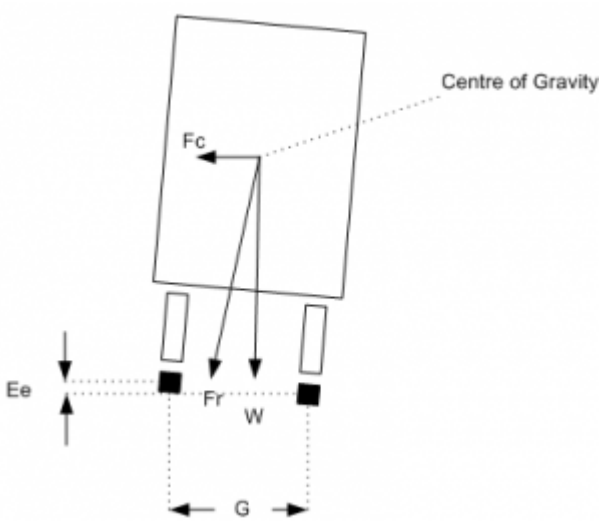


Fig. 5: This illustrates the concept.

8.8.6 Limitation of Super Elevation in Mixed Passenger & Freight Routes

Typical early railway operation resulted in rolling stock being operated at less than equilibrium velocity (all wheels equally sharing the rolling stock weight), or coming to a complete stop on curves. Under such circumstances excess super elevation may lead to a downward force sufficient to damage the inside rail of the curve, or cause derailment of rolling stock toward the centre of the curve when draft force is applied to a train. Routine operation of loaded freight trains at low velocity on a curve superelevated to permit operation of higher velocity passenger trains will result in excess wear of the inside rail of the curve by the freight trains.

Thus on these types of routes, super elevation is generally limited to no more than 6 inches.

8.8.7 Limitation of Super Elevation in High Speed Passenger Routes

Modern high speed passenger routes do not carry slower speed trains, nor expect trains to stop on curves, so it is possible to operate these routes with higher track super elevation values. Curves on these types of route are also designed with a relatively gentle radius, and are typically in excess of 2000m (2km) or 7000m (7km) depending on the speed limit of the route.

Parameters	France	Germany	Spain	Korea	Japan
Speed (km/h)	300/350	300	350	300/350	350
Horizontal curve radius (m)	10000 (10km)	7000 (7km)	7000 (7km)	7000 (7km)	4000 (4km)
Super elevation (mm)	180	170	150	130	180
Max Grade (mm/m)	35	40	12.5	25	15
Cant Gradient (mm/s)	50	34.7	32	N/A	N/A
Min Vertical radius (m)	16000 (16km)	14000 (14km)	24000 (24km)	N/A	10000 (10km)

Table: Curve Parameters for High Speed Operations (Railway Track Engineering by J. S. Mundrey)

8.8.8 Maximum Curve Velocity

The maximum velocity on a curve may exceed the equilibrium velocity, but must be limited to provide a margin of safety before overturning velocity is reached or a downward force sufficient to damage the outside rail of the curve is developed. This velocity is generally referred to as maximum safe velocity or safe speed. Although operation at maximum safe velocity will avoid overturning of rolling stock or rail damage, a passenger riding in a conventional passenger car will experience centrifugal force perceived as a tendency to slide laterally on their seat, creating an uncomfortable sensation of instability. To avoid passenger discomfort, the maximum velocity on a curve is therefore limited to what is generally referred to as maximum comfortable velocity or comfortable speed. Operating experience with conventional passenger cars has led to the generally accepted practice, circa 1980, of designating the maximum velocity for a given curve to be equal to the result for the calculation of equilibrium velocity with an extra amount added to the actual super elevation that will be applied to the curve. This is often referred to as unbalanced super elevation or cant deficiency. Tilt trains have been introduced to allow faster train operation on tracks not originally designed for *high speed* operation, as well as high speed railway operation. The tilting of the passenger cab allows greater values of unbalanced super elevation to be used.

8.8.9 Limitation of Velocity on Curved Track at Zero Cross Level

The concept of maximum comfortable velocity may also be used to determine the maximum velocity at which rolling stock is permitted to round curved track without super elevation and maintained at zero cross level. The lead curve of a turnout located between the heel of the switch and the toe of the frog is an example of curved track that is generally not super elevated. Other similar locations would include yard tracks and industrial tracks where the increased velocity capability made possible by super elevation is not required. In such circumstances the maximum comfortable velocity for a given curve may also be the maximum velocity permitted on tangent track adjoining the curve.

8.8.10 Height of Centre of Gravity

Operation on a curve at equilibrium velocity results in the centre of gravity of the rolling stock coinciding with a point on a line that is perpendicular to a line across the running rails and the origin of which is midway between the rails. Under this condition the height of the centre of gravity is of no consequence as the resulting force Fr coincides with the perpendicular line described above. When rolling stock stops on a super elevated curve or rounds a curve under any condition of non-equilibrium the resulting force Fr will not coincide with the perpendicular line previously described and the height of the centre of gravity then becomes significant in determining the location of the resulting force Fr relative to the centre line of the track. The elasticity of the suspension system of rolling stock under conditions of non-equilibrium will introduce a roll element that affects the horizontal displacement of the centre of gravity and that must also be considered when determining the location of the resulting force Fr.

8.8.11 Calculation of Curve Velocity

The generic formula for calculating the various curve velocities is as follows:

$$v = \sqrt{E \cdot g \cdot r \cdot G}$$

Where:

- E = Ea (track super elevation) + Ec (unbalanced super elevation)
- g = acceleration due to gravity
- r = radius of curve
- G = track gauge

8.8.12 Typical Super Elevation Values & Speed Impact – Mixed Passenger & Freight Routes

The values quoted below are "typical" but may vary from country to country.

Track super elevation typically will not be more than 6 inches (150mm). Naturally, depending upon the radius of the curve, speed restrictions may apply.

Normally unbalanced super elevation is typically restricted to 3 inches (75mm), and is usually only allowed for passenger stock.

Tilt trains may have values of up to 12 inches (305mm).

8.8.13 Typical Super Elevation Values & Speed Impact – High Speed Passenger Routes

	Cant D (SuperElevation) (mm)	Cant deficiency (Unbalanced SuperElevation) I (mm)
CEN (draft) – Tilting trains	180–200	300
Czech Rep. – Tilting trains	150	270
France – Tilting trains	180	260
Germany – Tilting trains	180	300
Italy – Tilting trains	160	275
Norway – Tilting trains	150	280
Spain – Tilting trains (equivalent for standard gauge)	160 (139)	210 (182)
Sweden – Tilting trains	150	245
UK – Tilting trains	180	300

Table: Super Elevation limits (source - Tracks for tilting trains - A study within the Fast And Comfortable Trains (FACT) project by B. Kufver, R. Persson)

8.9 Super Elevation (Curve Speed Limit) Application in OR

Open Rails implements this function, and has *standard* default values applied. The user may elect to specify some of the standard parameters used in the above formula.

8.9.1 OR Super Elevation Parameters

Typical OR parameters can be entered in the Wagon section of the .wag or .eng file, and are formatted as below.

```
ORTSUnbalancedSuperElevation ( 3in )
ORTSTrackGauge( 4ft 8.5in)
```

8.9.2 OR Super Elevation Default Values

The above values can be entered into the relevant files, or alternatively OR will default to the following functionality.

OR will initially use the speed limit value from the route's .trk file to determine whether the route is a conventional mixed freight and passenger route or a high speed route.

- Speed limit < 200km/h (125mph) – Mixed Freight and Pass route
- Speed limit > 200km/h (125mph) – High speed passenger route

Default values of tracksuperelevation will be applied based upon the above classifications.

Track gauge will default to the standard value of 4' 8.5" (1435mm).

Unbalancedsuperelevation (Cant Deficiency) will be determined from the value entered by the user, or will default to the following values:

- Conventional Freight – 0" (0mm)
- Conventional Passenger – 3" (75mm)
- Engines & tenders – 6" (150mm)

Tilting trains require the addition of the relevant unbalancedsuperelevation information to the relevant rolling stock files.

8.10 Tunnel Friction – Theory

8.10.1 Introduction

When a train travels through a tunnel it experiences increased resistance to the forward movement.

Over the years there has been much discussion about how to accurately calculate tunnel resistance. The calculation methodology presented (and used in OR) is meant to provide an indicative representation of the impacts that tunnel resistance will have on rolling stock performance.

8.10.2 Factors Impacting Tunnel Friction

In general, the train aerodynamics are related to aerodynamic drag, pressure variations inside the train, train-induced flows, cross-wind effects, ground effects, pressure waves inside the tunnel, impulse waves at the exit of tunnel, noise and vibration, etc. The aerodynamic drag is dependent on the cross-sectional area of the train body, train length, the shape of train fore- and after-bodies, the surface roughness of train body, and geographical conditions around the traveling train. The train-induced flows can influence passengers on a subway platform and is also associated with the cross-sectional area of the train body, the train length, the shape of train fore- and after-bodies, surface roughness of train body, etc.

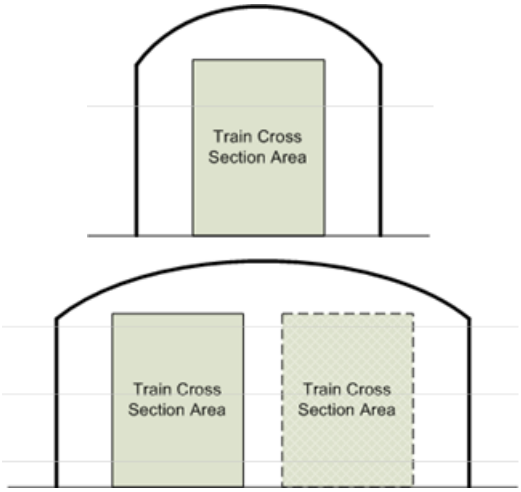
A high speed train entering a tunnel generates a compression wave at the entry portal that moves at the speed of sound in front of the train. The friction of the displaced air with the tunnel wall produces a pressure gradient and, as a consequence, a rise in pressure in front of the train. On reaching the exit portal of the tunnel, the compression wave is reflected back as an expansion wave but part of it exits the tunnel and radiates outside as a micro-pressure wave. This wave could cause a sonic boom that may lead to structural vibration and noise pollution in the surrounding environment. The entry of the tail of the train into the tunnel produces an expansion wave that moves through the annulus between the train and the tunnel. When the expansion pressure wave reaches the entry portal, it is reflected towards the interior of the tunnel as a compression wave. These compression and expansion waves propagate backwards and forwards along the tunnel and experience further reflections when meeting with the nose and tail of the train or reaching the entry and exit portals of the tunnel until they eventually dissipate completely.

The presence of this system of pressure waves in a tunnel affects the design and operation of trains, and they are a source of energy losses, noise, vibrations and aural discomfort for passengers.

These problems are even worse when two or more trains are in a tunnel at the same time. Aural comfort is one of the major factors determining the area of new tunnels or the maximum train speed in existing tunnels.

8.10.3 Importance of Tunnel Profile

As described above, a train travelling through a tunnel will create a bow wave of air movement in front of it, which is similar to a *piston effect*. The magnitude and impact of this effect will principally be determined by the **tunnel profile**, **train profile** and **speed**.



Typical tunnel profiles are shown in the diagrams.

As can be seen from these diagrams, the smaller the tunnel cross sectional area compared to the train cross sectional area, the less air that can *escape* around the train, and hence the greater the resistance experienced by the train. Thus it can be understood that a single train in a double track tunnel will experience less resistance then a single train in a single track tunnel.

8.10.4 Calculation of Tunnel Resistance

$$W_t = \frac{AL_{tr}}{(P+G)} v^2 \left(1 - \frac{1}{1 + \sqrt{\frac{B + C(L_t - L_{tr})}{L_{tr}}}} \right)^2$$

where

$$A = \frac{0.00003318 \cdot \rho \cdot F_t}{(1 - F_{tr}/F_t)^2},$$
$$B = 174.419(1 - F_{tr}/F_t)^2,$$
$$C = 2.907 \frac{(1 - F_{tr}/F_t)^2}{4F_t/R_t}.$$

F_t – tunnel cross-sectional area (m^2)	F_{tr} – train cross-sectional area (m^2)
ρ – density of air (= 1.2 kg/m^3)	R_t – tunnel perimeter (m)
L_{tr} – length of train (m)	L_t – length of tunnel (m)
v – train velocity (m/s)	P – locomotive mass (t)
W_t – additional aerodynamic drag in tunnel (N/kN)	G – train mass (t)

Source: Reasonable compensation coefficient of maximum gradient in long railway tunnels by Sirong YI*, Liangtao NIE, Yanheng CHEN, Fangfang QIN

8.11 Tunnel Friction – Application in OR

Tunnel resistance is designed to model the relative impact on the current train, and does not take into account multiple trains in the tunnel at the same time.

Tunnel resistance values can be seen in the *Train Forces HUD*.

The default tunnel profile is determined by the route speed recorded in the TRK file.

8.11.1 OR Parameters

The following parameters maybe included in the TRK file to overwrite standard default values used by Open Rails:

- `ORTSSingleTunnelArea ( x )` – Cross section area of single track tunnel – units area
- `ORTSSingleTunnelPerimeter ( x )` – Perimeter of single track tunnel – units distance
- `ORTSDoubleTunnelArea ( x )` – Cross section area of double track tunnel – units area
- `ORTSDoubleTunnelPerimeter ( x )` – Perimeter of double track tunnel – units distance

To insert these values in the .trk file, it is suggested that you add them just prior to the last parenthesis. You may also use an *Include file* method, described *here*.

8.11.2 OR Defaults

Open Rails uses the following standard defaults, unless overridden by values included in the TRK file.

Speed	1 track	2 tracks
Tunnel Perimeter		
< 160 km/h	21.3 m	31.0 m
160 < 200 km/h	25.0 m	34.5 m
200 < 250 km/h	28.0 m	35.0 m
250 < 350 km/h	32.0 m	37.5 m
Tunnel Cross Sectional Area		
< 120 km/h	27.0 m^2	45.0 m^2
< 160 km/h	42.0 m^2	76.0 m^2
200 km/h	50.0 m^2	80.0 m^2
250 km/h	58.0 m^2	90.0 m^2
350 km/h	70.0 m^2	100.0 m^2

8.12 Wind Resistance

The default Davis resistance formula is only valid for train operation in STILL air. At high train speeds, and especially for Very Fast trains the impact of wind can be quite significant, and special consideration is required when designing rolling stock, etc. If wind is present, then the impact of drag forces on the train will vary, and be in addition to the values calculated in the default (or still air) conditions.

The wind resistance in OR is modeled by the following two components:

Wind Drag Resistance - If a train is heading into a headwind then the train will experience greater resistance to movement, similarly if the train has a tailwind, then the trains resistance will decrease as the wind provides a "helping hand". As the wind swings from the head of the train to the rear resistance will decrease. When the wind is perpendicular to the train, drag impact due to the wind will be zero.

Wind Lateral Force Resistance - When the wind blows from the side of the train, the train will be pushed against the outside track rail, thus increasing the amount of resistance experienced by the train.

To activate calculation of wind resistance, select the tickbox for "Wind dependent resistance" in the Simulation TAB of the options menu. As wind only becomes significant at higher train speeds, the wind resistance calculation only commences once the train speed exceeds 5 mph.

The amount of wind resistance that the train is experiencing is shown in the FORCES INFORMATION HUD. (see attached screenshot) The current wind conditions are also shown in the HUD, and include the Wind speed and direction, train direction, and the resulting vectors for the combined train and wind speed. The value in the Friction column is the default still air conditions as calculated by the Davis formula. It should be noted that OR calculates the Wind Drag resistance as a difference compared to the still air Davis C value, and hence it is possible for values in the Wind column to go negative on occasions. This is most likely when the wind is blowing from the rear of the train, ie the ResWind direction is greater then 90°C degrees, and hence the wind is actually aiding the train movement, and in effect reducing the amount of still air resistance.

The wind model has been adjusted in the following way:

- Wind Update speed - 1 sec
- Wind direction will always be within +/- 45°C degrees of the randomly selected default value selected at startup
- Wind speed is limited to approx 10mph.

The Wind Resistance model will use default information, such as the width and height of the stock from the Size statement, so by default it is not necessary to add any additional parameters for its operation. However for those who like to customise, the following parameters can be inputted via the WAG file or section.

`ORTSWagonFrontalArea` – The frontal cross sectional area of the wagon. The default units are in ft^2, so if entering metres, include the Units of Measure.

`ORTSDavisDragConstant` – OR by default uses the standard Davis Drag constants. If alternate drag constants are used in calculating the still air resistance, then it might be worthwhile inputting these values.

8.13 Trailing Locomotive Resistance

Typically only one set of resistance parameters is allowed for each WAG file. In the case of locomotives this can create issues as a leading locomotive will have a higher drag resistance then a trailing locomotive.

OR automatically adjusts the Drag resistance for trailing locomotives based upon the ratio of the original Davis formula.

However for those who like to customise, the following parameter can be inputted via the WAG file or section.

`ORTSTrailLocomotiveResistanceFactor` – The constant value by which the leading locomotive resistance needs to be decreased for trailing operation.

For steam locomotive tenders it may be necessary to enter this value depending upon the Drag constant used to calculate the tender resistance.

8.14 OR-Specific *Include Files* for Modifying MSTS File Parameters

8.14.1 Modifications to .eng and .wag Files

In the preceding paragraphs many references have been made to OR-specific parameters and tables to be included in .eng and .wag files. MSTS is in general quite tolerant if it finds unknown parameters and even blocks within .eng and .wag files, and continues running normally. However this way of operating is not encouraged by the OR team. Instead, a cleaner approach, as described here, has been implemented.

Within the trainset folder containing the .eng and .wag files to be upgraded, create a subfolder named `OpenRails`. Only OR will read files from this folder. Within this subfolder a text file named xxxx.eng or xxxx.wag, where xxxx.eng or xxxx.wag is the name of the original file, must be created.

This new file may contain either:

- all of the information included in the original file (using (modified parts where desired) plus the OR-specific parts if any, or:
- at its beginning only an *include* reference to the original file, followed by the modified parts and the OR-specific parts. This does not apply to the `Name()` statement and the Loco Description Information, where in any case the data in the base .eng file is retained.

An example of an OR-specific `bc13ge70tonner.eng` file to be placed into the OpenRails subfolder that uses the second possibility is as follows:

```
include ( ../bc13ge70tonner.eng )
Wagon (
  MaxReleaseRate ( 2.17 )
  MaxApplicationRate ( 3.37 )
  MaxAuxilaryChargingRate ( .4 )
  EmergencyResChargingRate ( .4 )
  BrakePipeVolume ( .4 )
  ORTSUnbalancedSuperElevation ( 3in )
Engine (
  AirBrakeMainresvolume ( 16 )
  MainResChargingRate ( .5 )
  BrakePipeChargingRate ( 21 )
  EngineBrakeReleaseRate ( 12.5 )
  EngineBrakeApplicationRate ( 12.5 )
  BrakePipeTimeFactor ( .00446 )
  BrakeServiceTimeFactor ( 1.46 )
  BrakeEmergencyTimeFactor ( .15 )
  ORTSMaxTractiveForceCurves (
    0 (
      0 0 50 0 )
    .125 (
      0 23125
      .3 23125
      1 6984
      2 3492
      5 1397
      10 698
      20 349
      50 140 )
    .25 (
      0 46250
      .61 46250
      1 27940
      2 13969
      5 5588
      10 2794
      20 1397
      50 559 )
    .375 (
      0 69375
      .91 69375
      2 31430
      5 12572
```

(continues on next page)

(continued from previous page)

```
      10 6287
      20 3143
      50 1257 )
    .5 (
      0 92500
      1.21 92500
      5 22350
      10 11175
      20 5588
      50 2235 )
    .625 (
      0 115625
      1.51 115625
      5 34922
      10 17461
      20 8730
      50 3492 )
    .75 (
      0 138750
      1.82 138750
      5 50288
      10 25144
      20 12572
      50 5029 )
    .875 (
      0 161875
      2.12 161875
      5 68447
      10 34223
      20 17112
      50 6845 )
    1 (
      0 185000
      2.42 185000
      5 89400
      10 44700
      20 22350
      50 8940 )
    )
  )
)
```

Take into account that the first line must be blank (before the include line).

The `ORTSMaxTractiveForceCurves` are formed by blocks of pairs of parameters representing speed in metres per second and tractive force in Newtons; these blocks are each related to the value of the throttle setting present at the top of each block. For intermediate values of the speed an interpolated value is computed to get the tractive force, and the same method applies for intermediate values of the throttle.

If the parameter that is modified for OR is located within a named (i.e. bracketed) block in the original file, then in the OpenRails file it must be included in a matching bracketed block. For instance, it is not possible to replace only a part of the `Lights()` block. It must be replaced in its entirety. For example, to use a different `Cabview()`, it must be enclosed in an `Engine` block:

```
Engine ( BNSF4773
    CabView ( dash9OR.cvf )
)
```

This is also required in the case of certain Brake parameters; to correctly manage reinitialization of brake parameters, the entire block containing them must be present in the .eng file in the OpenRails folder.

This use of the `Include` command can be extended to apply to sections of groups of .wag or .eng files that the user wishes to replace by a specific block of data – the parameters can be provided by a text file located outside the usual MSTS folders; e.g. brake parameters.

8.15 Common locomotive subsystems

8.15.1 Battery switch

The battery switch controls the low voltage power supply of the locomotive. If the low voltage power supply is disabled, all of the systems of the locomotive are disabled (for example, the circuit breaker opens and the pantograph falls down).

The battery switch of all locomotives in a consist can be controlled by *Control Battery Switch Close* and *Control Battery Switch Open* commands (`<Insert>` and `<Ctrl+Insert>` by default). The status of the battery switch is indicated by the *Battery switch* value in the HUD view.

Three behaviours are available:

- By default, the battery switch is always closed (equivalent to MSTS).
- The battery switch can also be controlled directly by the driver with a switch. To get this behaviour, put the parameter `ORTSBattery( Mode ( Switch ) )` in the Engine section of the ENG file.
- The battery switch can also be controlled directly by the driver with two push buttons. To get this behaviour, put the parameter `ORTSBattery( Mode ( PushButtons ) )` in the Engine section of the ENG file.

In real life, the battery switch may not close instantly, so you can add a delay with the optional parameter `ORTSBattery( Delay ( ) )` (by default in seconds).

Example:

```
Engine (
  ORTSBattery (
    Mode ( PushButtons )
    Delay ( 2s )
  )
)
```

The state of the battery switch can be used in the *power supply scripts* and the *cabview controls*.

8.15.2 Master key

The master key controls the power supply of the cab. If the cab power supply is disabled, all of the systems of the cab are disabled (for example, the speed indicator switches off, the throttle controller is disabled, etc.).

The master key of the current cab can be controlled by the *Control Master Key* command (`<Enter>` by default). The status of the master key is indicated by the *Master key* value in the HUD view.

Two behaviours are available:

- By default, the master key is always on (equivalent to MSTS).
- The master key can also be controlled by the driver. To get this behaviour, put the parameter `ORTSMasterKey ( Mode ( Manual ) )` in the Engine section of the ENG file.

In real life, when the master key is switched off, the cab systems may not switch off instantly, so you can add a delay with the optional parameter `ORTSMasterKey ( DelayOff ( ) )` (by default in seconds).

The master key can also control the headlights on multiple units. For example, when the master key is switched on, the red lights can automatically be replaced by the white lights. In order to activate this behaviour, put The parameter `ORTSMasterKey ( HeadlightControl ( 1 ) )` in the Engine section of the ENG file.

Example:

```
Engine (
  ORTSMasterKey (
    Mode ( Manual )
    DelayOff ( 10s )
    HeadlightControl ( 1 )
  )
)
```

The state of the master key can be used in the *power supply scripts* and the *cabview controls*.

8.15.3 Service retention

The service retention allows for the systems of the train to still be supplied with electricity even if a cab is not in service. It maintains the pantographs up and the circuit breaker closed even if the master key is switched off.

This feature can only be used with a *custom power supply script*.

The service retention can be controlled by the *Control Service Retention* and *Control Service Retention Cancellation* commands (`<Delete>` and `<Ctrl+Delete>` by default).

Cabview controls are also available for this functionality.

8.15.4 Electric train supply

The electric train supply supplies passenger cars and heated wagons with electricity in order to power the battery chargers, the heating, ventilation and air conditioning systems.

The electric train supply can be controlled by the *Control Electric Train Supply* command (`<Alt+B>` by default). The status of the electric train supply switch state is indicated by the *Electric train supply* value in the HUD view.

Three behaviours are available:

- By default, the electric train supply is automatic (it will switch on as soon as the auxiliary power supply is on).
- The locomotive can also be not fitted with electric train supply. To get this behaviour, put the parameter `ORTSElectricTrainSupply ( Mode ( Unfitted ) )` in the Engine section of the ENG file.
- The electric train supply can also be controlled by the driver. To get this behaviour, put the parameter `ORTSElectricTrainSupply ( Mode ( Switch ) )` in the Engine section of the ENG file.

Example:

```
Engine (
  ORTSElectricTrainSupply (
    Mode ( Switch )
  )
)
```

If the locomotive is a diesel locomotive, the power consumed by the cars on the electric train supply is no longer available for traction.

The state of the electric train supply can be used in the *locomotive power supply scripts*, in the *passenger car power supply scripts* and the *cabview controls*.

8.15.5 Train Control System

The Train Control System is a system that ensures the safety of the train.

In MSTS, 4 TCS monitors were defined: the vigilance monitor, the overspeed monitor, the emergency stop monitor and the AWS monitor. Open Rails does not support the AWS monitor.

In order to define the behavior of the monitors, you must add a group of parameters for each monitor in the Engine section of the .eng file. These groups are called `VigilanceMonitor()`, `OverspeedMonitor()`, `EmergencyStopMonitor()` and `AWSMonitor()`.

In each group, you can define several parameters, which are described in the tables below.

Parameter	Description	Recom'd Input Units	Typical Examples
General Parameters			
MonitoringDeviceMonitorTimeLimit(x)	Period of time elapsed before the alarm or the penalty is triggered	Time	`(5s)`
MonitoringDeviceAlarmTimeLimit(x)	Period for which the alarm sounds prior to the penalty being applied	Time	`(5s)`
MonitoringDevicePenaltyTimeLimit(x)	Period in seconds before the penalty can be reset once triggered	Time	`(5s)`
MonitoringDeviceCriticalLevel(x)	Speed at which monitor triggers	Speed	`(200kph)`
MonitoringDeviceResetLevel(x)	Speed at which monitor resets	Speed	`(5kph)`
MonitoringDeviceAppliesFullBrake(x)	Sets whether full braking will be applied	Boolean – 0 or 1	`(0)`
MonitoringDeviceAppliesEmergencyBrake(x)	Sets whether emergency braking will be applied	Boolean – 0 or 1	`(1)`
MonitoringDeviceAppliesCutsPower(x)	Sets whether the power will be cut to the locomotive	Boolean – 0 or 1	`(1)`
MonitoringDeviceAppliesShutsDownEngine(x)	Sets whether the engine will be shut down	Boolean – 0 or 1	`(0)`
MonitoringDeviceResetOnDirectionNeutral(x)	Sets whether the monitor resets when the reverser is in the neutral position	Boolean – 0 or 1	`(0)`
MonitoringDeviceResetOnResetButton(x)	Sets whether the monitor resets when the reset button is pushed	Boolean – 0 or 1	`(0)`
MonitoringDeviceResetOnZeroSpeed(x)	Set whether the monitor resets when the speed is null	Boolean – 0 or 1	`(1)`
Specific parameters of the Overspeed Monitor			
MonitoringDeviceAlarmTimeBeforeOverSpeed(x)	Period for which the alarm sounds prior to the penalty being applied	Time	`(2s)`
MonitoringDeviceTriggerOnOverspeed(x)	Maximum allowed speed	Speed	`(200kph)`
MonitoringDeviceTriggerOnTrackOverspeed(x)	Activates the overspeed margin control	Boolean – 0 or 1	`(1)`
MonitoringDeviceTriggerOnTrackOverspeedMargin(x)	Allowed overspeed	Speed	`(5kph)`

Two other parameters in the Engine section of the ENG file are used by the TCS:

- `DoesBrakeCutPower( x )` sets whether applying brake on the locomotive cuts the traction (1 for enabled, 0 for disabled)
- `BrakeCutsPowerAtBrakeCylinderPressure( x )` sets the minimum pressure in the brake cylinder that cuts the traction (by default 4 PSI)

8.15.6 Train Derailment

Open Rails calculates when it is likely that a train derailment has occurred. The derailment modeled is based upon the wheel climbing the rail when the train is in a curve. Light (empty wagons) can sometimes derail due to 'string lining' where the train forces attempt to pull the train in a straight line, rather then following the curve.

OR calculates the Nadal Criteria for each wagon, and then calculates the actual L/V ratio based upon the wagon weight and the relevant "in train" forces. Open Rails uses some calculated default parameters for the various parameters required to determine the actual L/V ratio, however more accurate results will be obtained if actual parameters are entered into the ENG or WAG file. The derailment calculations use information relating to the wagon dimensions, weight and wheel profile information.

Wheel profile details can be entered with the following two parameters:

- `ORTSMaximumWheelFlangeAngle` - Wheel flange angle is defined as the maximum angle of the wheel flange relative to the horizontal axis.

UoM - Angle (deg, radians) - default is rad. Typically this value maybe between approx 60 and 75 degrees.

- `ORTSWheelFlangeLength` - Wheel flange length is defined as the length of flange starting from the beginning of the maximum flange angle

to the point where flange angle reduces to 26.6 degrees. UoM - Distance (m, in, ft, etc) - default is m

8.16 EOT - End of train device

8.16.1 General

See *https://en.wikipedia.org/wiki/End-of-train_device* for basic info about EOTs.

EOTs in Open Rails may be of three different levels (types):

```
- No communication: "dumb" EOTs, like flags or flashing lamps
- One way: the EOT is capable to transmit to the lead locomotive the brake pipe pressure␣
↪at the end of the train
- Two way: the EOT is also capable to receive a command to vent the air brake pipe.
```

8.16.2 How to define an EOT

EOTs must be defined within subfolders of the `Trains\ORTS_EOT` folder. These subfolders contain the same file set present in a subfolder of the `Trains\Trainset` folder; the file defining an EOT has the same format as a .wag file, but it must have an .eot extension. To define the level of the EOT the following block must be added at the end of the .eot file (after the closing parenthesis of the Wagon() block):

```
ORTSEOT (
      Level ( "level" )
)
```

level may assume following values: `NoComm`, `OneWay` and `TwoWay`.

Usually EOTs were provided as a very short wagon for simulation with MSTS or OR. To upgrade it to a functioning EOT for OR following simple steps are needed:

```
- Create the ``Trains\ORTS_EOT`` folder
- copy the EOT subfolder present in the ``Trains\Trainset`` folder into the ``Trains\ORTS_
↪EOT`` folder
- change the extension of the .wag file(s) to .eot
- add at the end of the .eot file the ORTSEOT block.
```

For Open Rails the EOT is a special type (a subclass) of wagon, with specific features. As such it appears at the end of the train in the Train Operations window.

8.16.3 How to attach and detach an EOT at the end of a train

An EOT may be attached at the end of a train (be it player train or AI train) already at game start, by inserting at the end of the .con file a block like the following one:

```
  ORTSEot (
        EOTData ( EOT_OR TrainSimulations_EOT )
        UiD ( 203 )
)
```

where `ORTSEot` and `EOTData` are fixed keywords, EOT_OR is the name of the .eot file and TrainSimulations_EOT is the folder where EOT_OR.eot resides. TrainSimulations_EOT is a subfolder of `Trains\ORTS_EOT`.

If an EOT is present at the end of the train since game start, it will be fully operating from the beginning (in one-way state if it is a one-way EOT, and in two-way state if it is a two-way EOT).

An EOT may also be attached at the end of the actual player train using the EOT list window



which can be recalled by pressing `<Ctrl+F9>`. Such window lists all .eot files present in the subfolders of `Trains\ORTS_EOT`. If the train has an EOT at the end of it, the related row in the EOT list window will be red. If the train has no EOT at the end of it (no red row), it can be attached at the end of the train with following sequence:

- within the EOT list window click on the row showing the desired EOT; the row will become red and the EOT will physically appear at the end of the train
- if it is an One Way or Two Way EOT type, connect its brake hose with the Car Operations Window (see *here*)
- using the Car Operations Window on the wagon preceding the EOT, open the rear angle cock.

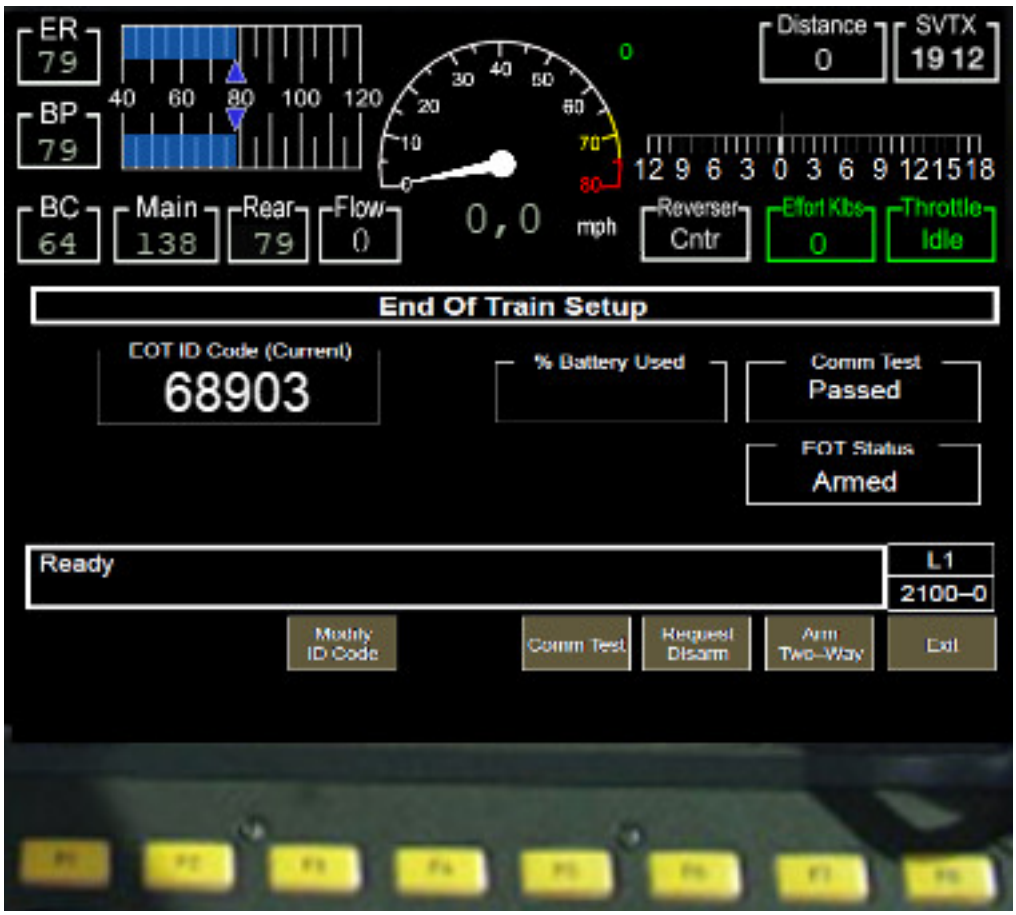
When an EOT is attached to the actual player train, a line indicating its presence will appear in the Train Driving Window (F5). The EOT will be in Disarmed state (that is fully disabled).

To detach an EOT from the end of the actual player train, recall the EOT list window and click on the red row. The EOT will disappear. Always remember to detach the EOT when this would occur in reality (e.g. when rear coupling other wagons, when decoupling the rear of the train and in general before shunting).

8.16.4 How to arm (enable) or disarm a one-way or two-way EOT

The arm and disarm procedure must be performed from the cabview, as the controls are available only through mouse. So it requires an equipped cabview. See *here* for a list of the available controls.

The procedure for a two-way EOT is explained basing on the picture below, which shows a sample case based on a cab of Borislav Miletic.



The possible states for a two-way EOT are following ones::

- Disarmed,
- CommTestOn,
- Armed,
- LocalTestOn,
- ArmNow,
- ArmedTwoWay

The EOT state is shown in the Train Driving window using above terminology, while the latter is a bit different in the EOT setup display in the sample cab following a real case (but it may be modified).

When the EOT is attached to the train with the EOT List window, the EOT is in `Disarmed` state. An EOT ID appears as a 5-digit random number. In the cab the Comm Test field shows `Failed`.

By clicking on the key below the `Comm Test` soft button, the state in the Train Driving window passes to the `CommTestOn` state. When the `CommTestOn` is terminated, the state in the Train Driving window passes to the `Armed` state for the one-way EOTs, and to the `LocalTestOn` state for the two-way EOTs. In the EOT setup display the Comm Test field shows `Passed` and the EOT Status Field shows `One Way`. Now the `Rear` field shows the brake pipe pressure at the end of the train.

The LocalTestOn in two-way EOTs is about 25 seconds long. After such time interval the state shown in the Train Driving window passes to the `ArmNow` state, and the EOT status in the display shows `Arm Now`.

At this point the train driver must click the key below the `Arm Two-Way` soft button. The EOT Passes to the `ArmedTwoWay` state, that is shown as `Armed` in the EOT status within the EOT Status display.

By clicking the key below the `Request Disarm` soft button, the EOT returns to the `Disarmed` state.

8.16.5 Emergency brake through EOT

Two-way EOTs, when in the `ArmedTwoWay` state, can be requested to vent the brake pipe and therefore to cause an emergency brake. This occurs automatically when an Emergency brake is triggered, and also manually when the `ORTS_EOT_EMERGENCY_BRAKE` control is activated. The manual activation may also occur by pressing `<Ctrl+Backspace>` .

CHAPTER 9

Further Open Rails Rolling Stock Features

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

9.1 Train Engine Lights

OR supports the whole set of lights accepted by MSTS.

9.2 Tilting trains

OR supports tilting trains. A train tilts when its .con file name contains the *tilted* string: e.g. `ETR460_tilted.con`.



9.3 Freight animations and pickups

9.3.1 OR implementation of MSTS freight animations and pickups

OR supports the freight animations as MSTS does (refueling of water, coal and diesel); when refueling from a water column the animation of the column arm is supported; coal level in the tender of the player loco decreases with consumption and increases when refueling.

The following pickup parameters are taken into consideration by OR for the MSTS animations:

- Pickup type
- Speed range
- Anim length

The pickup animation frame rate is computed as the ratio between the number of frames defined in the .s file, divided by the Anim length.

As in MSTS, Freight Animations are treated differently for tenders than for other vehicles.

Tenders:

- First numeric parameter: shape vertical position when full, relative to its origin, in meters
- Second numeric parameter: shape vertical position when empty, relative to its origin, in meters.
- Third numeric parameter: set to any positive value, or omitted, causes the shape to drop - see below.
  - As long as the second parameter is lower than the first and the third parameter is either omitted or has a non-zero value, the shape will drop, based on fuel consumption.
  - If the second parameter is not lower than the first, no movement will take place irrespective of the 3rd parameter.

Other Vehicles:

- The numeric parameters are not used.

9.3.2 OR specific freight animations and pickups

General

In addition to the support of the MSTS freight animations, Open Rails provides a large extension for freight animations (called *OR freightanims* below) and pickups.

Following are the native features Open Rails offers:

- two types of OR freightanims: continuous and static
- continuous OR freightanims are related to commodity loads, like coal, or stones: the load level in the trainset varies accordingly to the amount of load
- static OR freightanims are in fact additional shapes that can be attached to the main trainset shape. Such shapes may also include an animation (independent from train behaviour);
- both types of OR freightanims can be present in the same trainset, and can coexist with original MSTS freight animations
- both types of OR freightanims can be related to locomotives or wagons
- more than one static OR freightanim can be present in a single trainset
- a wagon can be loaded with different commodities in different moments
- commodities can be loaded (in pickup stations) and unloaded (in unloading stations).

- wagons supporting continuous OR freightanims may be provided with a physical animation that is triggered when unloading the wagon (like opening its bottom or fully rotating)
- OR freightanims are defined with an `ORTSFreightAnims ()` block within the .wag or within the wagon section of an .eng file. It is suggested that this block be defined within an include file as described *here*.

Continuous OR Freightanims

A description of this feature is best achieved by showing an example of an include file, (in this case named `AECX1636.wag` and located in an Openrails subfolder within the wagon's folder). Note that the first line of the file must be blank.:

```
include ( ../AECX1636.wag )

Wagon (
    ORTSFreightAnims
    (
        MSTSFreightAnimEnabled (0)
        WagonEmptyWeight(22t)
        IsGondola(1)
        UnloadingStartDelay (7)
        FreightAnimContinuous
        (
            IntakePoint ( 0.0 6.0 FreightCoal )
            Shape(Coal.s)
            MaxHeight(0.3)
            MinHeight(-2.0)
            FreightWeightWhenFull(99t)
            FullAtStart(0)
        )
        FreightAnimContinuous
        (
            IntakePoint ( 0.0 6.0 FuelCoal )
            Shape(Coal.s)
            MaxHeight(0.3)
            MinHeight(-2.0)
            FreightWeightWhenFull(99t)
            FullAtStart(0)
        )
    )
)
```

The `ORTSFreightAnims` block is composed by a set of general parameters followed by the description of the OR freightanims. Here below the general parameters are described:

- `MSTSFreightAnimEnabled` specifies if eventual MSTS freight animations within the trainset are enabled (1) or not (0). This is useful if one wants to use a wagon where the load is already shown with a (static) MSTS freight animation. In such a case the MSTS freight animation must be disabled, to use the OR freightanim, that allows to modify the vertical position of the freight shape.
- `WagonEmptyWeight` defines the mass of the wagon when empty. If the parameter is missing, the weight of the load is not considered and the weight of the wagon is always the value present in the root .eng file.
- `IsGondola` specifies (in case it is set to 1) if the load has to be rotated during unloading, as happens in a gondola wagon. If absent the parameter is set to 0.
- `UnloadingStartDelay` specifies, if present, after how many seconds after pressing of the T key the unloading starts. This is due to the fact that some seconds may be needed before the wagon is set

in a unloading layout. For example, a gondola must rotate more than a certain number of degrees before the load begins to fall down.

There may be more than one `FreightAnimContinuous` subblock, one for each possible load type. The parameters of the subblock are described below:

- `IntakePoint` has the same format and the same meaning of the IntakePoint line within the standard MSTS freight animations. Following types of loads are accepted: FreightGrain, FreightCoal, FreightGravel, FreightSand, FuelWater, FuelCoal, FuelDiesel, FuelWood, FuelSand, FreightGeneral, FreightLivestock, FreightFuel, FreightMilk, SpecialMail. All these types of loads can be defined. Some of the pickup types (to right of FuelDiesel) need to be coded in W text files.
- `Shape` defines the path of the shape to be displayed for the load
- `MaxHeight` defines the height of the shape over its 0 position at full load
- `MinHeight` defines the height of the shape over its 0 position at zero load
- `FreightWeightWhenFull` defines the mass of the freight when the wagon is full; the mass of the wagon is computed by adding the mass of the empty wagon to the actual mass of the freight
- `FullAtStart` defines wether the wagon is fully loaded (1) or is empty at game start; if there are more continuous OR freightanims that have `FullAtStart` set to 1, only the first one is considered.

As already outlined, the wagon may have a physical animation linked with the unload operation.

In a gondola this could be used to rotate the whole wagon, while in a hopper it could be used to open the bottom of the wagon.

The base matrix within the wagon shape that has to be animated must have a name that starts with `UNLOADINGPARTS`. There may be more than one, like `UNLOADINGPARTS1`, `UNLOADINGPARTS2` and so on. Its frame rate is fixed, and is 1 frame per second as for the other types of OR trainset animations.

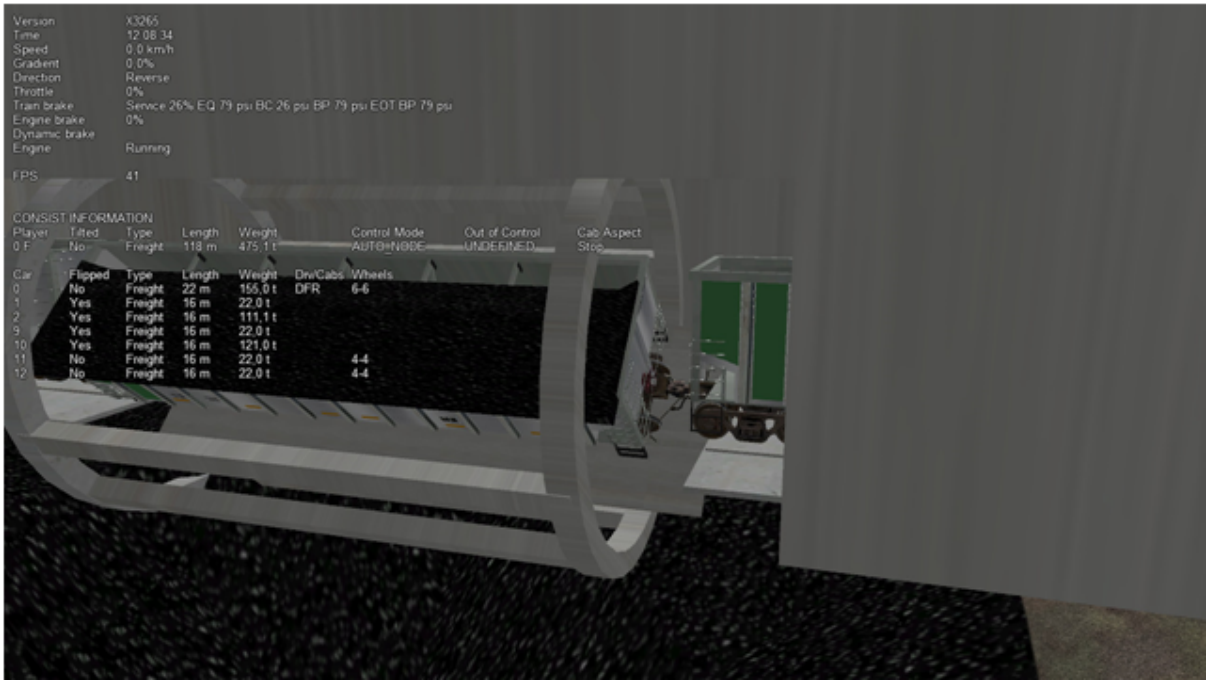
To define a pickup point as an unload point, its shape must be inserted in the .ref file of the route as a pickup object . Here is an example of the .ref block:

```
Pickup (
    FileName ( rotary_dump.s )
    Shadow ( DYNAMIC )
    Class ( "Track Objects" )
    PickupType ( _FUEL_COAL_ )
    Description ( "Rotary dumper" )
)
```

When laying it down in the route with the MSTS Route Editor, its fill rate must be set to a negative value.

Such a pickup (which in reality is an unloader) may be animated too. The base matrix within the wagon shape that has to be animated must have a name that starts with `ANIMATED_PARTS`. There may be more than one, like `ANIMATED_PARTS1`, `ANIMATED_PARTS2` and so on. As for the MSTS standard pickups, the pickup animation frame rate is computed as the ratio between the number of frames defined in the .s file, divided by the Anim length.

By combining a physical animation of the wagon with an unloader animation effects like that of a wagon within a rotary dumper may be achieved, as seen in the picture below.



Loading and unloading a trainset is triggered by pressing the <T> key when the trainset is at the pickup/unloader location.

Static OR Freightanims

Only the two general parameters shown below are used for static OR freightanims:

```
MSTSFreightAnimEnabled (0)
WagonEmptyWeight(22t)
```

The subblock (to be inserted within the `ORTSFreightAnims` block) has the following format:

```
FreightAnimStatic
(
    SubType(Default)
    Shape(xxshape.s)
    Offset(XOffset, YOffset, ZOffset)
    FreightWeight(weight)
    Flip()
    Visibility ( "Outside,Cab2D,Cab3D" )
)
```

Where:

- `SubType` is not currently used
- `Shape` is the path of the shape file.
- `XOffset`, `YOffset` and `ZOffset` are the offsets of the shape with respect to its zero position, and are useful to place the shape precisely.
- `FreightWeight` is the weight of the specific load. This weight is added to the `WagonEmptyWeight` value (if present) to provide the total weight of the wagon. If more static OR freightanims are present, each of their weights is added to define the total weight of the wagon.
- `Flip()`, if present, flips the shape around its pivot point.
- `Visibility`, if present, changes the default visibility of the static freightanim. Default is visible only from outside cameras and from any inside camera of locomotives different from the one hosting

the static freightanim. If substring `Outside` is present, the static freightanim is visible from outside cameras and from any inside camera of locomotives different from the one hosting the static freightanim; if `Cab2D` is present, the static freightanim is visible from the 2D cabview camera of loco hosting the freightanim; if `Cab3D` is present, the static freightanim is visible from the 3D cabview camera of loco hosting the freightanim. 1, 2 or 3 of such substrings may be inserted in the `Visibility` line allowing for any combination of visibility.

Because more static OR freightanims may be defined for a wagon, in the case of a container wagon that is able to carry more than one container, even as a double stack, it is possible to use a static OR freightanim for each container, defining its position within the wagon.

9.3.3 Physics Variation with Loads

Variable Loads (Continuous Freight Animation)

Open Rails supports the variation of key physics parameters in the wagon as the load varies within the wagon. The parameters which can be changed are:

- Mass
- Brake and handbrake force
- Friction (general and wind)
- Centre of Gravity (impacts on curve performance)
- Drive wheel weight (impacts upon locomotive adhesve weight)

Locomotives and tenders that are also configured will have their loads, and the above physics parameters adjusted as coal and water is used. The adhesive weight (Drive wheel weight) will also be adjusted as the load changes.

To support the correct operation of this feature a known physics starting and finishing point is required, ie the state of these parameters under empty conditions, and the state of these parameters when the wagon or locomotive is full.

To configure the stock correctly the following empty and full parameters need to be included in the ORTSFreightAnims file. Empty values are included in the first block, and full values are included in the second code block. A sample code block is shown below.:

```
ORTSFreightAnims
(
  MSTSFreightAnimEnabled (0)
  WagonEmptyWeight(10.0t-uk)
  EmptyMaxBrakeForce ( 29.892kN )
  EmptyMaxHandbrakeForce ( 9.964kN )
  EmptyORTSDavis_A ( 580.71 )
  EmptyORTSDavis_B ( 5.0148 )
  EmptyORTSDavis_C ( 0.694782 )
  EmptyORTSWagonFrontalArea ( 10.0m )
  EmptyORTSDavisDragConstant ( 0.0003 )
  EmptyCentreOfGravity_Y ( 1.41 )
  IsGondola(0)
  UnloadingStartDelay (5)

FreightAnimContinuous
 (
  IntakePoint ( 0.0 6.0 FreightCoal )
  Shape(H_Coal.s)
  MaxHeight(0.1)
  MinHeight(-0.85)
```

(continues on next page)

(continued from previous page)

```
    FreightWeightWhenFull(26.0t-uk)
    FullAtStart( 0 )
    FullMaxBrakeForce ( 89.676kN )
    FullMaxHandbrakeForce ( 9.964kN )
    FullORTSDavis_A ( 748.61 )
    FullORTSDavis_B ( 18.0157 )
    FullORTSDavis_C ( 0.838530 )
    FullORTSWagonFrontalArea ( 15.0m )
    FullORTSDavisDragConstant ( 0.005 )
    FullCentreOfGravity_Y ( 1.8 )
   )
)
```

Note for enclosed wagons, such as covered vans, the freight animation shape may not be required, and therefore the parameters Shape, MaxHeight, and MinHeight can be left out of the file.

The `IntakePoint` statement is necessary to ensure satisfactory operation of the feature.

Open Rails supports the following freight or fuel load types:

- FreightGrain = 1,
- FreightCoal = 2,
- FreightGravel = 3,
- FreightSand = 4,
- FuelWater = 5,
- FuelCoal = 6,
- FuelDiesel = 7,
- FuelWood = 8,
- FuelSand = 9,
- FreightGeneral = 10,
- FreightLivestock = 11,
- FreightFuel = 12,
- FreightMilk = 13,
- SpecialMail = 14

The key word, e.g. `FreightMilk`, is used to define the freight type in the `IntakePoint` statement, whilst the number is used to define the pickup point in the route (Replaces the first number in the `PickupType ( 1 0 )` statement).

For load variation in a locomotive, a similar configuration is used in regard to the full and empty parameters, but as the `IntakePoint` statement is normally included elsewhere in the ENG file or tender (or auxiliary tender) WAG file these statements can be left out of the freight animation section.

For example, the following code block would apply to a steam locomotive (note the absence of the `IntakePoint` statement):

```
ORTSFreightAnims
(
    WagonEmptyWeight(76.35t-uk)
    EmptyMaxBrakeForce ( 29.892kN )
    EmptyMaxHandbrakeForce ( 9.964kN )
    EmptyORTSDavis_A ( 580.71 )
    EmptyORTSDavis_B ( 5.0148 )
```

(continues on next page)

(continued from previous page)

```
    EmptyORTSDavis_C ( 0.694782 )
    EmptyCentreOfGravity_Y ( 1.41 )

  FreightAnimContinuous
   (
    FreightWeightWhenFull(10.34t-uk)
    FullMaxBrakeForce ( 89.676kN )
    FullMaxHandbrakeForce ( 9.964kN )
    FullORTSDavis_A ( 748.61 )
    FullORTSDavis_B ( 18.0157 )
    FullORTSDavis_C ( 0.838530 )

    FullCentreOfGravity_Y ( 1.8 )
   )
)
```

Notes:

- Intake points should be defined within the root WAG file
- Intake points, freight animations should not be defined within the INCLUDE file
- Empty weight of tender will be the full mass minus coal and water weight
- `FreightWeightWhenFull` will be the sum of the coal and water weight.
- Full physics values will be those values for the combined weight of the tender, water and coal.
- The parameters for wind resistance (ORTSWagonFrontalArea and ORTSDavisDragConstant) can be left out if the area and drag does not change between the full and empty states.

Static wagons (Static Freight Animations)

Static wagons can be defined with a full and empty state, however only one freight animation should have full values assigned to it,as OR cannot then calculate the known full state.

A typical configuration code block will be as follows:

```
ORTSFreightAnims
(
  MSTSFreightAnimEnabled (0)
  WagonEmptyWeight(6.5t-uk)

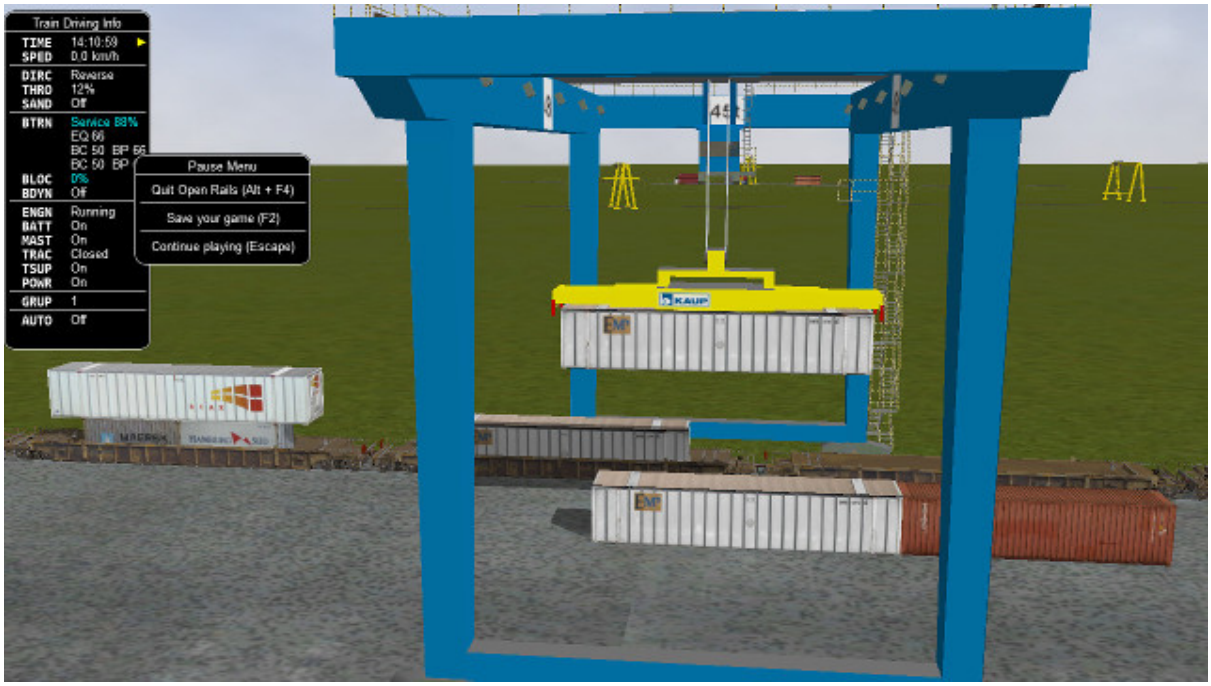
FreightAnimStatic
(
  SubType(Default)
  Shape( 15ft_3p_HumpSheet2.s )
  Offset( 0, 0, 0)
  FreightWeight( 9.0t-uk )
  FullMaxBrakeForce ( 19.43kN )
  FullMaxHandbrakeForce ( 6.477kN )
  FullORTSDavis_A ( 358.37 )
  FullORTSDavis_B ( 7.7739 )
  FullORTSDavis_C ( 0.718740 )
  FullORTSWagonFrontalArea ( 15.0m )
  FullORTSDavisDragConstant ( 0.005 )
  FullCentreOfGravity_Y ( 1.8 )
 )
)
```

The empty values for the wagon will be read from the normal base WAG file paramaters.

9.4 Container management

9.4.1 General

With this feature containers are not static objects laying on earth or on wagons, but may be loaded from a container station onto a wagon, or unloaded from a wagon and laid on a container station. The load/unload operations are performed through a crane, which is the heart of the container station.



The other component of the container station is the set of stack locations, that is the locations where containers may lay. Containers of same length can be stacked one above the other.

Wagons may be empty at game start, or partially or totally pre-loaded with containers, by inserting the related data either in the consist (`.con`) file or in the `.wag` files.

Also container stations may be empty at game start, or partially or totally populated with containers, inserting the related data in the activity (`.act`) file.

The loading and unloading operations are started by the player, by pressing the key `<T>` for loading, and the key `<Shift-T>` . The operation is performed on the first wagon (starting from the locomotive) which is within the container crane displacement range and which fulfills the required conditions (e.g. loading space available for loading, container present for unloading).

Double stack wagons are managed.

From a point of view of internal code structure, Open Rails handles container stations as special pickups.

9.4.2 How to define container data

Container shape files (`.s`) must be located in subfolders (or sub-subfolders) of the `Trainset` folder. Containers that can be managed must be provided with a Json `.load-or` file. The `.load-or` files must be located in a subfolder of the `Trainset` folder. It is warmly advised to keep all `.load-or` file in a single folder: `Common.ContainerData` is suggested. It is also advised to name the `.load-or` files in a consistent way: `40HCtriton.load-or` is suggested, where `40HC` is the container type and `triton` the brand painted on the container.

Format of the .load-or file

Here below a sample of a `.load-or` file:

```
{
      "Container":
      {
              "Name" : "triton",
              "Shape" : "COMMON_Container_3d\\Cont_40ftHC\\container-40ftHC_Triton.s",
              "ContainerType" : "C40ftHC",
              "IntrinsicShapeOffset": [0,1.175,0],
        }
}
```

- "Container" is a fixed keyword.
- "Name" has as value a string used by Open Rails when the container must be indentified in a message to the player.
- "Shape" has as value the path of the container shape, having `Trainset` as base.
- "ContainerType" identifies the container type, which may be one of the following ones:

  ```
  * C20ft
  * C40ft
  * C40ftHC
  * C45ft
  * C45ftHC
  * C48ft
  * C53ft
  ```

  C48ft and C53ft have a HC height (2.90m)
- "IntrinsicShapeOffset" has as value the offset in meters of the center of the bottom rectangle of the container with respect to the container shape file coordinates. Unfortunately often such offset is not [0,0,0], which would be advisable for newly produced containers. A simple way to state such offset is to use the `Show Bounding Info` of `Shape Viewer`.

9.4.3 Pre-setting a .wag file to accommodate containers

As a minimum following block must be present in the .wag file for a double stacker:

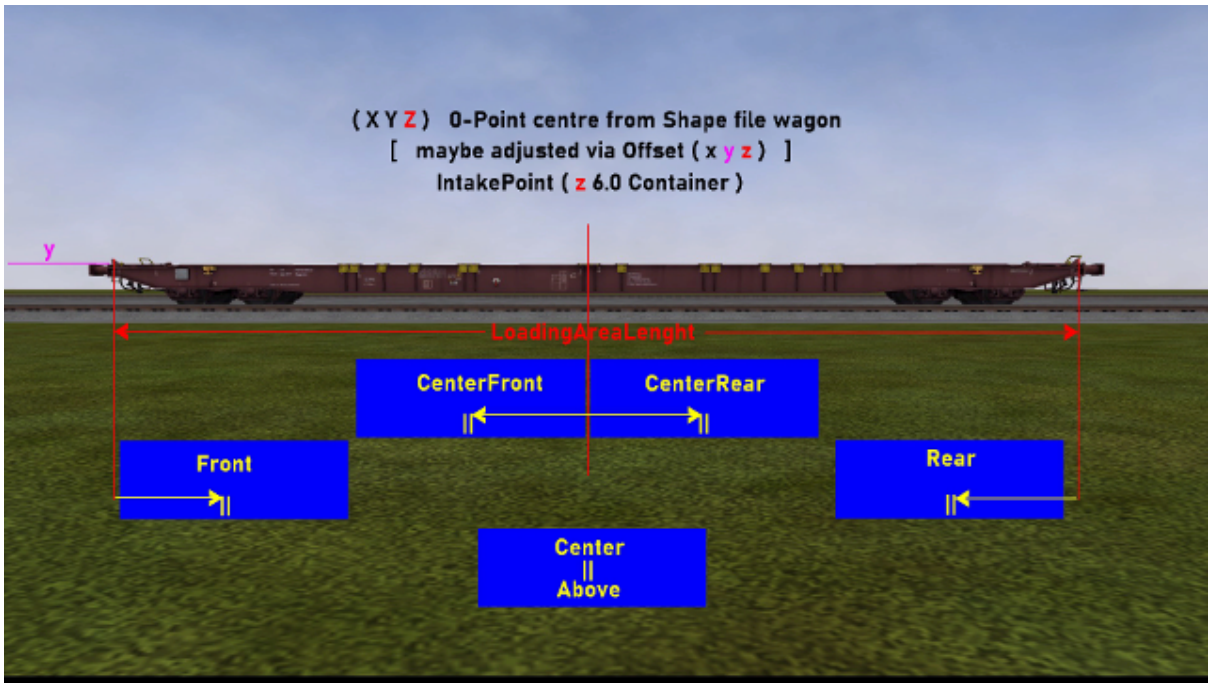
```
ORTSFreightAnims (
              LoadingAreaLength ( 12.20 )
              AboveLoadingAreaLength ( 12.20 )
              DoubleStacker ()
              Offset( 0 0.34 0 )
              IntakePoint ( 0 6.0 Container)
              )
```

- LoadingAreaLength is the length in meters of the loading area available for containers
- AboveLoadingAreaLength is the length in meters of the above loading area available for containers (parameter not needed if not double stacker)
- DoubleStacker must be present if the wagon allows double stacking
- Offset is the offset of the center of the rectangle of the loading area with respect to the shape file of the wagon.
- The first and the third IntakePoint parameters have the same meanings than the ones used for generic pickups. The first parameter must be equal to the Z value of the offset. `Container` is mandatory.

This `ORTSFreightAnims` block can include also static freight animations as described in the related paragraph.

9.4.4 Allocation of the containers on the wagons

A container may have following positions within the loading area of the wagon: Rear, CenterRear, Center, CenterFront, Front and Above. Following picture shows where the first five positions are located on the wagon, while Above is the above position in dual-stack configurations. The Above position is always centered.



Some loading configurations are shown in following picture:



From left to right the loading configurations are present (locomotive on the left):

- CenterFront, CenterRear, Above
- Center

- Front, Rear
- Front, Center, Rear
- Front, Rear
- Front, CenterFront, CenterRear, Rear.

The real rules to allocate double-stacked containers must be respected:

- no 20ft stacked above
- only one container above
- at least 40ft of containers below.

9.4.5 How to allocate containers on wagons at start of game

The containers may be allocated either by editing the `.con` file, or by editing the `.wag` file, or in a mixed mode (some wagons in one mode, some others in another mode).

Allocation through `.con` file

This allocation mode is the recommended one, as it is more flexible and provides easier visibility.

A wagon entry complete with the data about the containers loaded at startup is shown here:

```
Wagon (
          WagonData ( DTTX_620040_A ATW.DTTX_620040 )
          LoadData ( 20cmacgm common.containerdata CenterFront)
          LoadData ( 20hamburgsud common.containerdata CenterRear)
          LoadData ( 40msc common.containerdata Above)
          UiD ( 11 )
)
```

As can be seen, for each container loaded at startup a `LoadData` entry must be present. The meaning of the parameters is as follows:

- The first parameter is the name of the `.load-or` file
- The second parameter is the path (having `Trainset` as base path) where the `.load-or` file resides
- The third parameter indicates where the container is allocated on the wagon.

The entry for the container allocated `Above` must be the last one.

CenterFront and CenterRear entries must be entered after Front or Rear entries.

The advantage of this type of allocation is that, for a single `.wag` file (in the example `DTTX_620040_A.wag`) more possible container configurations are possible, sparing the time of creating many `.wag` files that differ only on the containers loaded.

Here below a picture with a sample entry in the `.con` file:



Allocation through `.wag` file

Content creators might prefer to provide packs of pre-loaded wagons. Therefore it is also possible to set in `.wag` file the containers to be loaded at startup.

A minimum `FreightAnimations` entry in a `.wag` file to have the same pre-loaded container set as in the previous paragraph is as follows:

```
ORTSFreightAnims (
            LoadingAreaLength ( 14.6 )
            AboveLoadingAreaLength ( 16.15 )
              DoubleStacker ()
              Offset( 0 0.34 0 )
              IntakePoint ( 0 6.0 Container)
              LoadData ( 20cmacgm common.containerdata CenterFront)
              LoadData ( 20hamburgsud common.containerdata CenterRear)
              LoadData ( 40msc common.containerdata Above)
            )
```

As can be seen, the syntax of the `LoadData` entries is the same as in the case of the `.con` file.

Obviously, using `.wag` files for this type of info, a different `.wag` file must be created for every desired pre-loaded set of containers.

A single `.con` file can include Wagon entries for both types of allocation definition.

9.4.6 Container Station

The Container Station is composed by a container crane and a container stack area.

To insert a Container Station in a route, its object must be present in the `.ref` file as a Pickup object. A `.ref` file entry sample is as follows:

```
Pickup (
  Class                   ( "Animated loader" )
  Filename                ( RMG_45.s )
  PickupType              ( _FUEL_COAL_ )
  Description             ( "Animated container crane" )
)
```

PickupType is set to `_FUEL_COAL`, but this will be overwritten by the data inserted in the extension `.w` file (see *here*) within the `Openrails` sufolder of the World folder.

Such extension `.w` file is formed by a general part, a container crane related part, and a stack locations related part, as per following example (parts separated by blank lines)

```
SIMISA@@@@@@@@@@JINX0w0t______

Tr_Worldfile (
               Pickup (
                       UiD ( 21 )
                       PickupType ( 15 1 )

                       ORTSPickingSurfaceYOffset ( 2.25 )
                       ORTSPickingSurfaceRelativeTopStartPosition ( 0 6.75 0 )
                       ORTSGrabberArmsParts ( 2 )
                       ORTSCraneSound ( "ContainerCrane.sms" )

                       ORTSMaxStackedContainers ( 2 )
                       ORTSStackLocationsLength ( 12.19 )
                       ORTSStackLocations ( 12
                               StackLocation (
                                       Position ( -10 0 26 )
                                       Length ( 16.15 )
                                       )
                               StackLocation (
                                       Position ( -10 0 26 )
                                       MaxStackedContainers ( 1 )
                                       Flipped ( 1 )
                                       )
                               StackLocation (
                                       Position ( -10 0 0 )
                                       MaxStackedContainers ( 2 )
                                       )
                               StackLocation (
                                       Position ( -10 0 0 )
                                       Flipped ( 1 )
                                       )
                               StackLocation (
                                       Position ( -10 0 -26 )
                                       )
                               StackLocation (
                                       Position ( -10 0 -26 )
                                       Flipped ( 1 )
                                       Length ( 16.15 )
                                       )
                               StackLocation (
                                       Position ( -7 0 26 )
                                       Length ( 16.15 )
                                       )
                               StackLocation (
                                       Position ( -7 0 26 )
                                       Flipped ( 1 )
                                       )
                               StackLocation (
                                       Position ( -7 0 0 )
                                       )
                               StackLocation (
```

(continues on next page)

(continued from previous page)

```
                                Position ( -7 0 0 )
                                Flipped ( 1 )
                                )
                        StackLocation (
                                Position ( -7 0 -26 )
                                )
                        StackLocation (
                                Position ( -7 0 -26 )
                                Flipped ( 1 )
                                Length ( 16.15 )
                                )
                        )
                )
)
```

- The UiD number must correspond to the uiD number that the pickup has in the main `.w` file.
- PickupType (15 1) identifies this pickup as being a container station.

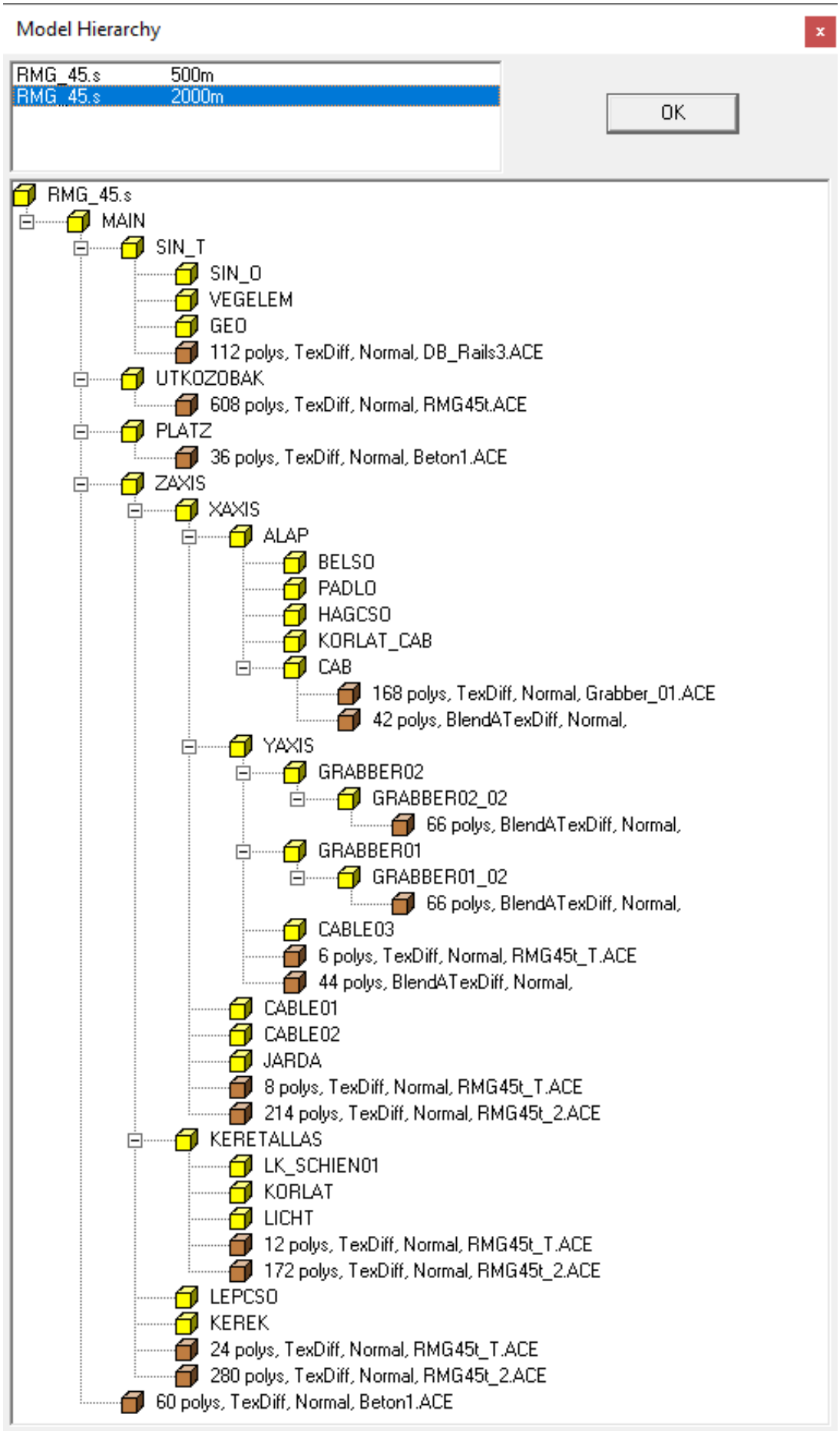
More than a Pickup() block can be present in such extension file, one for every container station present in the route.

The container crane and stack location related data are described at a convenient point below.

Container Station (including container crane) shape file developing rules

- The shape file must have its Z Axis aligned with the track where the wagons to be loaded or unloaded stay.
- The Z-zero of the shape file must be in the middle of the segment that the crane can cover in its motion (e.g. the crane Z-span could be -30 meters to 30 meters).
- The animation of the part of the crane moving along the Z axis must be called `ZAXIS`.
- The animation of the part of the crane moving transversally along the X axis must be called `XAXIS`, and must be hierarchically dependent from `ZAXIS`.
- The animation of the part of the crane moving vertically along the Y axis must be called `YAXIS`, and must be hierarchically dependent from `XAXIS`.
- The grabbers are the extensible arms that pick the container. In the simplest case there are two sections, one extending towards positive Z for longer containers, and one extending towards negative Z. The first one must be called `GRABBER01` and the second one `GRABBER02`. Both must be hierarchically dependent from `YAXIS`. In the most complex case each of the two "arms" is composed by two parts, which move like a telescope. Such second couple of arms must be called `GRABBER01_O2` and `GRABBER02_02`. They must be hierarchically dependent from `GRABBER01` and `GRABBER02`. The spans of `GRABBER01` and `GRABBER02` must be symmetric, and the same applies for the other couple of spans. Moreover the spans of `GRABBER01` and `GRABBER01_02` must be equal (and symmetrically also the other couple).
- The names of the cable parts that have a partially autonomous motion along the Y axis (to simulate cable winding and unwinding) must start with `CABLE` and must be hierarchically dependent from `YAXIS`.

The following diagram, taken from Shape Viewer, sums up the above rules.


Model Hierarchy
RMG_45.s 500m
RMG_45.s 2000m
OK
RMG_45.s
MAIN
SIN_T
SIN_O
VEGELEM
GEO
112 polys, TexDiff, Normal, DB_Rails3.ACE
UTKOZOBAK
608 polys, TexDiff, Normal, RMG45t.ACE
PLATZ
36 polys, TexDiff, Normal, Beton1.ACE
ZAXIS
XAXIS
ALAP
BELSO
PADLO
HAGCSO
KORLAT_CAB
CAB
168 polys, TexDiff, Normal, Grabber_01.ACE
42 polys, BlendATexDiff, Normal,
YAXIS
GRABBER02
GRABBER02_02
66 polys, BlendATexDiff, Normal,
GRABBER01
GRABBER01_02
66 polys, BlendATexDiff, Normal,
CABLE03
6 polys, TexDiff, Normal, RMG45t_T.ACE
44 polys, BlendATexDiff, Normal,
CABLE01
CABLE02
JARDA
8 polys, TexDiff, Normal, RMG45t_T.ACE
214 polys, TexDiff, Normal, RMG45t_2.ACE
KERETALLAS
LK_SCHIEN01
KORLAT
LICHT
12 polys, TexDiff, Normal, RMG45t_T.ACE
172 polys, TexDiff, Normal, RMG45t_2.ACE
LEPCSO
KEREK
24 polys, TexDiff, Normal, RMG45t_T.ACE
280 polys, TexDiff, Normal, RMG45t_2.ACE
60 polys, TexDiff, Normal, Beton1.ACE

Following are the significant animation entries of a crane's shape file:

```
animations ( 1
               animation ( 2 30
                       anim_nodes ( 30
                               anim_node MAIN (
                                       controllers ( 0 )
                               )
               ...
                               anim_node ZAXIS (
                                       controllers ( 1
                                               linear_pos ( 3
                                                       linear_key ( 0 0 0 -139.5 )
                                                       linear_key ( 12 0 0 139.5 )
                                                       linear_key ( 24 0 0 -139.5 )
                                               )
                                       )
                               )
                               anim_node XAXIS (
                                       controllers ( 1
                                               linear_pos ( 3
                                                       linear_key ( 0 0 0 0 )
                                                       linear_key ( 3 26.4 0 0 )
                                                       linear_key ( 6 0 0 0 )
                                               )
                                       )
                               )
               ...
                               anim_node YAXIS (
                                       controllers ( 1
                                               linear_pos ( 3
                                                       linear_key ( 0 0 11.7 0 )
                                                       linear_key ( 2 0 0 0 )
                                                       linear_key ( 4 0 11.7 0 )
                                               )
                                       )
                               )
                               anim_node GRABBER02 (
                                       controllers ( 1
                                               linear_pos ( 3
                                                       linear_key ( 0 0 0 -2.515 )
                                                       linear_key ( 1 0 0 0 )
                                                       linear_key ( 2 0 0 -2.515 )
                                               )
                                       )
                               )
                               anim_node GRABBER02_02 (
                                       controllers ( 1
                                               linear_pos ( 3
                                                       linear_key ( 0 0 0 -2.513 )
                                                       linear_key ( 1 0 0 0 )
                                                       linear_key ( 2 0 0 -2.513 )
                                               )
                                       )
                               )
                               anim_node GRABBER01 (
                                       controllers ( 1
```

(continues on next page)

(continued from previous page)

```
                                        linear_pos ( 3
                                                linear_key ( 0 0 0 2.515 )
                                                linear_key ( 1 0 0 0 )
                                                linear_key ( 2 0 0 2.515 )
                                        )
                                )
                        )
                        anim_node GRABBER01_02 (
                                controllers ( 1
                                        linear_pos ( 3
                                                linear_key ( 0 0 0 2.513 )
                                                linear_key ( 1 0 0 0 )
                                                linear_key ( 2 0 0 2.513 )
                                        )
                                )
                        )
        ...
                        anim_node CABLE02 (
                                controllers ( 1
                                        linear_pos ( 3
                                                linear_key ( 0 0 22.32 0 )
                                                linear_key ( 1 0 15.72 0 )
                                                linear_key ( 2 0 22.32 0 )
                                        )
                                )
                        )
                        ...
                )
        )
)
```

It can be noted that the frame count is different for different animation nodes, e.g. the ZAXIS has 0, 12, 24. This permits to scale down the motion speed along that axis to a realistic value.

Parameters of extension `.w` file related to the crane and its animations

- ORTSPickingSurfaceYOffset (0.0) : the Y offset of the lower face of the grabbers (the one which gets in contact with the upper face of the container) when `YAXIS` is equal to 0
- ORTSPickingSurfaceRelativeTopStartPosition (0 11.7 0) : the values of `XAXIS`, `YAXIS` and ZAXIS at game start (should be centered on the Z axis, above the rails and at top height)
- ORTSGrabberArmsParts (4) : is 4 if there are all four `GRABBER01`, `GRABBER02`, `GRABBER01_02` and `GRABBER02_02` animations; is 2 if there is only `GRABBER01` and `GRABBER02`
- ORTSCraneSound ("ContainerCrane.sms") : name and path of the crane sound file; the path is based on the `SOUND` folder of the route; if the file is not found there, the path becomes based on the `SOUND` folder of `TRAIN SIMULATOR`. The specific discrete sound triggers available are listed *here* .

Stack Locations

Within the area that can be reached by the container crane (rails area apart) stack locations where the containers can be laid down can be defined in the extension `.w` file.

The stack locations are defined by following parameters:

- `Position`: the coordinates of the center point of one of the short sides of the stack location; if no `Flipped ( 1 )` line is present, the location area extends towards the increasing Z axis; if instead such line is present, the location area extends towards the decreasing Z axis. If two stack locations have the same position, and one is flipped and the other isn't, the containers will be laid back-to-back, optimizing space used.
- `Length`: the maximum length of the containers that can be laid down on that stack location
- `MaxStackedContainers`: The maximum number of containers that can be stacked one above the other on that stack location

The `Length` and `MaxStackedContainers` parameters are optional and, when present, override the default values present in the `ORTSStackLocationsLength` and `ORTSMaxStackedContainers`.

If `ORTSStackLocationsLength` is greater or equal to 12.20m, which is twice the length of a 20ft container, Open Rails applies a space optimization strategy: for each stack location (let's call it the mother stack location), another one (let's call it the child stack location) is created on a position with a Z value which is 6.095m greater than the mother stack location (if the latter is flipped the Z value is 6.095m smaller). This child stack location can be occupied by a 20ft container only, and only if the mother stack location is empty or occupied by a 20ft container too. The child stack location has an index which is equal to the mother stack location index plus the total number of mother stack locations. Once both the mother and the child stack locations are empty, the mother stack location is again available for any type of container of suitable length.

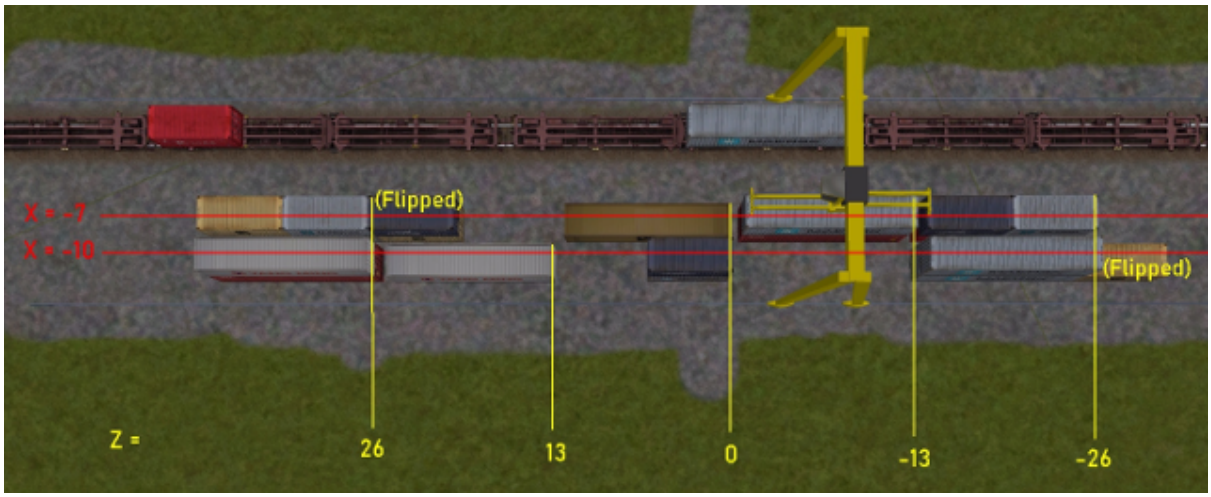
A further example of a stack locations allocation code and of its physical counterpart in the container station follows. It can be noted that stack location 0 has a 20ft container on it, and so has its child stack location 10. Same applies to stack location 3 and its child stack location 13.

```
ORTSMaxStackedContainers ( 2 )
ORTSStackLocationsLength ( 12.19 )
ORTSStackLocations ( 10

        StackLocation (
                Position ( -7 0.05 -26 )
                )
        StackLocation (
                Position ( -7 0.05 -13 )
                )
        StackLocation (
                Position ( -7 0.05 0 )
                MaxStackedContainers ( 1 )
                )
        StackLocation (
                Position ( -7 0.05 26 )
                )
        StackLocation (
                Position ( -7 0.05 26 )
                Flipped ( 1 )
                )

        StackLocation (
                Position ( -10 0.05 -26 )
                )
        StackLocation (
                Position ( -10 0.05 -26 )
                Flipped ( 1 )
                )
        StackLocation (
                Position ( -10 0.05 0 )
                )
        StackLocation (
                Position ( -10 0.05 13 )
                MaxStackedContainers ( 1 )
                )
        StackLocation (
                Position ( -10 0.05 26 )
                )


        )
```



Population of container stations at game start

Container stations may be populated at game start. This occurs by inserting a `.load-stations-loads-or` file in the `Openrails` subfolder of the "Activities" folder of the route, and inserting the following line at the bottom of the `Tr_Activity_Header` in `.act` files

```
ORTSLoadStationsPopulation ( BigContainerStationPopulation )
```

where `BigContainerStationPopulation` is the name of the `.load-stations-loads-or` file. At the moment population at game start is possible only in Activity mode.

The `.load-stations-loads-or` file is a Json file. An example is shown here below

```
      "ContainerStationsPopulation": [
              {
                      "LoadStationID" : { "wfile" : "w-005354+014849.w", "UiD" :  21, },
                      "LoadData" : [
                              { "File" : "40HCcai", "Folder" : "common.containerdata",
↪"StackLocation" : 0, },
                              { "File" : "40HCcai", "Folder" : "common.containerdata",
↪"StackLocation" : 0, },
                              { "File" : "20cmacgm", "Folder" : "common.containerdata",
↪"StackLocation" : 2, },
                              { "File" : "20kline", "Folder" : "common.containerdata",
↪"StackLocation" : 2, },
                              { "File" : "45HCtriton", "Folder" : "common.containerdata",
↪"StackLocation" : 5, },
                              { "File" : "45HCtriton", "Folder" : "common.containerdata",
↪"StackLocation" : 5, },
                              { "File" : "48emp", "Folder" : "common.containerdata",
↪"StackLocation" : 6, },
                              { "File" : "20maersk", "Folder" : "common.containerdata",
↪"StackLocation" : 14, },
                              { "File" : "20maersk3", "Folder" : "common.containerdata",
↪"StackLocation" : 14, },
                      ]
              },
              {
                      "LoadStationID" : { "wfile" : "w-005354+014849.w", "UiD" :  210, },
                      "LoadData" : [
                      ...
                      ]
              },
              ...
      ]
}
```

The file can define the population at startup of many container stations.

- The `LoadStationID` contains the info needed to identify the container station.
- The `LoadData` array contains the data to populate the container station.
- The value of `File` is the name of the `.load-or` file identifying the container.
- The value of `Folder` is the path where the `.load-or` can be found, starting from the `TRAINSET`.
- The value of `StackLocation` is the index of the Stack Location. If the index is equal or higher than the number of stack locations defined in the extension `.w` file, the index refers to a child stack location.
- If more than a container is defined for a stack location, they are stacked one above the other.

The container station population file must be written taking into account the constraints of the stack locations (container length must be smaller than stack location lenght, stacked containers can't exceed the allowed number, a stack location must contain containers of same length).

9.5 Multiple passenger viewpoints

Additional passenger viewpoints may be added within a carriage that is provided with passenger viewpoint.

Such additional passenger viewpoints are defined within an include file with the format shown in following example for the legacy oebarcar.wag (located in the 380 folder) MSTS wagon:

```
include ( ../oebarcar.wag )

Wagon (
      ORTSAlternatePassengerViewPoints (
              ORTSAlternatePassengerViewPoint (
                      PassengerCabinHeadPos ( -0.0 2.85801 -6.091 )
                      RotationLimit ( 50 270 0 )
                      StartDirection ( 0 0 0 )
              )
              ORTSAlternatePassengerViewPoint (
                      PassengerCabinHeadPos ( -0.5 2.35801 -1.791 )
                      RotationLimit ( 50 270 0 )
                      StartDirection ( 0 0 0 )
              )
              ORTSAlternatePassengerViewPoint (
                      PassengerCabinHeadPos ( 0.9 2.35801 -1.791 )
                      RotationLimit ( 50 270 0 )
                      StartDirection ( -5 -90 0 )
              )
      )
)
```

At runtime, when in passenger view, the player may pass from one viewpoint to the other by pressing Shift-5.

9.6 Bell animation

Open Rails supports bell animation. The bell animation matrix must be named ORTSBELL within the engine's .s file. Its default frame rate is 8 frames per second. The default frame rate may be modified through the optional parameter ESD_ORTSBellAnimationFPS (n), to be inserted within the .sd file related to the .s file. n defines the animation FPS. It is advisable that the related sound stream within the .sms file is synchronized with the visible animation. To accomplish this the .wav file should contain two bell strokes, which time interval is equal to the time interval of a bell swing from an oscillation end point to the opposite end point. As the first bell stroke should not start immediately, but when the bell is about at the maximum of the swing, the first stroke within the .wav file should be at the time distance equivalent to the oscillation from center point to an oscillation end point. The file should have one cue point at its beginning and one after the time interval of a complete bell swing forward and backward, and should have a final fadeoff for best result.

9.7 Coupler and Airhose Animation

Open Rails supports animation of couplers and air hoses. Coupler animation will move the couplers and air hoses as the train moves and the coupler slack increases or decreases. Couplers will also rotate as the train travels around a curve.

To implement this separate models need to be provided for the couplers and air hoses. A separate model for the coupled and uncoupled state is suggested.

To enable coupler animation the following parameters need to be included in the coupler code section of the WAG file:

`FrontCouplerAnim` - Coupler shape to be displayed at the front of the car when it is coupled. `FrontCouplerOpenAnim` - Coupler shape to be displayed at the front of the car when it is uncoupled. `RearCouplerAnim` - Coupler shape to be displayed at the rear of the car when it is coupled. `RearCouplerOpenAnim` - Coupler shape to be displayed at the rear of the car when it is uncoupled

All four of the above will have the following format:

CouplerAnimation (couplershape.s, x, y, z) where the coupler shape file name is included along with x, y, z values that offset the coupler in the three axis.

For the airhose animation the following parameters must be included in the coupler code section of the WAG file:

`FrontAirHoseAnim` - Air hose shape to be displayed at the front of the car when it is coupled. `FrontAirHoseDisconnectedAnim` - Air hose shape to be displayed at the front of the car when it is uncoupled. `RearAirHoseAnim` - Air hose shape to be displayed at the rear of the car when it is coupled. `RearAirHoseDisconnectedAnim` - Air hose shape to be displayed at the rear of the car when it is uncoupled.

Each of these parameters will have the same format as indicated above for the coupler shapes.

Open rails uses some defaults to calculate the required movement and angles for coupler and air hose shape movement, however for greater accuracy the modeler can add specific values such as `ORTSLengthAirHose`. In addition the length values suggested in the Derailment Coefficient should also be added.

9.8 C# engine scripting

To simulate especially complex behavior, Open Rails provides a C# scripting interface for a number of systems on the player locomotive. Like the Open Rails program itself, these scripts are written in .cs files containing C# classes, but they are compiled and linked at runtime, so they don't depend on changes in the core program itself and can be distributed with rolling stock content. Scripts will run if referenced by OR-specific fields in the .eng file.

Table 1: Currently scriptable locomotive systems

System	C# class	.eng block
Train brake controller	ORTS.Scripting.Api.BrakeController	Engine (ORTSTrainBrakeController ("DemoBrakes.cs"))
Engine brake controller	ORTS.Scripting.Api.BrakeController	Engine (ORTSEngineBrakeController ("DemoBrakes.cs"))
Circuit breaker	ORTS.Scripting.Api.CircuitBreaker	Engine (ORTSCircuitBreaker ("DemoBreaker.cs"))
Traction cut-off relay	ORTS.Scripting.Api.TractionCutOffRelay	Engine (ORTSTractionCutOffRelay ("DemoRelay.cs"))
Diesel power supply	ORTS.Scripting.Api.DieselPowerSupply	Engine (ORTSPowerSupply ("DemoPower.cs"))
Electric power supply	ORTS.Scripting.Api.ElectricPowerSupply	Engine (ORTSPowerSupply ("DemoPower.cs"))
Passenger car power supply	ORTS.Scripting.Api.PassengerCarPowerSupply	Wagon (ORTSPowerSupply ("DemoPower.cs"))
Train Control System	ORTS.Scripting.Api.TrainControlSystem	Engine (ORTSTrainControlSystem ("DemoTCS.cs"))

Scripts reside in a `Script` subfolder within the engine's folder and must contain a class named after the script's own filename. For example, if the script's filename is `AmtrakTCS.cs`, OR will search for a single class named `AmtrakTCS`. (It is also possible to place the script in another location, such as a `Common.Script` folder in the `TRAINSET` folder, by prepending the appropriate amount of parent directory tokens `..\` relative to the engine's `Script` folder.) The script's code runs on the UpdaterProcess thread. This example, which would need to be placed in a file named `DemoTCS.cs`, illustrates the minimum code required for a Train Control System script:

```
using System;
using ORTS.Scripting.Api;

namespace ORTS.Scripting.Script
{
    class DemoTCS : TrainControlSystem
    {
        public override void HandleEvent(TCSEvent evt, string message) {}
        public override void Initialize()
        {
            Console.WriteLine("TCS activated!");
        }
        public override void SetEmergency(bool emergency) {}
        public override void Update() {}
    }
}
```

Observe that the script's class *must* reside in the `ORTS.Scripting.Script` namespace and that it subclasses the abstract class of the desired system. It also references external assemblies with `using` directives. OR makes the following .NET assemblies available to scripts:

- System
- System.Core
- ORTS.Common
- Orts.Simulation

Scripts communicate with the simulator by invoking methods in the base class. For example, this script

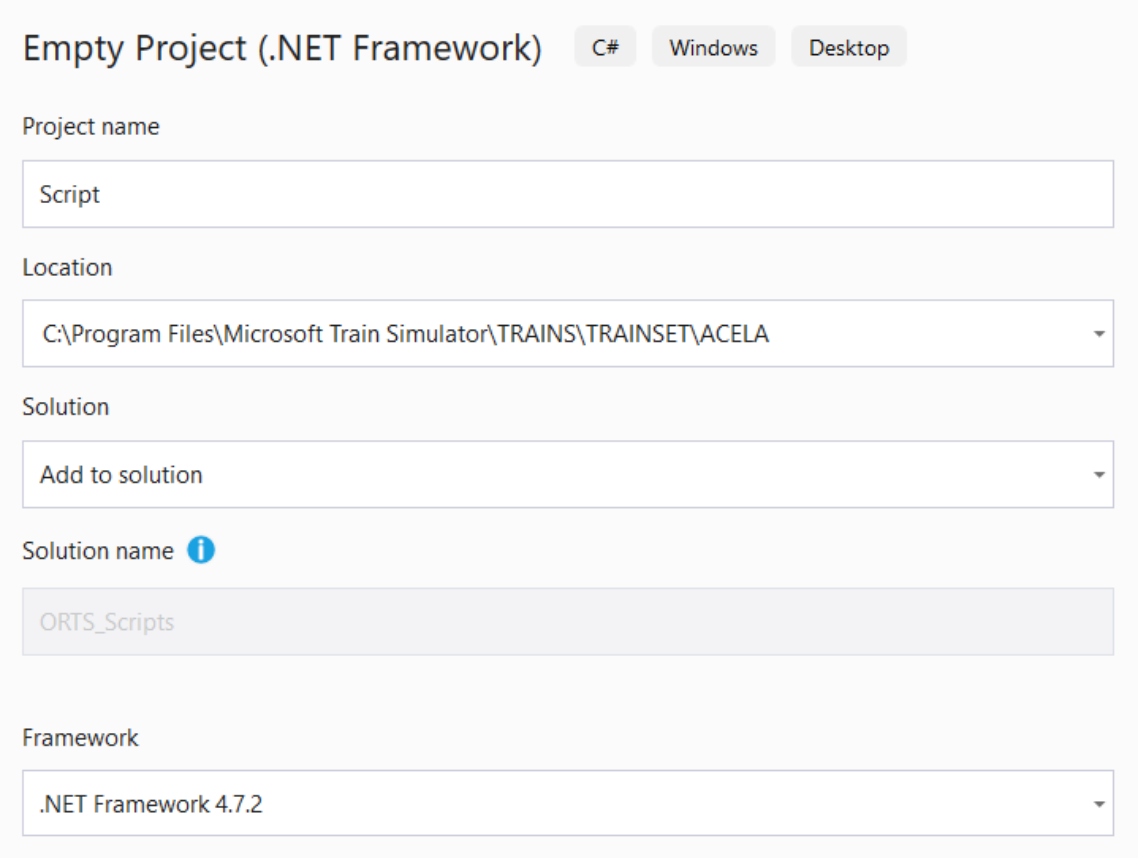
might invoke the `TrainLengthM()` method of the `TrainControlSystem` class, which returns the length of the player train. More methods are available in the `ORTS.Scripting.Api.AbstractScriptClass` class, which `TrainControlSystem` is itself a subclass of.

Finally, if a script contains a syntax or typing error, OR will log an exception during the loading process and run the simulation without it.

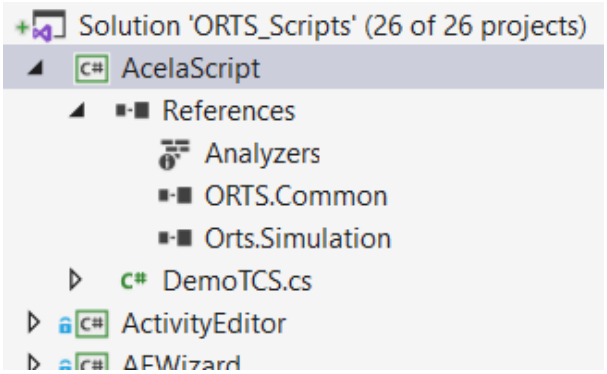
9.8.1 Developing scripts with Visual Studio

While it is certainly possible to develop scripts with a plain text editor, the code completion and debugging aids available in an IDE like Visual Studio make for a vastly more comfortable programming experience. If you have a development environment set up to build Open Rails, you can use Visual Studio to edit your scripts with these creature comforts. What follows is a suggested workflow:

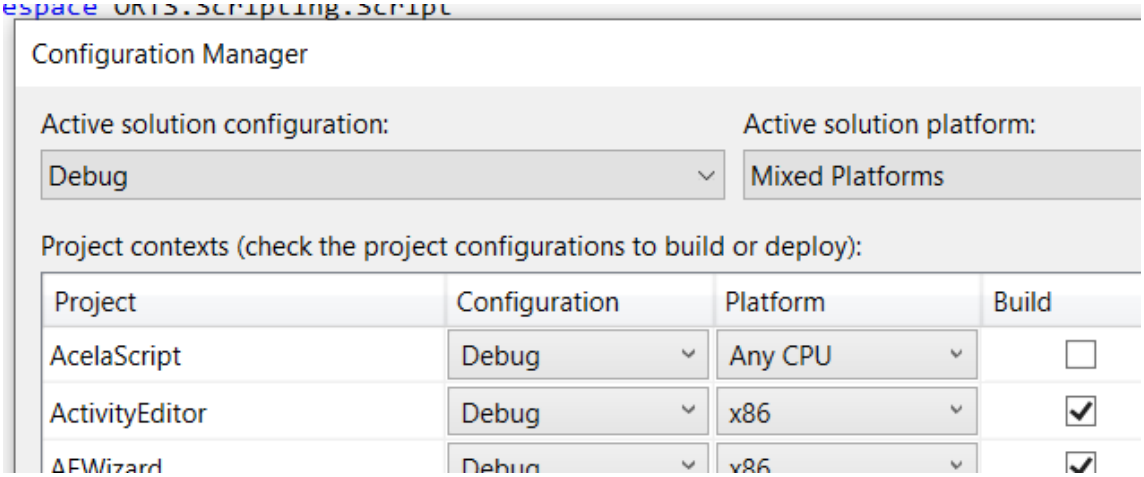
1. First, in your copy of the OR source code, make a copy of your `Source\ORTS.sln` file. Keep it in the `Source\` folder, but give it a novel name like `ORTS_Scripts.sln`. (You could also modify the original ORTS solution, but then you'd have to remember not to check it in to source control.) Add a new project to the solution and select the empty .NET project.
2. In the configuration dialog, set the new project to be added to the existing solution, set its location to be the folder of the engine you're scripting, and set its name to "Script". (For now, you must use "Script", but you can rename the project after it's created.) You can leave the .NET framework version set to its default. Then, create the project.



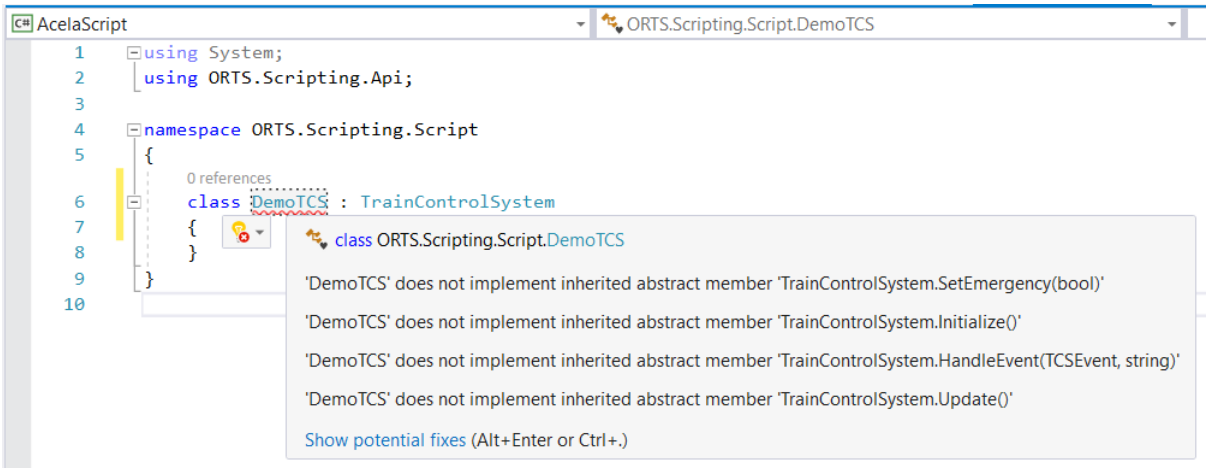
3. The new project folder becomes the very `Script` subfolder that OR will search for scripts. Add references to the ORTS.Common and Orts.Simulation assemblies, which will enable IntelliSense features inside your editor when you edit scripts. You may now rename the project as you like (which will not rename the folder) and delete the pregenerated App.config file.



4. Finally, open the Build Configuration Manager and set the new script project not to build for both the Debug and Release configurations.



With this setup, Visual Studio will type-check your scripts and make suggestions when you use the Open Rails API. You can also set breakpoints within your script, which will be caught by RunActivity.exe if run inside Visual Studio.



Note that Visual Studio uses relative paths, so if you ever move any folders, you'll need to fix the references by hand.

9.8.2 Brake controller

The brake controller script customizes the behavior of the train's brakes, allowing for much greater systems fidelity compared to what is possible with the model inherited from MSTS. For this purpose, the script can read the state of the brake controls and set the air pressures of the brake reservoirs.

Use the following .eng parameter to load a brake controller script:

```
Engine (
    ORTSTrainBrakeController ( "YourBrakes.cs" )
)
```

or:

```
Engine (
    ORTSEngineBrakeController ( "YourBrakes.cs" )
)
```

The .cs extension is optional. "MSTS" loads the default MSTS-compatible implementation, so do *not* use this name for your own script.

9.8.3 Circuit breaker

Available for electric locomotives only. The circuit breaker script controls the behavior of the locomotive's *circuit breaker*.

Use the following .eng parameter to load a circuit breaker script:

```
Engine (
    ORTSCircuitBreaker ( "YourCB.cs" )
    ORTSCircuitBreakerClosingDelay ( 2s )
)
```

`ORTSCircuitBreaker` refers to the circuit breaker script in the engine's `Script` subfolder. For this field, the .cs extension is optional. "Automatic" and "Manual" load the generic OR circuit breaker implementation, so do *not* use these names for your own script.

`ORTSCircuitBreakerClosingDelay` refers to the delay between the closing command of the circuit breaker and the effective closing of the circuit breaker.

9.8.4 Traction cut-off relay

Available for diesel locomotives only. The traction cut-off relay script controls the behavior of the locomotive's *traction cut-off relay*.

Use the following .eng parameter to load a traction cut-off relay script:

```
Engine (
    ORTSTractionCutOffRelay ( "YourTCOR.cs" )
    ORTSTractionCutOffRelayClosingDelay ( 2s )
)
```

`ORTSTractionCutOffRelay` refers to the traction cut-off relay script in the engine's `Script` subfolder. For this field, the .cs extension is optional. "Automatic" and "Manual" load the generic OR traction cut-off relay implementation, so do *not* use these names for your own script.

`ORTSTractionCutOffRelayClosingDelay` refers to the delay between the closing command of the traction cut-off relay and the effective closing of the relay.

9.8.5 Diesel and electric power supply

Available for diesel and electric locomotives only. The power supply script determines whether or not the locomotive is serviceable (see also the description of *the diesel power supply* and *the electric power supply*) given the current line voltage, pantograph position, circuit breaker state, etc. It is also capable of forbidding some operations related to the power supply if some conditions are not met.

Use the following .eng parameter to load a power supply script:

```
Engine (
    ORTSPowerSupply ( "YourEPS.cs" )
    ORTSPowerOnDelay ( 5s )
    ORTSAuxPowerOnDelay ( 10s )
)
```

`ORTSPowerSupply` refers to the power supply script in the engine's `Script` subfolder. For this field, the .cs extension is optional. "Default" will load the generic OR power supply implementation, so do *not* use this name for your own script.

`ORTSPowerOnDelay` refers to the delay between the closing of the circuit breaker or the traction cut-off relay and the availability of the power for traction.

`ORTSAuxPowerOnDelay` refers to the delay between the closing of the circuit breaker or the traction cut-off relay and the availability of the power for auxiliary systems.

9.8.6 Passenger car power supply

Available for passenger cars using electric heating. The power supply script determines whether or not the systems of the cars have power and calculates the power consumption on the *Electric Train Supply*.

If the locomotive is a diesel locomotive, the power consumed by the cars is no longer available for traction.

Use the following .wag parameter to load a power supply script:

```
Wagon (
    ORTSPowerSupply ( "YourEPS.cs" )
    ORTSPowerOnDelay ( 5s )
    ORTSPowerSupplyContinuousPower ( 500W )
    ORTSPowerSupplyHeatingPower ( 2kW )
    ORTSPowerSupplyAirConditioningPower ( 3kW )
    ORTSPowerSupplyAirConditioningYield ( 0.9 )
    ORTSHeatingCompartmentTemperatureSet ( 20degC )
)
```

`ORTSPowerSupply` refers to the power supply script in the wagon's `Script` subfolder. For this field, the .cs extension is optional. "Default" will load the generic OR power supply implementation, so do *not* use this name for your own script.

`ORTSPowerOnDelay` refers to the delay between the availability of the power on the Electric Train Supply cable and the availability of the power for the systems (for example, start-up of the static power converter).

`ORTSPowerSupplyContinuousPower` refers to the power which is consumed continuously (for example, battery chargers, lights, etc.).

`ORTSPowerSupplyHeatingPower` refers to the power which is consumed when the heating is active.

`ORTSPowerSupplyAirConditioningPower` refers to the power which is consumed when the air conditioning (cooling) is active.

`ORTSPowerSupplyAirConditioningYield` refers to the yield of the air conditioning (ratio of the heat flow rate by the electric power of the air conditioning system).

`ORTSHeatingCompartmentTemperatureSet` refers to the desired temperature inside the car.

9.8.7 Train Control System

General

The Train Control System, or TCS, script is intended to model train safety and cab signalling systems. It can manipulate the locomotive's controls and speed limit displays, impose penalty brake applications, read upcoming signal aspects and speed limits, and play warning sounds.

Use the following .eng parameters to load a TCS script:

```
Engine (
    ORTSTrainControlSystem ( "YourTCS.cs" )
    ORTSTrainControlSystemParameters ( "YourTCS.ini" )
    ORTSTrainControlSystemSound ( "YourTCSSounds.sms" )
)
```

`ORTSTrainControlSystem` refers to the TCS script in the engine's `Script` subfolder. For this field, the .cs extension is optional.

`ORTSTrainControlSystemParameters`, an optional field, refers to an .ini file, also in the `Script` subfolder, whose parameters will be made available to the TCS script through the `GetBoolParameter()`, `GetIntParameter()`, `GetFloatParameter()`, and `GetStringParameter()` methods of the `TrainControlSystem` class. This .ini file provides for easy customization of the behavior of the TCS script by end users.

This is an excerpt from an .ini file:

```
[General]
AWSMonitor=true
EmergencyStopMonitor=false
VigilanceMonitor=true
OverspeedMonitor=false
DoesBrakeCutPower=true
BrakeCutsPowerAtBrakeCylinderPressureBar=
[AWS]
Inhibited=false
WarningTimerDelayS=3
BrakeImmediately=false
TrainStopBeforeRelease=false
ActivationOnSpeedLimitReduction=true
SpeedLimitReductionForActivationMpS=11.176
BeaconDistanceToPostM=1186
AppliesCutsPower=true
```

As can be seen, the .ini file is divided in subgroups. As an example, parameter [AWS]Inhibited would be read by following line of code in the script :

```
AWSInhibited = GetBoolParameter("AWS", "Inhibited", false);
```

where the final `false` is the default value, if the parameter can't be found.

`ORTSTrainControlSystemSound`, an optional field, refers to a .sms file either in the engine's `SOUND` folder or in the global `SOUND` folder. If provided, OR will load this sound library alongside the locomotive's standard cab sounds. The TCS script can play back sounds using any of the `TriggerSound...` methods of the base class, which in turn activate the TCS-related *discrete triggers* numbered from 109 through 118.

8 further generic discrete sound triggers are available, named GenericEvent1 to GenericEvent8 and accessible to the script by lines like following one:

```
SignalEvent(Event.GenericEvent1);
```

Access to the Simulation methods and variables

The abstract class for the TCS scripts provides a significant amount of methods to access variables of interest for the TCS: as an example:

```
public Func<int, Aspect> NextSignalAspect;
```

might be called within the script as follows:

```
var nextSignalAspect = NextSignalAspect(1);
```

which would return the aspect of the second normal signal in front of the player train.

However it is quite impossible to foresee all needs that a TCS script has and to provide a method for everyone of these needs. For this reason following method is available:

```
public Func<MSTSLocomotive> Locomotive;
```

which returns a handle for the player locomotive instance of the MSTSLocomotive class. Through such handle all public classes, methods and variables of the OR Simulation environment can be accessed within the script.

The Train Control System class provides the ETCSStatus field, which controls the information to be displayed in the ETCS DMI. For example, the following block orders the DMI to show the circular speed gauge in yellow colour as the train approaches a speed restriction:

```
ETCSStatus.CurrentMonitor = Monitor.TargetSpeed;
ETCSStatus.CurrentSupervisionStatus = SupervisionStatus.Indication;
ETCSStatus.TargetDistanceM = 1234.5f;
ETCSStatus.AllowedSpeedMpS = 50;
ETCSStatus.InterventionSpeedMpS = 52.5f;
ETCSStatus.TargetSpeedMpS = 25;
```

Emergency braking triggered by the simulator

The emergency brakings triggered by the simulator are always sent to the TCS script.

Two functions are used to transmit this information:

```
public override void HandleEvent(TCSEvent evt, string message)
```

The events sent are `EmergencyBrakingRequestedBySimulator`, `EmergencyBrakingReleasedBySimulator` and `ManualResetOutOfControlMode`. For the first event, the reason of the emergency braking is also sent:

- `SPAD`: The train has passed a signal at danger at the front of the train
- `SPAD_REAR`: The train has passed a signal at danger at the rear of the train
- `MISALIGNED_SWITCH`: The train has trailed a misaligned switch
- `OUT_OF_AUTHORITY`: The train has passed the limit of authority
- `OUT_OF_PATH`: The train has ran off its allocated path
- `SLIPPED_INTO_PATH`: The train has slipped back into the path of another train
- `SLIPPED_TO_ENDOFTRACK`: The train has slipped off the end of the track
- `OUT_OF_TRACK`: The train has moved off the track

- OTHER_TRAIN_IN_PATH: Another train has entered the train's path
- SLIPPED_INTO_TURNTABLE: The train has entered a misaligned turntable
- TRAIN_ON_MOVING_TURNTABLE: The train has started moving on a moving turntable

```
public override void SetEmergency(bool emergency)
```

This function is deprecated and will be deleted in a future version. The parameter indicates if the emergency braking is requested (true) or released (false).

Generic cabview controls

Often Train Control Systems have a quite sophisticated DMI (driver-machine interface), which can include a (touch screen) display and buttons. Being the display fields and icons and the buttons specific of every TCS, a set of generic cabview controls are available, which can be customized within the TCS script. More precisely 48 generic cabview controls, named from ORTS_TCS1 to ORTS_TCS48 are available. All 48 may be used as two state or multistate controls, like e.g.:

```
MultiStateDisplay (
                    Type ( ORTS_TCS13 MULTI_STATE_DISPLAY )
                    Position ( 405 282.3 36.3 20.8 )
                    Graphic ( ../../Common.Cab/Cruscotto_SCMT/Ripetizioni_estese.ace )
                    States ( 6 3 2
                            State (
                                    Style ( 0 )
                                    SwitchVal ( 0 )
                            )
                            State (
                                    Style ( 0 )
                                    SwitchVal ( 1 )
                            )
                            State (
                                    Style ( 0 )
                                    SwitchVal ( 2 )
                            )
                            State (
                                    Style ( 0 )
                                    SwitchVal ( 3 )
                            )
                            State (
                                    Style ( 0 )
                                    SwitchVal ( 4 )
                            )
                            State (
                                    Style ( 0 )
                                    SwitchVal ( 5 )
                            )
  )
)
```

Each one of the first 32 can be also used as Two-state commands/displays, like e.g.:

```
TwoState (
        Type ( ORTS_TCS7 TWO_STATE )
        Position ( 377 298 9 7.8 )
        Graphic ( ../../Common.Cab/Cruscotto_SCMT/Button_SR.ace )
        NumFrames ( 2 2 1 )
```

(continues on next page)

(continued from previous page)

```
        Style ( PRESSED )
        MouseControl ( 1 )
)
```

The commands are received asynchronously by the script through this method:

```
public override void HandleEvent(TCSEvent evt, string message)
```

Where evt may be TCSEvent.GenericTCSButtonPressed or TCSEvent.GenericTCSButtonReleased and message is a string ranging from "0" to "31", which correspond to controls from ORTS_TCS1 to ORTS_TCS32. The commands may only be triggered by the mouse, except the first two which may also be triggered by key combinations `Ctrl,` (comma) and `Ctrl.` (period). Here's a code excerpt from the script which manages the commands:

```
public override void HandleEvent(TCSEvent evt, string message)
{
    if (message == String.Empty)
    {
        switch (evt)
        {
            case ...
                                        ...
                break;

            case ...
                                        ...
                break;
         }
    }
    else
    {
        var commandEvent = TCSCommandEvent.None;
        var messageIndex = 0;
        if (Int32.TryParse(message, out messageIndex))
        {
            commandEvent = (TCSCommandEvent)(messageIndex + 1);
            switch (evt)
            {
                case TCSEvent.GenericTCSButtonPressed:
                    TCSButtonPressed[(int)commandEvent] = true;
                    break;
                case TCSEvent.GenericTCSButtonReleased:
                    TCSButtonPressed[(int)commandEvent] = false;
                    TCSButtonReleased[(int)commandEvent] = true;
                    break;
            }
        }
    }
}
```

Within the Update method of the script TCSButtonPressed and TCSButtonReleased may be tested, e.g.:

```
if (TCSButtonPressed[(int)(TCSCommandEvent.Button_Ric)])
```

After having tested it, TCSButtonPressed should be set to false by the script code.

You can also use `TCSEvent.GenericTCSSwitchOff` and `TCSEvent.GenericTCSSwitchOn` for a cabview control representing a switch (style ONOFF instead of PRESSED in the CVF file).

To request a display of a cabview control, method:

```
public Action<int, float> SetCabDisplayControl;
```

has to be used, where `int` is the index of the cab control (from 0 to 47 corresponding from ORTS_TCS1 to ORTS_TCS48), and `float` is the value to be used to select among frames.

When the player moves the mouse over the cabview controls linked to commands, the name of such control shortly appears on the display, like e.g. "speedometer", as a reminder to the player. In case of these generic commands, strings from "ORTS_TCS1" to "ORTS_TCS32" would appear, which aren't mnemonic at all. Therefore following method is available:

```
public Action<int, string> SetCustomizedCabviewControlName;
```

which may be used this way within the script:

```
// Initialize customized TCS cabview control names
SetCustomizedCabviewControlName(0, "AWS acknowledge"); // Sets the name "AWS acknowledge"␣
↪for the cabview control ORTS_TCS1
```

so that, instead of ORTS_TCSnn the related mnemonic string is displayed.

9.8.8 Helper classes

3 helper classes are available in the `Orts.Scripting.Api` namespace:

- A timer class
- An odometer class
- A blinker class

Timer

The timer can be used to execute some code after a time has elapsed. In order to use the timer, you have to create a property in your script class in order to store the object.

```
public Timer MyTimer;
```

In the constructor of your script class, you have to instanciate the object and set the delay of the timer.

```
MyTimer = new Timer(this);
MyTimer.Setup(5f); // Sets the timer's delay to 5 seconds
```

Then, when you want to start the timer, use the `Start` function.

```
MyTimer.Start();
```

If you want to reset the timer, use the `Stop` function.

```
MyTimer.Stop();
```

When the delay has been reached, the `Triggered` property of the timer will become `true`.

```
if (MyTimer.Triggered)
{
    // Do something
}
```

Please note that, when the timer is stopped, the `Triggered` property is `false`.

Odometer

The odometer can be used to execute some code after a distance has been traveled by the train. In order to use the odometer, you have to create a property in your script class in order to store the object.

```
public Odometer MyOdometer;
```

In the constructor of your script class, you have to instanciate the object and set the distance at which the odometer will be triggered.

```
MyOdometer = new Odometer(this);
MyOdometer.Setup(200f); // Sets the odometer's trigger value at 200 meters
```

Then, when you want to start the odometer, use the `Start` function.

```
MyOdometer.Start();
```

If you want to reset the odometer, use the `Stop` function.

```
MyOdometer.Stop();
```

When the distance has been reached, the `Triggered` property of the odometer will become `true`.

```
if (MyOdometer.Triggered)
{
    // Do something
}
```

Please note that, when the odometer is stopped, the `Triggered` property is `false`.

Blinker

The blinker can be used to make a cabview control blink. In order to use the blinker, you have to create a property in your script class in order to store the object.

```
public Blinker MyBlinker;
```

In the constructor of your script class, you have to instanciate the object and set the frequency at which the cabview control will blink.

```
MyBlinker = new Blinker(this);
MyBlinker.Setup(6f); // Sets the blinker frequency to 6 Hz
```

Then, when you want to start the blinker, use the `Start` function.

```
MyBlinker.Start();
```

If you want to reset the blinker, use the `Stop` function.

```
MyBlinker.Stop();
```

The blinker `On` property will alternate between `true` and `false` at the set frequency.

```
SetCabDisplayControl(0, MyBlinker.On ? 1 : 0);
```

Please note that, when the blinker is stopped, the `On` property is `false`.

CHAPTER 10

Open Rails Train Operation

Note that this document details behaviour while in single-player mode only. For *multi-player mode*, different rules may apply.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

10.1 Open Rails Activities

OR has the aim of running in a compatible way most of the activities written for MSTS.

Also, activities specifically for OR can be created, using the additional functions OR features, like *Extended AI Shunting*. Discussions of the execution of some functions in ORTS and MSTS are given here.

10.1.1 Player Paths, AI Paths, and How Switches Are Handled

If the player path requires a switch to be aligned both ways, the alignment that is the last on the path is used. If an AI train crosses the player path before the player train gets there, the AI train will leave the switches aligned for the main route (the default setting for most switches)

If you throw a switch to move into a siding, the switch at the far end of the siding is aligned to let you leave when your train first occupies the siding. But after that it is not changed back to its original setting. If the switch gets thrown the other way, you can leave the siding with the switch aligned incorrectly. If you uncouple and re-couple to the train while it occupies the misaligned switch, the rear end of the train will switch tracks.

10.2 Open Rails AI

Basic AI Functionality

- OR supports AI trains. The AI system is becoming more and more advanced with new features.
- OR supports two distinct ways of controlling trains: it supports traditional activities in compatibility with MSTS, and it also supports *Timetable* mode. Note that various options and settings are sometimes limited to either activity or Timetable mode.
- AI trains can meet if both paths have passing sections defined at the same place, or if their paths lead them to different tracks at the meet station.
- Waiting points and reverse points work. Reverse points can be used in both Activity and Timetable modes, while waiting points can only be used in Activity mode.
- AI trains throw switches not lined properly before engaging them.
- In activity mode AI trains can perform shunting actions.
- Priorities: AI trains should start as scheduled as long as there is no other AI train already on a conflict path.

10.3 Control Mode

Control Mode defines what interactions there are between the player and the control system, and the level of control of the player on signals and switches.

There are two basic modes: *Auto Mode* and *Manual Mode*.

Use the <Ctrl+M> key to toggle between these modes.

10.3.1 Auto Mode

In Auto Mode the control system sets the train's path and signals, and the player cannot change the setting of the switches or request for signals at danger to clear. The train's route is taken from the path as defined in the Activity Editor or timetable definition, and the system will attempt to clear the route ahead of the train according to the signalling rules and interaction with other trains.

No route is cleared in the reverse direction as the train is assumed not to run in reverse. Selecting a reverse cab or changing the position of the reverser does not change the direction of the route. In fact, the route will not be reversed other than at reversal points as defined in the train's path. At these reversal points, the route will reverse automatically as soon as the train stops.

If the train does accidentally run backward, e.g. due to slipping or setting back after overshooting a platform, only safety checks are performed for the rear end of the train with respect to signals, switch alignment, other trains and end of track. There is no check on speed limits behind the train.

Setting switches using the F8 window or <G>/<Shift+G> is not allowed. Setting switches using Alt+left mouseclick is possible, but is not allowed for switches in the train's path. However, any switches set manually will automatically be reset by an approaching train according to that train's path. So, in Auto Mode the train cannot deviate from the defined path.

A request to clear a signal ahead of the train using the Tab command is only allowed when the track ahead is occupied by another train which is at a stand-still, and when that track is in the train's route. A request to clear a signal which would lead the train off its route is not allowed. A request to clear a signal behind the train using Shift+Tab is also not possible.

Auto Mode is intended for normal running under control of signals or traffic control. Shunting moves can be performed if fully defined in the train's path, using reversal points etc..

Details on Auto Mode: *Auto Signal* & *Auto Node*

There are two sub-modes to *Auto Mode*: *Auto Signal* and *Auto Node*.

Auto Signal is the normal mode on signalled routes. The train's route is generally cleared from signal to signal. Only in specifically defined situations can routes be cleared short of a signal as detailed below.

Auto Node is set when the train has not encountered any signals yet, e.g. on unsignalled routes or at the start of the route when there is no signal along the path of the train as far as it can be cleared - e.g. in yards where the train starts but has no clear route yet to the first signal.

Auto Node can also be set if the route ahead cannot be fully cleared up to the next signal, and partial clearing is allowed.

A number of sub-states are defined in Auto Node, depending on the reason that clearance is terminated. In the list below, (**A**) indicates a subtype which can occur if no signal has yet been encountered, (**B**) indicates a subtype when a route from a signal is partially cleared.

The following states are possible :

- (**A**) route ahead is clear to the maximum distance for which the track is cleared. The control mode is set to Auto Node – Max Distance.
- (**A**) route ahead is blocked at a switch which is aligned for and occupied or reserved by another train. Control mode is set to Auto Node – Misaligned Switch.
- (**A**)(**B** – only if signal allows access to occupied track, or after <Tab> command) route ahead is occupied by a stationary train or train moving in the same direction. Control mode is set to Auto Node – Train Ahead.
- Note that, for (**A**), it should not be possible that the route ahead is occupied by a train moving in opposite direction - in that case, there should always be a misaligned switch in the train's path.
- For (**B**), a signal will never clear when the train ahead is moving in the opposite direction, nor will the Tab request be granted.
- (**A**)(**B**) the train's defined path terminates short of the next signal, or there is a reversal point short of the next signal, and there is at least one switch between this point and the next signal. The control mode changes to Auto Node – End of Path. Note that if there is no switch between the terminating or reversal point and the next signal the route is automatically extended to the next signal.
- (**A**)(**B**) the train has passed the last signal before the end of the track, or the train has reached the end of track without encountering any signal. The control mode changes to Auto Node – End of Track.

Changes from Auto Node to Auto Signal and vice-versa are automatic and cannot be influenced by the player.

10.3.2 Manual Mode

When it is required that a train move off its defined path, a player can switch his train to Manual Mode. This will allow the player to set switches and request to clear signals off its path. However, there are a number of restrictions when running a train in Manual Mode.

In Manual Mode, a route is cleared from the train in both directions, ahead of and behind the train. The route is cleared to a shorter distance as compared to Auto Mode, and is never cleared automatically beyond the first signal. If a train is moving and passes a signal in the opposite direction, the route behind the train will automatically retract to that signal as that is now the next signal in the reverse route. The same restrictions apply with respect to signals ahead when the train is running in reverse.

The route orientation will not change whatever direction the train is running. It is fixed to the orientation of the route as it was the moment the player switched to Manual Mode. So, changing to a reverse-facing cab or changing the position of the loco's reverser does not change the direction of the route orientation. This is not a limitation to the train's behaviour, as routes are always cleared in both directions. It does, however, affect the display of the F4 and F8 windows, as the top/bottom direction of these windows is

linked to the route direction and will therefore not change if the train reverses. To assist the player in his orientation in which direction the train is moving, an “eye” has been added to these displays symbolizing the direction of the cabview, and an “arrow” has been added to symbolize the direction of the reverser.

The player can set all switches in the train’s path using the F8 window or the <G>/<Shift+G> keys. The G key will set the first switch ahead of the train (as defined by the route direction), Shift+G sets the switch behind the train. It is also possible to set switches as required using the Alt+Left Mouseclick command. Switches can be set even if they are in the train’s path and a signal has been cleared over that path. Switches, of course, can not be set if already set as part of a cleared route for another train.

The following rules apply to the setting of switches :

- all switches will remain in the position in which they were set by the last train passing over that switch. If no train has yet passed over the switch, it is in its default position.
- when in Manual Mode, trailing switches will not be automatically aligned for the approaching player train, except :
- when a route is cleared through a signal while in Manual Mode, any trailing switches in the train’s path up to the end of authority (e.g. next signal) will be aligned. Note that in this case, trailing switches in the path cleared by the signal can no longer be reset.

Signals which the train approaches will not be cleared automatically. The player must request clearance of all signals encountered, by using the <Tab> or <Shift+Tab> keys.

The <Tab> key will clear the signal ahead of the train (according to the route direction), the <Shift+Tab> key will clear the signal behind the train. Repeated use of (<Shift> +)``<Tab>`` will clear the next signal beyond the first cleared signal etc., but only up to the maximum clearing distance.

Signals will always clear on request except when the section immediately behind the signal is already cleared for a train from the opposite direction. The normal route-setting limitations etc. are ignored. The signal will only clear to the first available most restrictive aspect above Stop.

Note that, in contrast to the situation in Auto Mode, as the signal will clear even if the full route behind the signal is not available, a cleared signal is no indication of the cleared distance beyond that signal. It may be that the first switch beyond the signal is already cleared for another train. Therefore, when in Manual Mode, use of the F4 window or the Dispatcher window to check on the route availability is essential when running in an area with AI traffic.

When in Manual Mode, deadlock prevention processing is switched off. This is because the changes in the train’s route and direction which are likely to occur in Manual Mode could jeopardise the stability of the deadlock processing. So care should be taken when using Manual Mode in an area with AI traffic, specifically on single track sections.

Switching from Auto Mode to Manual Mode can be performed with the train at a standstill or with the train moving. The <Ctrl+M> key toggles between Auto Mode and Manual Mode. When switching from Auto Mode to Manual Mode, all signals already cleared will be reset, and new routes are cleared ahead of and behind the train for the maximum distance if possible, or up to the first signal.

To switch back from Manual Mode to Auto Mode the front of the train must be on the path as defined in the Activity Editor. If the path contains reversal points, the train must be in between the same reversal points as it was when it switched to Manual Mode (i.e. same subpath).

If the train is moving in the direction as the path defines, switching back to Auto Mode can be done while the train is moving. The rear of the train need not be on the defined path, only the front.

If the train is moving in the opposite direction, it must be at a standstill in order to switch back to Auto Mode. If the orientation of the train’s route was somehow reversed (e.g. by moving through a balloon-line or a Y-section) and differs from the direction in the defined path, both the front and rear must be on the defined path. In this situation, the orientation will switch back to the direction as defined in the path.

10.3.3 Out-of-Control Mode

This is a special mode. Normally, the player train should not be in this mode. The out-of-control mode is activated when the player violates a security rule. Such incidents are:

- when the player train passes a signal at danger (SPAD);
- when the player train passes over a misaligned switch;
- when the player train runs beyond the end of the authorised path.

These actions will place the player train into out-of-control mode. In this situation, the emergency brake is activated and maintained until the train is stopped. The player has no control over his train until it is at a standstill.

Once the train has stopped, the player can switch to Manual Mode to try to return to a correct situation (e.g. get back to in front of the signal at danger, authorised path etc.). Once a normal situation has been restored, the player can switch back to Auto Mode. If the action led the player train onto a section of track already cleared for another train, that train is also stopped.

10.3.4 Explorer Mode

When OR is started in Explorer Mode instead of in an activity, the train is set to Explorer Mode. The player has full control over all switches. Signals will clear as normal but signals can be cleared over routes which are not normally available using the `<Tab>` or `<Shift+Tab>` commands.

10.4 Track Access Rules

All trains clear their own path. When in Auto Signal mode, part of that function is transferred to the signals.

In *Auto Node* mode, trains will clear their path up to 5000 metres, or the distance covered in 2 mins at the maximum allowed speed, whichever is greater. In *Auto Signal* mode, the number of signals cleared ahead of the train is taken from the value of the `SignalNumClearAhead` parameter as defined in the `sigcfg.dat` file for the first signal ahead of the train.

In Manual mode, the distance cleared is 3000 metres maximum, or as limited by signals.

Distances in Explorer Mode are similar to those in Auto Mode.

If a train is stopped at a signal it can claim the track ahead ensuring it will get priority as the next train onto that section, but to avoid needless blocking of other possible routes, no claim is made if the train ahead is also stopped.

No distinctions are made between types of train, and there are no priority rules.

10.5 Deadlock Processing

When a train is started, it will check its path against all other trains (including those not yet started). If a section is found on which this train and the other train are due in opposite directions, the boundaries of that total common section are determined, and *deadlock traps* are set at those boundaries, for each train in the appropriate direction. These boundaries are always switch nodes. When a train passes a node which has a *deadlock trap* for that train, the trap is sprung. When a train approaches a node which has an active deadlock, it will stop at that node, or at the last signal ahead of it if there is one. This train will now also spring its deadlock traps, and will claim the full common section of that deadlock to ensure it will be the next train allowed onto that section. The deadlock traps are removed when a train passes the end node of a deadlock section.

When a train is started, and the train's path includes one or more reversal points, deadlocks are only checked for the part of the path up to the first reversal point. On reversal, deadlocks are checked for the next part, etc..

Deadlock traps are removed when a train switches to Manual mode. When the train switches back to Auto mode, the deadlock check is performed again.

There are no deadlock checks in Explorer Mode as there are no AI trains when running in this mode.

If an alternative path is defined (using the Passing Path definition in MSTS Activity Editor), and the train is setting a route to the start node of this alternative path, it will check if a deadlock is set for the related end node. If so, and the alternative path is clear, it will take the alternative path, allowing the other train to use the main path. If the alternative path is already occupied, the train will wait short of the node where the path starts (or the last signal in front, if any); this is to prevent blocking both tracks which would leave the opposite train nowhere to go.

Further rules for the use of alternative paths :

- Trains from both direction must have the same main path through the area.
- If only one train has an alternative path defined, and the trains are to pass, that train will always use the alternative path; the other train will always use the main path regardless of which train arrives first.
- If both trains have an alternative path defined, and the trains are to pass, the first train to clear its route will take the alternative path. Note that this need not always be the first train to arrive - it could be that the train which first clears its path takes much longer to actually get to the passing loop.

10.6 Reversal Points

If a reversal point is defined, the path will be extended beyond that point to the end of the section, this is to the next switch or signal or the end of track.

The *diverging* point is determined – this is the switch node where the reverse route diverges from the incoming route. From this point, a search is made for the last signal facing the reverse direction which is located such that the full train will fit in between the signal and the end of the path. If there is such a signal, this will become the *diverging* point. In order for a train to be able to reverse, the rear of the train must be clear of this *diverging* point.

Reversal for AI trains occurs as in MSTS; that is, when the AI train's first car reaches the reversal point. If at that point the rear of the train has not yet cleared the diverging point, the reversal takes place later, when the diverging point is cleared.

For player trains the reversal can take place starting from 50 meters before the reversal point provided the diverging point is cleared. The colour of the reversal point icon in the *Track Monitor* is green if the *diverging* point has been cleared (meaning that the player train may already revert, even if it hasn't yet reached the reversal point), while it is white in the opposite case (meaning that the player train must proceed further towards the diverging point, eventually reaching it if colour does not change to green, before reverting).

As in MSTS, double reversal points can be used to set a signal at red after such reversal points. However waiting points are recommended for this, as explained in the next paragraph.

10.7 Waiting Points

10.7.1 General

Waiting points (WP) set in a path used by an AI train are regularly respected by the train, and executed when the head of the train reaches the WP.

Differently from MSTS, waiting points do not influence the length of the reserved path, except when the WP is followed by a signal in the same track section (no nodes – that is switches – in between).

WPs set in a path used by a player train have no influence on the train run, except – again – when the WP is followed by a signal in the same track section. In such cases, for both AI trains and player train, the signal is set to red when the train approaches the WP.

For AI trains the signal returns to green (if the block conditions after the signal allow this) one second after expiration of the WP.

For player trains the signal returns to green 5 seconds after expiration of the WP.

If there are more WPs in the track section where the signal resides, only the last one influences the signal.

Waiting points cannot be used in Timetable mode.

10.7.2 Absolute Waiting Points

Waiting points with a *waiting time* between 30000 and 32359 are interpreted as absolute time-of-day waiting points, with a format 3HHMM, where HH and MM are the hour and minute of the day in standard decimal notation.

If the AI train will reach the WP before this time of day, the WP will expire at HH:MM. If the AI train will reach the WP later, the WP will be alreay expired. This type of WP can also be used in conjunction with a signal in the same track section, as explained in preceding paragraph.

Again, such waiting points won't have an effect on a player train if there is no signal in the same section; if instead there is a signal, it will stay red until the WP has expired.

Absolute waiting points are a comfortable way of synchronizing and scheduling train operation.

10.8 Signals at Station Stops

If the Experimental Option *Forced red at station stops* has been selected, and if there is a signal at the end of a platform, that signal will be held at danger up to 2 minutes before the booked departure. If the station stop is less than 2 minutes, the signal will clear as the train comes to a stand. This applies to both AI train and player trains.

However, if the platform length is less than half the train length, the signal will not be held but will clear as normal to allow the train to properly position itself along the platform. Signals which only protect plain track will also not be held.

In some railway control systems trains do not get a red at the station starting signal when they have to stop in that station. In these cases the above option must be disabled.

10.9 Speedposts and Speed Limits Set by Signals

Speed limits which raise the allowed speed, as set by speedposts or signals, only become valid when the rear of the train has cleared the position of speedpost or signal.

When a speed limit set by a signal is lower than the speed limit set by the last speedpost, the speed limit is set to the lower value. However, when a speed limit as set by a signal is higher than the present speed limit set by the last speedpost, the limit defined by the speedpost will be maintained. If a lower speed limit was in force due to a limit set by another signal, the allowed limit is set to that as defined by the speedpost.

In timetable mode if a speedpost sets a limit which is higher than that set by the last signal, the limit set by the signal is overruled and the allowed limit is set to that as defined by the speedpost.

In activity mode in the preceding case the lower of the two limits becomes valid.

10.10 Further Features of AI Train Control

- AI trains always run in Auto control mode.
- AI trains will ignore any manual setting of switches and will reset all switches as defined in their path.
- AI trains will stop at stations and will adhere to the booked station departure times if possible.
- AI trains will stop at a platform such that the middle of the train is in the middle of the platform. If the train is longer than the platform, both the front and rear of the train will extend outside the platform. If the platform has a signal at the end, and this signal is held at danger (see further *above*), and the train is too long for the platform, it will stop at the signal. But if the train length is more than double the platform length, the signal will not be held.
- AI trains will adhere to the speed limits.
- AI trains will stop at a signal approximately 30 m. short of a signal at danger in Timetable mode, and at a shorter distance in activity mode.
- Where AI trains are allowed to follow other trains in the same section passing permissive signals, the train will adjust its speed to that of the train ahead, and follow at a distance of approx. 300 m. If the train ahead has stopped, the train behind will draw up to a distance of about 50m. However, if the train ahead is stopped in a station, and the train behind is also booked to stop at that station, the train will draw up behind the first train up to a distance of a few metres.
- The control of AI trains before the start of an activity is similar to the normal control during an activity, except that the update frequency is reduced from the normal update rate to just once per second. But all rules regarding speed limits, station stops, deadlock, interaction between AI trains (signals etc.) are followed. The position of all AI trains at the start of an activity therefore is as close as possible to what it would have been if the activity had been started at the start time of the first AI train.

10.11 Location-linked Passing Path Processing

Passing paths can be used to allow trains to pass one another on single track routes. The required passing paths are defined per train path in the MSTS Activity Editor or in the native ORTS path editor included within TrackViewer.

The present version is an ‘intermediate’ stage leading to complete new processing. The data structure and processing have already been prepared for the next stage, when ‘alternative paths’ (not just a single passing path but multiple paths through a certain area) will be defined per location, and no longer per train.

The present version, however, is still based on the MSTS activity and path definition, and therefore is still based on the definition of alternative paths per train.

The setup of this version is as detailed below :

- Passing paths defined for the *player* train are available to *all* trains – in both directions. The 'through' path of the player train is taken to be the "main" path through that location. This only applies to Activity mode, as there is no predefined player train when running in Timetable mode.
- Each train can have definitions for additional passing paths, these will be available to that train only. Note that this implies that there can be more than one passing path per location.
- When possible passing locations are determined for each pair of trains, the train lengths are taken into consideration. A location is only 'valid' as a passing location if at least one of the trains fits into the shortest of the available passing paths.
- The order in which passing paths are selected:
  - If no train is approaching from the opposite direction (through route):
    * Train's own path.
    * "Main" path.
    * Any alternative path.
  - If train is to pass another train approaching from the opposite direction (passing route):
    * Train's own path (if not the same as "main" path).
    * Alternative path.
    * "Main" path.

However, in the situation where the train does not fit on all paths, for the first train to claim a path through the area, preference is given to the paths (if any) where the train will fit.

The setting of the 'deadlock' trap (the logic which prevents trains from getting on a single track from both directions) has also been changed.

In the 'old' version, the trap was 'sprung' as a train claimed its path through a possible passing area.

However, this often lead to quite early blocking of trains in the opposite direction.

In this version the trap is 'sprung' when a train actually claims its path in the single track section itself.

One slight flaw in this logic is that this can lead to the train which is to wait being allocated to the "main" path, while the train which can pass is directed over the "loop". This can happen when two trains approach a single track section at almost the same time, each one claiming its path through the passing areas at either end before the deadlock trap is actually sprung.

If a passing location contains platforms and there are passenger trains which are booked to stop there, OR will try to locate an alternate platform on the passing path, and if it can find it, this platform will replace the original one as the stop platform. This behavior occurs only if the *Location-linked Passing Path Processing* option has been checked.

Selecting this type of passing path with the related experimental option processing can lead to considerable changes in the behaviour of trains on single track routes – and behaviour that is certainly significantly different from that in MSTS.

10.12 Other Comparisons Between Running Activities in ORTS or MSTS

10.12.1 End of run of AI trains

AI trains end their run where the end point of their path resides, as in MSTS. However they always end their run at zero speed.

10.12.2 Default Performance and Performance Parameters

If the AI train does not make station stops, its maxspeed (not considering signal, speedpost and route speed) is given by the first MaxVelocity parameter in the .con file, expressed in meters per second, multiplied by the "Default performance" parameter (divided by 100) that can be found and modified in the MSTS AE in the "Service editor". Such parameter divided by 100 is written by the AE in the .srv file as "Efficiency".

If the AI train makes station stops, its maxspeed depends from the "Performance" parameter for every route section, as can be seen and defined in the AI train timetable (that is maxspeed is the product of the first MAxVelocity parameter by the "Performance" parameter divided by 100).

Such performance parameter list is written (divided by 100) by the AE in "Service_Definition" block in the activity editor, again as "Efficiency" (for every station stop).

From the starting location of the AI train up to the first station, the "Performance" linked to such station is used; from the first station to the second one, the "Performance" linked to the second station is used and so on. From the last station up to end of path the "Default performance" mentioned above is used.

This corresponds to MSTS behaviour.

Moreover the Efficiency parameter is used also to compute acceleration and braking curves.

10.12.3 Calculation of Train Speed Limit

For the player train: speed limit is the lowest among:

- route speed limit as defined in the `.trk` file
- local signal speed limit
- local speedpost speed limit
- local temporary speedpost speed limit
- first parameter `MaxVelocityA` in `.con` file, if bigger than zero and not equal 40
- locomotive speed limit in `.eng` file in the other cases.

For the AI trains: speed limit is the lowest among:

- route speed limit as defined in the `.trk` file
- local signal speed limit
- local speedpost speed limit
- local temporary speedpost speed limit
- first parameter `MaxVelocityA` in `.con` file, if bigger than zero and not equal 40
- locomotive speed limit in `.eng` file in the other cases.
- route speed limit as defined in the `.trk` file
- local signal speed limit

- local speedpost speed limit
- local temporary speedpost speed limit
- first parameter `MaxVelocityA` in `.con` file, if bigger than zero, multiplied by the Efficiency as explained :ref:`here <operation-performance>`.

10.12.4 Start of Run of AI train in a Section Reserved by Another Train

The AI train is created as in MSTS. It is up to the activity creator not to generate deadlocks. Creation of a train in a section where another train resides is possible only if the created train is not front-to-front with the existing train.

10.12.5 Stop Time at Stations

The platform passenger number as defined by the MSTS activity editor is read by OR.

Each passenger requires 10 seconds to board. This time must be divided by the number of passenger wagons within the platform boundaries. Also locomotives with the line `PassengerCapacity` in their .eng file count as passenger wagons (EMU, DMU). The criterion to define if a passenger wagon is within the platform boundaries is different for player trains and AI trains. For player trains an individual check is made on every passenger wagon to check if it is within the plaform boundaries (it is assumed that this is OK if at least two thirds of the wagon are within). For AI trains instead the number of wagons + engines within the platform is computed, and all of them, up to the number of the passenger wagons in the consist, are considered as passenger wagons. The player or AI train boarding time is added to the real arrival time, giving a new departure time; this new departure time is compared with the scheduled departure time and the higher value is selected as the real departure time.

A train is considered to be a passenger train if at least one wagon (or engine) carries passengers.

AI real freight trains (0 passenger cars) stop 20 seconds at stations as in MSTS if scheduled starting times are not present. If they are present the freight trains will stop up to the scheduled starting time or up to the real arrival time plus 20 seconds, whichever is higher.

A special behaviour has been introduced for trains with more than 10 cars and having a single passenger car. This type of train has been used in MSTS to have the possibility of also defining schedules for freight trains. These trains are managed – like MSTS – as passenger trains with the rules defined above. However a simplification for the player has been introduced for the player train: if the train stops with the single passenger car outside of the platform, the stop is still considered valid.

All this is compatible with MSTS operation; only the fact that the scheduled departure time is considered for AI trains differs, as it is considered an improvement.

10.12.6 Restricted speed zones defined in activities

OR manages restricted speed zones defined in activities as MSTS. Start of a restricted speed zone can be recognized on the Track Monitor Window because the maxspeed is shown in red; the maxspeed at an end of a restricted speed zone is shown in green.

10.13 Extended AI Train Shunting

10.13.1 General

Having AI trains performing shunting operations ensures more interesting and varied activities.

Note that this feature is not available in Timetable mode, which has other ways to perform AI Train shunting.

The following additional shunting functions are available:

1. AI train couples to a static consist and restarts with it.
2. AI train couples to a player or AI train and becomes part of it; the coupled train continues on its path.
3. AI train couples to a player or AI train and leaves to it its cars; the coupled and coupling train continue on their path.
4. AI train couples to a player or AI train and *steals* its cars; the coupled and coupling train continue on their path.
5. AI train uncouples any number of its cars; the uncoupled part becomes a static consist. With the same function it is possible to couple any number of cars from a static consist.
6. AI train couples to a player or AI train; the resulting combined train runs for part of the path, then stops; the train is split there into two parts that continue on their own paths (*join and split* function).
7. AI train can get permission to pass a signal at danger.

These functions are described in detail below.

A sample activity can be found in `Documentation\SampleFiles\Manual\Show_AI_shunting_enh.zip`.

10.13.2 Activity Design for Extended AI Train Shunting Functions

Activity design can be performed with the MSTS Activity Editor, and does not need post-processing of the created files.

Extended AI Functions 1 to 4 (these all involve coupling)

It is not always desired that AI trains couple to other trains; e.g. the activity could have been designed so that the trains proceed separately, but then, at runtime, they could be at the same place at the same moment because of timing problems. In such a case it would be undesirable that the trains couple. So coupling is activated only if certain conditions are met.

In general the signal protection rules apply, that is, an AI train will find a red signal if its path leads it directly to another train. So in general these functions can be used only if there are no signals between the coupling train and the coupled train. However, this can be overcome in three modes:

- by the activity developer, by inserting a double reversal point between the signal and the coupled train (this works only if the double reversal point is not in the track section occupied by the coupled train).
- by the player, forcing the signal to the clear state by using the *dispatcher window*.
- or even better, by using extended AI shunting function #7, which is described further below, that allows the AI train to pass a signal at danger.

Coupling with a static consist is not subject to other conditions, since if the activity designer decided that the path would lead an AI train up to against a static consist, it was also desired that the AI train would couple to it.

Coupling with another AI train or with the player train is subject to the following conditions. Either:

- the coupling happens in the last path section of the coupling AI train, and the path end point is under the coupled train or beyond it in the same section, or
- the coupling happens in the last section before a reverse point of the coupling AI train, and the reverse point is under the coupled train or beyond it in the same section.

In this way undesired couplings are avoided in case the AI train has its path running in the same direction beyond the coupled train.

Just after coupling OR performs another check to define what happens next.

In the case where the coupled train is static:

- if there is at least one reverse point further in the path or if there are more than 5 track sections further in the path, the coupling train couples with the static train, and then the resulting formed train restarts following the path of the coupling train, or
- if not, the coupling train couples with the static train and becomes part of the static train itself (is absorbed by it), stopping movement.

In case the coupled train is a player train or an AI train:

- if there is at least one reverse point under the coupling train or further in the same track section, the coupling train couples with the coupled train; at that point there are two possibilities:
    1. The trainset coupling to the coupled train is a wagon: in this case the coupling train leaves to the coupled train all the cars between its locomotive and the coupled train, decouples and moves further in its own path (it can only reverse due to above conditions). The coupled train follows its own path.
    2. The trainset coupling to the coupled train is a locomotive: in this case the coupling train *steals* from the coupled train all the cars between the coupled train's locomotive and the coupling train, decouples and moves further in its own path (it can only reverse due to the above conditions). The coupled train follows its own path.
- or if there is no reverse point further in the path of the coupling train, the coupling train couples with the coupled train and becomes part of it (is absorbed by it). The coupled train follows its own path.

Now on how to design paths:

- If one wants the coupling train to be absorbed by the coupled train: simply put the end point of the path of the coupling train below the coupled train or further, but in the same track section.
- If one wants the coupling train to move further on in its path after having coupled with the coupled train: put in the path of the coupling train a reverse point below the coupled train. If one also wants that the coupling train does not immediately restart, but that it performs a pause, a waiting point has to be added in the path of the coupling train, subsequent to the reverse point. It is suggested to put the waiting point near the reverse point, and in any case in the same track section. OR will execute the waiting point even if it is not exactly below what remains of the coupling train after coupling/decoupling is only the locomotive.
- If the coupled train is an AI train, obviously it must be stopped on a waiting point when it has to be coupled by the coupling train.

Extended AI Function 5 (AI train uncouples any number of its cars)

To uncouple a predefined number of cars from an AI train, a special waiting point (WP) has to be inserted.

The format of this waiting point (in decimal notation) is usually 4NNSS, where NN is the number of cars in front of the AI train that are NOT uncoupled, locomotive included, and SS is the duration of the waiting point in seconds.

The 5NNSS format is also accepted. In this case the remaining AI train is formed by NN cars (locomotives included) starting from the rear of the train. Of course there must be at least one locomotive in this part of the train.

It must be noted that the "front" of the AI train is the part which is at the front of the train in the actual forward direction. So, if the consist has been created with the locomotive at first place, the locomotive will be at the front up to the first reverse point. At that point, "front" will become the last car and so on.

The following possibilities arise:

- The AI train proceeds and stops with the locomotive at the front, and wants to uncouple and proceed in the same direction: a WP with the format 4NNSS is inserted where the AI train will stop, counting cars starting from the locomotive.
- The AI train proceeds with the locomotive at the rear, and wants to uncouple and proceed in the reverse direction: a reverse point has to be put in the point where the train will stop, and a 4NNSS WP has to be put sequentially after the reverse point, somewhere under the part of the train that will remain with the train, formatted as above. As the train has changed direction at the reverse point, again cars are counted starting from the locomotive.
- The AI locomotive proceeds and couples to a loose consist, and wants to get only a part of it: a reverse point is inserted under the loose consist, and a 4NNSS WP is inserted sequentially after the reverse point, somewhere under the part of the train that will remain with the train, formatted as above.

What is NOT currently possible is the ability to couple the AI train to the player train or to another AI train, and to "steal" from it a predefined number of cars. With the currently available functions it is only possible to steal all the cars or to pass all the cars. If it is desired that only a number of cars be passed from an AI or player train to the other, the first AI train has to uncouple these cars as described above, then move a bit forward, and then make the second AI train couple to these cars.

Function 6 (Join and split)

Introduction

Join and split means that two trains (AI or player) each start running on their own path; then they join and run coupled together a part of their path, and then they split and run further each on its own path (in the same direction or in opposite directions).

This can have e.g. the following example applications:

Application 1:

- a pair of helper locomotives couples to the rear or to the front of a long train;
- the resulting train runs uphill;
- when they have arrived uphill, the helper locomotives uncouple from the train.
  - if the helpers were coupled to the rear of the other train, the train continues forward on its path, while the helper locomotives return downhill.
  - If the helpers were coupled to the front, the helpers will enter a siding and stop; the train will continue forward on its path, and when the train has passed, thee helpers can reverse and return downhill.

  This means that a complete helper cycle can be simulated.

Application 2:

- a passenger train is formed from two parts that join (e.g. two sections of a HST);
- the train reaches an intermediate station and the two sections decouple;
- one section takes the main line, while the other one takes a branch line (this can happen in any direction for both trains).
- Both the joining train (the one that moves and couples to the other train – the joined train) and the joined train may be an AI train or a player train.

Activity development

1) The two trains start as separate trains, couple together and decouple later in the game . After that of course such trains can couple to other trains, and so on.
2) The coupling train becomes an "Incorporated" train after coupling, that is it has no more cars or locomotives (they all become part of the coupled train) and is a sort of virtual train. In this phase it is not shown in the Dispatcher information HUD. It will return to life when an uncoupling command (automatic or manual) is issued.
3) To become an "Incorporated" train, the coupling train if of AI type, must pass in its path before coupling over a Waiting Point with value 60001 (the effective waiting time is 0 seconds); such WP is not necessary if the coupling train is the player train.
4) For the coupling train to couple to the rear of the coupled train there are no particular requirements; if however you want to have very short runs from coupling train start to coupling moment, it could be necessary to insert a couple of reversal points in between, or else the train could stop and avoid coupling. Please don't disdain double reversals: they are sometimes the only way to limit the authority range of a train.
5) If the coupling train has to couple to the front of the coupled train, obviously a reversal point is needed for the coupling train: it must be laid somewhere under the coupled train, or even farther down in the same track section; also in this case there can be a problem of authority, that could require that the coupled train has a couple of reversal points after the point where it waits to be coupled.
6) The incorporated train has its own path, but from coupling to decoupling point it must pass over the same track sections of the path of the incorporating train. The incorporated train must not have waiting points nor station stops in the common path part (the coupled train instead may have them). If there are reversals within the common path part, they must be present in both paths.
7) At the point of decoupling the number of cars and locomotives to be decoupled from the train can be different from the number of the original train.
8) The whole train part to be decoupled must lie on the same track section. After decoupling, the "incorporated" train returns to being a standard AI train.
9) Manual decoupling (for player trains) occurs using the F9 window; automatic decoupling occurs with the 4NNSS and 5NNSS commands (see previous paragraph); the first one has to be used when the part to be decoupled is at the rear of the train, and the second one where the part is at the front of the train.
10) In the standard case where the main part of the train continues in the same direction, the following cases can occur:
    - If the decoupled part is on the front, this decoupled part can only proceed further in the same direction (ahead of the main part of the train). To avoid it starting immediately after decoupling, it is wise to set a WP of some tens of seconds in the path of the decoupled train. This WP can be set at the beginning of the section where decoupling occurs; OR will move it under the decoupled part, so you don't need to be precise in positioning it.
    - If the decoupled part is on the rear, two cases are possible: either the decoupled part reverses or the decoupled part continues in the same direction. In the first case a reversal point has to be put anywhere in the section where the decoupling occurs (better towards the end of the section), and OR will move it to the right place so that the train reverses at the point where

decoupling occurred; moreover it is also advised to put a WP of some tens of seconds, so that the train does not restart immediately. This WP must be located logically after the reversal point, and in the same track section; OR will move it under the decoupled train.

- If the decoupled part continues in the same direction, neither WP nor RP are needed. This train part will wait that the part ahead will clear the path before starting.

Activity run hints

- When you run as player, you have to uncouple the train where foreseen by the activity (the uncoupled train must lay in a route section present in its path). If you don't uncouple on a track section present in the path of the uncoupled train, the uncoupled train will become a static train, because it's not on its path.
- You can run the train formed by the original train plus the incorporated train from any cab (also in a cab of the incorporated train). However before uncoupling (splitting) the trains, you have to return to a cab of the original train.

Function 7 (Permission to pass signal at danger for AI train)

During AI train shunting there are cases where it is necessary that the AI train is conditionally able to pass a red signal, in a similar way of the player trains when pressing TAB.

This can be accomplished by defining a specific WP with value 60002 to be laid down in the AI train path before the signal to be passed (in the track section just in front of the signal).

10.14 Signal related files

For content developers

OR manages signals as defined in the files `sigcfg.dat` and `sigscr.dat` in a way that is highly compatible to MSTS. A description of their contents and how to modify these two files is contained in the Word document `How_to_make_Signal_config_and_Script_files.doc` that is found in the `TECH DOCS` folder of an MSTS installation. Note that these files must be edited with a Unicode text editor.

Additionally, a C# scripting interface is available to complement the `sigscr.dat`` file for more complex systems.

10.14.1 SignalNumClearAhead

Specific rules, however, apply to the sigcfg.dat parameter `SignalNumClearAhead ()`, that is not managed in a consistent way by MSTS.

In this paragraph the standard case is discussed, where sigcfg.dat and sigscr.dat are located in the root of of the route.

If for a SignalType only one SignalNumClearAhead () is defined (as is standard in MSTS files), then this parameter defines the number of NORMAL signal heads (not signals!) that are cleared down the route, including the signal heads of the signal where the SignalType resides. This is not exactly as in MSTS, where quite complex and strange calculations are perfomed, and in some cases could lead to too few signals being cleared for a satisfactory train operation. Moreover MSTS doesn't consider the SignalNumClearAhead () value related to the signal, but the maximum SignalNumClearAhead () encountered in the signal types used in the route. Therefore, if it is desired that OR approaches the MSTS operation, the value of SignalNumClearAhead ()of all signals must be set at the same maximum value. To avoid affecting also MSTS operation, there are two approaches that are described here below.

If for a SignalType a second SignalNumClearAhead () parameter is added just before the existing one, OR interprets it as the number of NORMAL SIGNALS that are cleared down the route, including the signal where the SignalType resides.

MSTS will skip this first SignalNumClearAhead () and will consider only the second. In this way this change to sigcfg.dat does not affect its use in MSTS.

However, instead of modifying the copy of the file sigcfg.dat residing in the route's root, the approach described in the next paragraph is recommended.

10.14.2 Location of OR-specific sigcfg and sigscr files

By simply copying the original sigscr.dat and sigcfg.dat into a subfolder named `OpenRails` created within the main folder of the route, OR will no longer consider the pair of files located in the route's root folder, and will interpret the (single) SignalNumClearAhead () line as defining the number of signals cleared. So OR interprets sigscr.dat in a different way, depending whether there is a copy of this file in the `OpenRails` subfolder or not. In this way the problem of too few signals cleared for satisfactory train operation is usually solved.

If however this single line standard sigscr.dat doesn't behave satisfactorily even counting signals (a reason has been described in preceding paragraph), it will have to be optimized for OR by modifying the parameter SignalNumClearAhead () for the unsatisfactory signals; if preferred the line can stay as it is, and an optimized line can be added before the existing one, and it will again count signals. In this case the sigscr.dat file behaves the same as if it would if located in the route's root folder.

Sigcfg.dat must keep its name, while the sigscr files can also have other names, provided that within sigcfg.dat there is a reference to these other names.

10.14.3 OR-unique values for SignalNumClearAhead ()

OR recognizes two additional unique values of the parameter SignalNumClearAhead (), when this parameter is located on a line preceding the line with the MSTS value, or if the sigcfg.dat file is located in the subfolder `OpenRails`:

- 0 : no signal will be cleared beyond this signal until train passes this signal.
- -1: signal does not count when determining the number of signals to clear.

10.14.4 C# signal scripting

To simulate especially complex behavior, Open Rails provides a C# scripting interface for signals. These scripts are written in .cs files containing C# classes, but they are compiled and linked at runtime, so they don't depend on changes in the core program itself and can be distributed with the route.

C# signal scripts are placed in the `Script/Signal` subfolder within the main folder of the route. All C# files present in that folder will be compiled together at runtime into a single assembly.

For each signal type defined in the `sigcfg.dat` file, OR tries to find a class with the same name as the signal type in the compiled assembly. If there are compile errors or no class with the required name is found, the script defined in the `sigscr.dat` file will be used instead, if there is an adequate script there.

Each signal script must be inside the `ORTS.Scripting.Script` namespace and has to inherit from the `CsSignalScript` class, which contains all the API functions available for the script.

This example illustrates the minimum code required for a signal script:

```
using System;
using Orts.Simulation.Signalling;

namespace ORTS.Scripting.Script
{
    public class MYSIGNALTYPE : CsSignalScript
    {
```

(continues on next page)

(continued from previous page)

```
        public override void Initialize()
        {
            // Perform some initializations here, taking into account
            // that no route information is available at this point
        }
        public override void Update()
        {
            // Set the aspect of your signal here depending on route state
        }
        public override void HandleSignalMessage(int signalId, string message) {}
    }
}
```

For a list of the API calls available for signal scripts, refer to the `Orts.Simulation/Simulation/Signalling/CsSignalScript.cs` file in the OR source code.

A development environment can be set up to accelerate development process. See the *engine scripting* section for further information.

10.15 OR-specific Signaling Functions

A set of powerful OR-specific signaling functions are available. Sigcfg and sigscr files referring to these functions must be located as described in the previous paragraph.

10.15.1 SPEED Signals – a New Signal Function Type

The SPEED signal function type allows a signal object marker to be used as a speed sign.

The advantages of such a use are :

- The signal object marker only applies to the track on which it is placed. Original speed signs always also affect any nearby lines, making it difficult and sometimes impossible to set a specific speed limit on just one track in complex areas.
- As a signal object, the SPEED signal can have multiple states defined and a script function to select the required state, e.g. based on route selection. This allows different speed limits to be defined for different routes through the area, e.g. no limit for the main line but specific limits for a number of diverging routes.

The SPEED signal is fully processed as a speed limit and not as a signal, and it has no effect on any other signals.

Limitation: it is not possible to define different speeds related to type of train (passenger or freight).

Definition and usage

The definition is similar to that of any other signal, with `SignalFnType` set to `SPEED`.

It allows the definition of drawstates and aspects like any other signal. Different speed values can be defined per aspect as normal.

An aspect can be set to not have an active speed limit. If this aspect is active, the speed limit will not be changed. This can, for instance, be used if a route-linked speed limit is required. This aspect can then be set for a route for which no speed limit is required.

An aspect can also be set to not have an active speed limit but with a special signal flag : `OR_SPEEDRESET`.

If this flag is set, the speed limit will be reset to the limit as set by the last speed limit sign. This can be used to reset any limit imposed by a specific signal aspect. Note that this does not overrule any speed limits set by another SPEED signal as those limits are processed as if set by a speed limit sign.

Example 1:

```
SignalType ("SpeedSignal"
    SignalFnType ( SPEED )
    SignalLightTex ( "ltex" )
    SignalDrawStates ( 5
        SignalDrawState ( 0
            "speed25"
        )
        SignalDrawState ( 1
            "speed40"
        )
        SignalDrawState ( 2
            "speed50"
        )
        SignalDrawState ( 3
            "speed60"
        )
        SignalDrawState ( 4
            "speed70"
        )
    )
    SignalAspects ( 5
        SignalAspect ( APPROACH_1    "speed25"    SpeedMPH ( 25 ) )
        SignalAspect ( APPROACH_2    "speed40"    SpeedMPH ( 40 ) )
        SignalAspect ( APPROACH_3    "speed50"    SpeedMPH ( 50 ) )
        SignalAspect ( CLEAR_1       "speed60"    SpeedMPH ( 60 ) )
        SignalAspect ( CLEAR_2       "speed70"    SpeedMPH ( 70 ) )
    )
    SignalNumClearAhead ( 2 )
)
```

Notes:

- The SignalNumClearAhead value must be included to satisfy syntax but has no function.
- The actual speed can be set either using fixed aspect selection through user functions, or can be route linked.

The actual use is defined in the related script and the related shape definition.

Example 2:

```
SignalType ( "SpeedReset"
    SignalFnType ( SPEED )
    SignalLightTex ( "ltex" )
    SignalDrawStates ( 1
        SignalDrawState ( 0
            "Red"
        )
    )
    SignalAspects ( 1
        SignalAspect ( STOP     "Red"  signalflags (OR_SPEEDRESET) )
    )
    SignalNumClearAhead ( 2 )
)
```

This example resets the speed to the limit as set by the last speed sign, overruling any speed limits set by signal aspects.

10.15.2 Approach control functions

Approach control signals are used, specifically in the UK, to keep a signal at 'danger' until the train is within a specific distance ahead of the signal, or has reduced its speed to a specific value. Such control is used for diverging routes, to ensure the speed of the train is reduced sufficiently to safely negotiate the switches onto the diverging route.

Three script functions for use in OR have been defined which can be used to control the signal until the train has reached a specific position or has reduced its speed.

These functions are:

```
APPROACH_CONTROL_POSITION(position)
APPROACH_CONTROL_POSITION_FORCED(position)
APPROACH_CONTROL_SPEED(position, speed)
```

These functions are Boolean functions: the returned value is 'true' if a train is approaching the signal and is within the required distance of the signal and, for `APPROACH_CONTROL_SPEED`, has reduced its speed below the required values.

`APPROACH_CONTROL_POSITION_FORCED` function is similar to `APPROACH_CONTROL_POSITION`, but it can be used with any type of signal. Meanwhile, `APPROACH_CONTROL_POSITION` requires NORMAL signals, and will only clear the signal if it is the train's next signal.

Parameters :

- position : required distance of train approaching the signal, in meters
- speed : required speed, in m/s

Note that the speed is checked only when the train is within the defined distance.

Important note : although the script uses 'float' to define local variables, these are in fact all integers. This is also true for the values used in these functions : if direct values are used, these must be integer values.

The values may be set directly in the signal script, either as variables or as numbers in the function call.

However, it is also possible to define the required limits in the sigcfg.dat file as part of the signal definition.

The syntax definition for this is:

```
ApproachControlLimits ( <definitions> )
```

Allowed definitions :

- Position :
  - Positionm : position in meters.
  - Positionkm : position in kilometers.
  - Positionmiles : position in miles.
  - Positionyd : position in yards.
- Speed :
  - Speedkph : speed in km / hour.
  - Speedmph : speed in miles / hour.

These values are referenced in the script file using the following variable names :

- `Approach_Control_Req_Position`
- `Approach_Control_Req_Speed`

These variables must not be defined as floats etc., but can be used directly without prior definition.

Note that the values as defined in the sigcfg.dat file will be converted to meters and meters/sec and rounded to the nearest integer value.

The following example is for a three-head search light signal, which uses Approach Control if the route is set to the 'lower' head.

Route selection is through 'dummy' DISTANCE type route-selection signals.

Signal definition:

```
SignalType ( "SL_J_40_LAC"
    SignalFnType ( NORMAL )
    SignalLightTex ( "bltex" )
    SigFlashDuration ( 0.5 0.5 )
    SignalLights ( 8
        SignalLight ( 0 "Red Light"
            Position ( 0 6.3 0.11 )
            Radius ( 0.125 )
        )
        SignalLight ( 1 "Amber Light"
            Position ( 0 6.3 0.11 )
            Radius ( 0.125 )
        )
        SignalLight ( 2 "Green Light"
            Position ( 0 6.3 0.11 )
            Radius ( 0.125 )
        )
        SignalLight ( 3 "Red Light"
            Position ( 0 4.5 0.11 )
            Radius ( 0.125 )
        )
        SignalLight ( 4 "Amber Light"
            Position ( 0 4.5 0.11 )
            Radius ( 0.125 )
        )
        SignalLight ( 5 "Green Light"
            Position ( 0 4.5 0.11 )
            Radius ( 0.125 )
        )
        SignalLight ( 6 "Amber Light"
            Position ( 0 2.7 0.11 )
            Radius ( 0.125 )
        )
        SignalLight ( 7 "White Light"
            Position ( 0 2.7 0.11 )
            Radius ( 0.125 )
        )
    )
    SignalDrawStates ( 8
        SignalDrawState ( 0
            "Red"
            DrawLights ( 1
                DrawLight ( 0 )
            )
        )
        SignalDrawState ( 1
            "TopYellow"
            DrawLights ( 1
                DrawLight ( 1 )
```

(continues on next page)

(continued from previous page)

```
            )
        )
        SignalDrawState ( 2
            "TopGreen"
            DrawLights ( 1
                DrawLight ( 2 )
            )
        )
        SignalDrawState ( 3
            "TopYellowMidGreen"
            DrawLights ( 2
                DrawLight ( 1 )
                DrawLight ( 5 )
            )
        )
        SignalDrawState ( 4
            "MidYellow"
            DrawLights ( 2
                DrawLight ( 0 )
                DrawLight ( 4 )
            )
        )
        SignalDrawState ( 5
            "MidGreen"
            DrawLights ( 2
                DrawLight ( 0 )
                DrawLight ( 5 )
            )
        )
        SignalDrawState ( 6
            "LowYellow"
            DrawLights ( 3
                DrawLight ( 0 )
                DrawLight ( 3 )
                DrawLight ( 6 )
            )
        )
        SignalDrawState ( 7
            "LowWhite"
            DrawLights ( 3
                DrawLight ( 0 )
                DrawLight ( 3 )
                DrawLight ( 7 SignalFlags ( FLASHING ))
            )
        )
    )
    SignalAspects ( 8
        SignalAspect ( STOP                 "Red" )
        SignalAspect ( STOP_AND_PROCEED     "LowWhite"  SpeedMPH(25) )
        SignalAspect ( RESTRICTING          "LowYellow" SpeedMPH(25) )
        SignalAspect ( APPROACH_1           "MidYellow" SpeedMPH(40) )
        SignalAspect ( APPROACH_2           "TopYellowMidGreen" )
        SignalAspect ( APPROACH_3           "TopYellow" )
        SignalAspect ( CLEAR_1              "MidGreen"  SpeedMPH(40) )
        SignalAspect ( CLEAR_2              "TopGreen" )
    )
```

(continues on next page)

(continued from previous page)

```
    ApproachControlSettings (
            PositionM  ( 500 )
            SpeedMpH   ( 10  )
    )
    SignalNumClearAhead ( 5 )
)
```

Signal function (reduced to show use of approach control only). This function uses approach control for the 'lower' route.:

```
SCRIPT SL_J_40_LAC

// Searchlight Top Main Junction
extern float    block_state ();
extern float    route_set ();
extern float    def_draw_state ();
extern float    next_sig_lr ();
extern float    sig_feature ();
extern float    state;
extern float    draw_state;
extern float    enabled;
//
// Returned states
// drawn :
//      SIGASP_STOP
//
//   Top Cleared :
//      SIGASP_APPROACH_3
//      SIGASP_APPROACH_2
//      SIGASP_CLEAR_2
//
//   Middle Cleared :
//      SIGASP_APPROACH_1
//      SIGASP_CLEAR_1
//
//   Lower Cleared :
//      SIGASP_RESTRICTING
//      SIGASP_STOP_AND_PROCEED
//
// User Flags
//
// USER1 : copy top approach
// USER2 : top approach junction
// USER3 : copy middle approach
// USER4 : no check block for lower
//
float           clearstate;
float           setstate;
float           diststate;
float           adiststate;
float           nextstate;
float           routestate;
float           blockstate;

blockstate = 0;
clearstate = 0;
```

(continues on next page)

(continued from previous page)

```
routestate = 0;
setstate   = 0;
nextstate  = next_sig_lr(SIGFN_NORMAL);
diststate  = next_sig_lr(SIGFN_DISTANCE);
adiststate = diststate;

if (diststate ==# SIGASP_CLEAR_1)
{
   diststate = SIGASP_CLEAR_2;
}
if (diststate ==# SIGASP_APPROACH_1)
{
   diststate = SIGASP_APPROACH_3;
}

// get block state
if (!enabled)
{
    clearstate = -1;
}

if (block_state () ==# BLOCK_JN_OBSTRUCTED)
{
    clearstate = -1;
}

if (block_state() ==# BLOCK_OCCUPIED)
{
    blockstate = -1;
}

// check if distant indicates correct route
if (diststate ==# SIGASP_STOP)
{
    clearstate = -1;
}

// top route
state = SIGASP_STOP;

if (blockstate == 0 && clearstate == 0 && diststate ==# SIGASP_CLEAR_2)
{
    // aspect selection for top route (not shown)
    .......
}

// middle route
if (blockstate == 0 && clearstate == 0 && diststate ==# SIGASP_APPROACH_3)
{
    // aspect selection for middle route (not shown)
    .......
}

// lower route
if (blockstate == 0 && clearstate == 0 && diststate ==# SIGASP_RESTRICTING)
{
```

(continues on next page)

(continued from previous page)

```
    if (Approach_Control_Speed(Approach_Control_Req_Position, Approach_Control_Req_Speed))
    {
        state = SIGASP_RESTRICTING;
    }
}

// Get draw state
draw_state = def_draw_state (state);
```

10.15.3 TrainHasCallOn, TrainHasCallOn_Advanced Functions

This function is intended specifically to allow trains to 'call on' in Timetable mode when allowed to do so as defined in the timetable. The use of this function allows a train to 'call on' into a platform in Timetable mode without jeopardizing the functionality in normal Activity mode.

The Function TrainHasCallOn will open the Signal only if the train has arrived on the block before the Signal. If the Signal shall open earlier, use the TrainHasCallOn_Advanced Function instead, the opening of the Signal will then follow the rules of the Sigcfg.dat-Parameter SignalNumClearAhead().

It is a Boolean function and returns state as follows:

- Activity Mode :
  - Returns true if :
    - Route from signal is not leading into a platform.
- Timetable Mode :
  - Returns true if :
    - Route from signal is not leading into a platform.
    - Route from signal is leading into a platform and the train has a booked stop in that platform, and any of the following states is true:
      - Train has `$CallOn` command set for this station.
      - Train has `$Attach` command set for this station and the train in the platform is the train which it has to attach to.
      - Train is part of `RunRound` command, and is to attach to the train presently in the platform.

Additionally, both in Timetable and Activity modes, this functions will return true if the CallOn option is selected from signal's context menu in the *Dispatcher Window*.

The use of this function must be combined with a check for:

```
blockstate ==# BLOCK_OCCUPIED
```

Note : this function must **NOT** be used in combination with:

```
blockstate ==# JN_OBSTRUCTED
```

The state `JN_OBSTRUCTED` is used to indicate that the route is not accessible to the train (e.g. switch set against the train, opposite movement taking place etc.).

Some signal scripts allow signals to clear on `blockstate ==# JN_OBSTRUCTED`. This can lead to all kinds of incorrect situations. These problems are not due to programming errors but to route signal script errors.

Example (part of script only):

```
if (enabled && route_set() )
{
    if (block_state == #BLOCK_CLEAR)
    {
    // normal clear, e.g.
         state = #SIGASP_CLEAR_1;
    }
    else if (block_state == #BLOCK_OCCUPIED && TrainHasCallOn() )
    {
    // clear on occupied track and CallOn allowed
        state = #SIGASP_STOP_AND_PROCEED;
    }
    else
    {
    // track is not clear or CallOn not allowed
        state = #SIGASP_STOP;
    }
}
```

10.15.4 TrainHasCallOn_Restricted, TrainHasCallOn_Restricted_Advanced Functions

This function has been introduced because signals with call-on aspects can be used not only as entrance signals for stations, but also on 'free line' sections, that is, away from stations.

The Function TrainHasCallOn_Restricted will open the Signal only if the train has arrived on the block before the Signal. If the Signal shall open earlier, use the TrainHasCallOn_Restricted_Advanced Function instead. the opening of the Signal will then follow the rules of the Sigcfg.dat Parameter SignalNumClearAhead().

In next lines, where `TrainHasCallOn` appears, `TrainHasCallOn` and `TrainHasCallOn_Advanced` is meant; analogously, when `TrainHasCallOn_Restricted` appears, `TrainHasCallOn_Restricted` and `TrainHasCallOn_Restricted_Advanced` is meant.

`TrainHasCallOn` always allows call-on if the signal is on a 'free-line' section. This is to allow proper working for USA-type permissive signals.

Some signal systems however use these signals on sections where call-on is not allowed. For this case, the `TrainHasCallOn_Restricted` function has been introduced.

When approaching a station, both functions behave the same, but on 'free line' sections, the `TrainHasCallOn_Restricted()` will never allow call-on.

So, in a nutshell :

- Use on approach to stations:
  - `TrainHasCallOn()` and `TrainHasCallOn_Restricted()`:
    - Activity: call-on not allowed
    - Timetable: call-on allowed in specific situations (with `$callon`, `$stable` or `$attach` commands)
- Use on 'free line' :
  - `TrainHasCallOn()`:
    - Activity or Timetable: call-on always allowed
  - `TrainsHasCallOn_Restricted()`:
    - Activity or Timetable: call-on never allowed

All these functions can be set to true by hand from the *Dispatcher Window*.

These signals can be laid down with the MSTS RE. In the .tdb file only a reference to the SignalType name is written, an in the world file only a reference to the signal head is written. As these are accordingly to MSTS standards, no need to manually edit route files exists.

10.15.5 Signalling Function NEXT_NSIG_LR

This function is similar to `NEXT_SIG_LR`, except that it returns the state of the nth signal ahead.

Function call:

```
state = NEXT_NSIG_LR(MstsSignalFunction fn_type, int n).
```

Returned value:

- state of nth signal ahead, except,
  - When there are less than n signals ahead of the train.
  - when any of the intermediate signals is at danger.

  In those situations, the function will return SIGASP_STOP.

Usage : take, for instance, the sequence of signals as shown below.



The distance between signals B and C, as well as between C and D, is shorter than the required braking distance. Therefore, if D is at danger, both C and B must show yellow; similar, if C is at danger, both B and A must be yellow.

Problem now is what aspect should be shown at A : if B is yellow, is it because C is at red, so A must also be yellow, or is it because C is at yellow as D is at red – in which case A can show green. One could, of course, use two different states for yellow at C, but that soon gets rather complicated, and also one might soon run out of available aspects.

With the new function, it becomes simpler : if B is at yellow, A can directly check the state of C, and so decide if it can clear to green or must show yellow.

Suppose state `SIGASP_STOP` shows red, `SIGASP_APPROACH_1` shows yellow and `SIGASP_CLEAR_1` shows green for all signals, the related part of the script could be as follows:

```
if (next_sig_lr(SIGFN_NORMAL) == SIGASP_APPROACH_1)
{
    if (next_nsig_lr(SIGFN_NORMAL, 2) == SIGASP_STOP)
    {
        state = SIGASP_APPROACH_1;
    }
    else
    {
        state = SIGASP_CLEAR_1;
    }
}
```

The function is also very useful when a distant signal is to reflect the state of more than one home signal, but `dist_multi_sig_mr` cannot be used because there is no distant signal further on.

10.15.6 Signalling Function HASHEAD

This function can be used for any optional SIGNAL_HEAD as defined for the relevant signalshape in sigcfg.dat, to check if that has been selected for this signal or not.

Using 'DECOR' dummy heads, this allows these heads to be used as additional user settings, and as such are kind of an extension to the four available SIGFEAT_USER flags.

Please note that this function is still experimental.

Function call:

```
state = HASHEAD( n );
```

where n is the SignalSubObj-Number in question. Function returns 1 if head SignalSubObj is set, else 0.

10.15.7 Signalling flag OR_NOSPEEDREDUCTION

Differently from MSTS, as default AI trains pass signals with aspect RESTRICTED or STOP_AND_PROCEED at reduced speed. To provide also an MSTS-compatible operation and to take into account signalling systems where no a speed reduction is required when passing such signals, the flag OR_NOSPEEDREDUCTION has been introduced. This is an example of usage of such flag:

```
SignalAspects ( 7
      SignalAspect ( STOP             "Red" )
      SignalAspect ( STOP_AND_PROCEED "LowYellowFlash" SpeedMPH(25) signalflags (OR_
↪NOSPEEDREDUCTION) )
      SignalAspect ( RESTRICTING      "LowYellow"     SpeedMPH(25) signalflags (OR_
↪NOSPEEDREDUCTION) )
      SignalAspect ( APPROACH_2                       "TopYellowMidGreen" )
      SignalAspect ( APPROACH_3                       "TopYellow" )
      SignalAspect ( CLEAR_1                          "MidGreen" )
      SignalAspect ( CLEAR_2                          "TopGreen" )
)
```

With this flag set, no speed reduction is applied when passing the signal.

10.16 OR-Specific Additions to Activity Files

The additions described below will be ignored by MSTS. Since activity files are not used in Timetable mode, none of the following features will operate in that mode. You can make these additions in three different ways, which are described in following subparagraphs.

10.16.1 Manually modifying the .act file

Make these additions by modifying the .act file with a Unicode-enabled editor. Note that these additions will be removed by the MSTS Activity Editor if the .act activity file is opened and saved as an .act file by the AE. However, if the activity is opened in the AE and saved in an .apk Activity Package, the additions will instead be included.

10.16.2 Using the TSRE5 activity editing capabilities

The TSRE5 Route Editor includes activity editing capabilities. These capabilities include addition of some OR-specific additions to activity files described in following paragraphs. A note is present where this does not apply.

10.16.3 Generating an extension activity file

If the TSRE5 editor isn't used, and if it is desired to avoid the problem that the OR-specific additions are lost by later modifying the activity with the MSTS Activity Editor, it is recommended to use this third possibility: an OpenRails subfolder must be created within the route's ACTIVITIES folder, and an .act file including only the OR-specific extensions used can be created with an Unicode-enable editor and then located there. An example of an unmodified .act file and of an extension .act file within the route's OpenRails subfolder is included in file ORActivityExtensionFileSample.zip, which may be found within the `Documentation\SampleFiles\Manual` subfolder within the OpenRails folder. As can be seen, the name of such extension .act file must be the same as the one of the base .act file. Re events, to ensure a correct cross-correspondence between event definitions within the base file and within the extension file, in the extension file within the EventCategory block of every modified event the first line must be the ID () one, and the ID must correspond with the one present in the base .act file. Only the added lines within such EventCategory block must be present in the extension .act file.

10.16.4 No Halt by Activity Message Box

MSTS activities may contain instructions to display a message box when the player train reaches a specific location in the activity, or at a specific time. Normally the simulation is halted when the message box is displayed until the player manually closes the box. This behavior can be modified if the line:

```
ORTSContinue ( nn )
```

Where nn = number of seconds to display the box, is added to the event declaration (EventTypeLocation or EventTypeTime) in the .act file.

For example:

```
EventCategoryLocation (
    EventTypeLocation ( )
    ID ( 1 )
    Activation_Level ( 1 )
    Outcomes (
        DisplayMessage ( "Test nopause." )
    )
    Name ( Location1 )
    Location ( -146 14082 -1016.56 762.16 10 )
    TriggerOnStop ( 0 )
    ORTSContinue ( 10 )
)
```

Now, the activity will continue to run while the message window is displayed. If the player does nothing, the window disappears automatically after nn seconds. The player may close the window manually or pause the activity by clicking on the appropriate button in the window. Note that this modification does not work for the terminating event of the activity.

10.16.5 AI Train Horn Blow

Waiting points can be used to instruct AI trains to blow their horns at specific locations.

If the waiting time value is between 60011 (1 second horn blow) and 60020 (10 seconds horn blow), a single horn blow is generated.

If the waiting time value is 60021, a horn blow sequence is generated, with the pattern long blow - long blow - short blow - long blow (North American horn pattern at level crossings).

The AI train will not stop at these waiting points, but will continue at its regular speed.

If the lead locomotive of the AI train has parameter DoesHornTriggerBell set to 1 in the .eng file, the bell is played for further 30 seconds after the end of the horn blow.

To implement this feature it is not necessary to proceed as described in the first three paragraphs of this chapter. It is enough to insert the waiting points within the paths with either the MSTS AE or through TrackViewer.

10.16.6 AI Horn Blow at Level Crossings

Open Rails can also be instructed to have AI trains automatically blow their horns at level crossings. This feature is activated using special properties in the `Tr_Activity_File` block:

Property	Meaning
ORTSAIHornAtCrossings	Have AI trains blow their horns at level crossings — `( 1 )` for yes, `( 0 )` or omitted for no.
ORTSAICrossingHornPattern	Specifies the horn pattern blown at level crossings — `( US )` for a North American long-long-short-long pattern, `( Single )` or omitted for a single blast between 2 to 5 seconds long.

These lines **must** be placed after the `NextActivityObjectUID ( 32768 )` line, or else the activity file will become unloadable in the MSTS Activity Editor.

Simple road crossings, not defined as level crossings, may also be present in the route. The AI train will not blow the horn at these crossings. Examining the route with TrackViewer allows identification of the true level crossings. If a horn blow is also desired for a *simple* road crossing, the feature *AI Train Horn Blow* described above must be used.

If the lead locomotive of the AI train has parameter DoesHornTriggerBell set to 1 in the .eng file, the bell is played for further 30 seconds after the end of the horn blow.

10.16.7 Location Event triggered by AI Train

Under MSTS location events may only be triggered when the player train reaches them. OR provides also location events that are triggered by AI trains. In this case a line like following one must be added within the EventCategoryLocation block:

```
ORTSTriggeringTrain ( "TestEventAI" 43230 )
```

where “TestEventAI” is the service name of the AI train, and 43230 is the starting time of day (in seconds) of the AI train. The second parameter may be omitted in case there is only one AI train with the service name present in the above line.

This feature in connection with the *AI train Waiting Point modification through event* allows synchronization between AI trains or also between an AI train and the player train.

This feature is not yet managed by TSRE5.

10.16.8 Location Event and Time Event Sound File

An activity file can be modified so that a sound file is played when the train reaches a location specified in an EventTypeLocation event in the .act file, or when a certain time interval specified in an EventTypeTime event has elapsed since the start of the activity. Within the Outcomes() subblock of the event add following subblock:

```
ORTSActivitySound (
    ORTSActSoundFile ( Filename SoundType )
    ORTSSoundLocation ( TileX TileZ X Y Z )
    )
```

to the `EventCategoryLocation` or `EventCategoryTime` event, where:

- *Filename* = name, in quotations, of a .wav file located in the SOUND folder of the route. (If the .wav file is located elsewhere in the computer, the string must contain also the path from the `SOUND` folder to the location where the sound is located.)
- *Soundtype* = any one of the strings:
  - **`Everywhere` – sound is played in all views at the same volume** without fading effects
  - `Cab` – sound is played only in the cab
  - `Pass` – sound is played only in the active passenger view
  - **`Ground` – sound is played externally from a fixed position, the** one that the locomotive has reached when the event is triggered. The sound is also heard in internal views in an attenuated way, and becomes attenuated by moving away from the position.
  - **`Location` – sound is played externally from a fixed position** defined in the ORTSSoundLocation parameter.

Note: Parameter ORTSSoundLocation is needed only when *Soundtype* is `Location`.

For example:

```
EventCategoryLocation (
    EventTypeLocation ( )
    ID ( 7 )
    Activation_Level ( 1 )
    Outcomes (
        DisplayMessage ( "Won't be shown because ORTSContinue = 0")
            ORTSActivitySound (
                            ORTSActSoundFile ( "milanogrecopirelli.wav" "Ground" )
                    )
    )
    Name ( Location6 )
    Location ( -146 14082 -1016.56 762.16 10 )
    TriggerOnStop ( 0 )
    ORTSContinue ( 0 )
)
```

Including the `ORTSContinue` line (explained above) inhibits the normal halting of the activity by the event. Also, if the value of 0 is inserted in the line as in the example above, the display of the event message is completely suppressed. Only one sound file per event is allowed.

This feature is not yet managed by TSRE5 in this format.

10.16.9 Weather Change Activity Event

An activity can be modified so that the weather changes when running the activity in ORTS. MSTS operation is not affected by these WeatherChange events. The following block can be added within the Outcomes () block of an Event Block (either a Location or a Time event) of the .act file:

```
ORTSWeatherChange (
    ORTSOvercast (
        final_overcastFactor(float)
        overcast_transitionTime(int)
    )
    ORTSFog ( final_fogDistance(float) fog_transitionTime(int) )
    ORTSPrecipitationIntensity (
        final_precipitationIntensity(float)
        precipitationIntensity_transitionTime(int)
    )
    ORTSPrecipitationLiquidity (
        final_precipitationLiquidity(float)
        precipitationLiquidity_transitionTime(int)
    )
)
```

The weather will change accordingly during the activity. The ranges of the factors are as follows:

- *final_overcastFactor*: value from 0 to 1.
- *final_fogDistance*: value from 10 (meters) to 100000.
- *final_precipitationIntensity*: value from 0 to 0.020 (clamped to 0.010 if a 16 bit graphics card is used).
- *final_precipitationLiquidity*: value from 0 to 1.

The weather type will change accordingly to the following rules:

- when *precipitationIntensity* falls to 0, the weather type is set to Clear.
- when *precipitationIntensity* rises above 0 the weather type is selected accordingly to *final_precipitationLiquidity*.
- when *precipitationLiquidity* is above 0.3 the weather type is set to Rain.
- when *precipitationLiquidity* is below or equal to 0.3, weather type is set to Snow.

The parameter `ORTSPrecipitationLiquidity` allows for a smooth transition from rain (ORTSPrecipitationLiquidity = 1) to snow (ORTSPrecipitationLiquidity = 0) and vice-versa.

The xx_transitionTime is expressed in seconds, and indicates the time needed to pass from the initial weather feature value (overcastFactor, fogDistance and so on) to the final weather feature value. If such xx_transitionTime is set to 0, the weather feature takes immediately the final value. This is useful to start activities with weather features in intermediate states.

The event can also include an ORTSContinue (0) line, therefore not displaying messages and not suspending activity execution.

Manual commands related to weather interrupt the weather change triggered by the above events.

Each Event Block in the activity file may include only one WeatherChange block, and every WeatherChange block may include one to all of the lines specified above.

Event blocks including WeatherChange blocks may be partly interlaced (execution of one block can be still active at the moment a new WeatherChange block is triggered). Execution of the various weather parameter changes remains independent. If one weather parameter is present in both events, the execution of the parameter change commanded by the first block is stopped and the one commanded by the second block is started.

Note: editing the .act file with the MSTS Activity Editor after inclusion of WeatherChange events will remove them, so they should be backed up separately. Opening an .act file that contains WeatherChange events with the MSTS Activity Editor and packaging it without editing it generates an .apk file that contains the WeatherChange events.

This feature is not managed by TSRE5 in this format.

10.16.10 AI train Waiting Point modification through event

Purpose of the feature

An event outcome is available which modifies the waiting point expiration time when the event is hit (e.g. when the player train reaches it, in case of a location event).

This solves AI train sync problems. If e.g. an AI train is due to couple or uncouple cars to/from the player train, it must be ensured that the two trains are at the right place at the right time. If however this occurs after a long run of the player train, this one could be delayed, and so it is difficult to guarantee that the rendez-vous occurs correctly. In this case a long lasting waiting point may be placed on the AI train path. The AI train will wait there for the player train. At the sync location (usuall few before the point where the player train must be touched by the AI train) a location event is positioned, which indicates the updated waiting point value for the AI train (usually a short waiting point). When the player train will hit such location event, the AI train wating point will be updated and such train will restart after the updated waiting point has expired, and it will couple to the player train.

The feature may be used also for other features, like having an AI train coupling to the player train as helper, or like guaranteeing a passenger train connection in a station, or like having an AI train coupling to another AI train (as the event may also be triggered by an AI train, see *Location Event triggered by AI Train*.

Syntax of the feature

To make use of this feature it is suggested to generate an *Extension activity file* .

Here is an example of an extension activity file using such feature:

```
SIMISA@@@@@@@@@@JINX0a0t______

Tr_Activity (
      Tr_Activity_File (
              Events (
                      EventCategoryLocation (
                              ID ( 1 )
                              ORTSContinue ( 3 )
                              Outcomes (
                                      ORTSRestartWaitingTrain (
                                              ORTSWaitingTrainToRestart ( "TesteventWP_ai_
↪longerpath" 23240 )
                                                      ORTSDelayToRestart ( 60 )
                                                      ORTSMatchingWPDelay ( 31500 )
                                                      )
                              )
                      )
              )
      )
)
```

Description of parameters:

1) ORTSWaitingTrainToRestart has as first parameter the service name of the AI train whose waiting point has to be modified, and as second (optional) parameter the starting time of the AI train.
2) ORTSDelayToRestart is the new delay for the waiting point. It is expressed in seconds.
3) ORTSMatchingWPDelay indicates the original value of the AI train waiting point; this is used to ensure that the correct waiting point is modified.

The above file is also available as file TesteventWP_longerpath_extension.zip, which may be found within the `Documentation\SampleFiles\Manual` subfolder within the OpenRails folder. A sample activity using such file is available as file testeventwp_longerpath.zip in the same subfolder. It is an .apk file.

The activity uses the MSTS legacy route USA1 and legacy trainsets.

The player train exits from the tunnel and stops at the Baltimore station. Just before this, it hits the location event setting the AI train WP. Later an AI train will enter the station and stop. This train hits an absolute WP just after terminating passenger unloading. As the player train arrived before, such absolute WP becomes zeroed and the AI train will restart without further waiting.

If instead the player train is stopped before entering the station, and stays there until the AI train has entered the station and unloaded passsengers, the AI train will stay further there until the player train restarts, hits the location event and the modified WP time has expired.

This feature is not yet managed by TSRE5.

10.16.11 Old formats

Following alternate formats are accepted by OR for Event Sound Files and Weather Change. These formats are not recommended for new activities.

Event Sound Files: The sound file may be defined by a single line:

```
ORTSActSoundFile ( Filename SoundType )
```

to be inserted directly in the EventCategoryLocation () or EventCategoryTime () block, instead of being inserted within the Outcomes() subblock. In this alternate format the `Location` SoundType is not supported.

TSRE5 manages this format.

Weather Change events: the ORTSWeatherChange () block may be inserted directly in the EventCategoryLocation () or EventCategoryTime () block, instead of being inserted within the Outcomes() subblock.

TSRE5 manages this format.

CHAPTER 11

Timetable Mode

11.1 Introduction

The timetable concept is not a replacement for the activity definition, but is an alternative way of defining both player and computer-controlled (AI and Static) trains.

In an activity, the player train is defined explicitly, and all AI trains are defined in a traffic definition. Static trains are defined separately.

In a timetable all trains are defined in a similar way. On starting a timetable run, the required player train is selected from the list of available trains. In the timetable definition itself, no distinction is made between running trains – any of the running trains can be selected as player train, and if not selected as such they will be run as AI trains. Static trains are also defined in the same way but cannot be selected as the player train.

As a result, the number of different 'activities' that can be played using the same timetable file is equal to the number of trains which are defined in the timetable, less static trains.

Important aspects where the use of specific OR or MSTS items for timetables differs significantly from its use in an activity are shown in bold.

The development of the timetable concept is still very much a work in progress. As work continues, all items are still subject to change.

11.2 General

11.2.1 Data definition

The timetable data is defined in a Spreadsheet, and saved as a *.csv file (character separated file) in Unicode format. As the separation character, either ',' (comma), ';' (semi-colon) or the tab character must be used.

Do not select space as the separation character.

As ';', ',' or tab are possible separation characters, these symbols must not be used anywhere within the actual data. Enclosure of text by quotes (either single or double) has no effect. Also, the character '#' should not be used in train names, since it is the prefix for reserved words in the Timetable.

11.2.2 File structure

The saved `*.csv` files must be renamed with the extension `*.timetable-or`. The timetable files must be placed in a subdirectory named `OpenRails` created in the route's `Activities` directory.

11.2.3 Timetable groups

Multiple timetables can be loaded simultaneously using timetable group files. A group file is a plain text file that has the extension `.*.timetablelist-or`, that is also located in the `OpenRails` subdirectory of the route's `Activities` directory, and that contains the filenames of one or more timetable files listed on each line. The first line may also start with a `#` symbol, in which case the text that follows will be used as the timetable group's display name in the Open Rails menu.

Here is an example of a timetable group file:

```
#All Northeast Corridor Services - Fri Aug 2018
Amtrak - Fri Aug 2018.timetable-or
MARC Camden Line - Fri Aug 2018.timetable-or
MARC Penn Line - Fri Aug 2018.timetable-or
SEPTA Wilmington-Newark - Fri Aug 2018.timetable-or
```

11.2.4 Pool files

Pools can be used to store out-of-service trains on a first-come, first-serve basis, without the need to manually program paths into and out of storage tracks. Pool files are located in the same `OpenRails` directory as other timetable files. They have the extension `.pool-or` or `.turntable-or`.

11.2.5 Weather files

Weather files, a feature exclusive to timetable mode, orchestrate changes in the cloud cover, precipitation, and visibility factors over the course of the timetable day. They are located in a special `WeatherFiles` subdirectory of the route's folder and they have the `*.weather-or` file extension. The player activates a weather file by selecting it from the timetable mode section of the main menu; this overrides the static weather condition.

11.2.6 File and train selection

When starting a timetable run, the mode *Timetable* is selected in the menu. The desired timetable file or timetable group file must then be selected in the *Timetable set* display.

After selecting the required timetable, a list of all trains contained in that timetable is displayed and the required train can be selected.

Season and weather (static or file-defined) can also be selected; these are not preset within the timetable definition.

11.3 Timetable Definition

11.3.1 General

A timetable consists of a list of trains, and, per train, the required timing of these trains. The timing can be limited to just the start time, or it can include intermediate times as well.

At present, intermediate timings are limited to 'platform' locations as created using the MSTS Route Editor.

Each column in the spreadsheet contains data for a train and each row represents a location. A cell at the intersection of a train and location contains the timing data for that particular train at that location.

Special rows and columns can be defined for general information or control commands.

The first row for each column contains the train definition.

The first column for each row contains the location definition.

The cell at the intersection of the first row and first column **must be empty**.

This paragraph only lists the main outline, a fuller detailed description will follow in the next paragraphs.

11.3.2 Column definitions

A column is defined by the contents of the first row.

Default, the first row defines the train name.

Special columns can be defined using the following syntax :

- `#comment`: column contains comment only and is ignored when reading the timetable.
- <blank>: column is extension of preceding column.

11.3.3 Row definitions

A row is defined by the contents of the first column.

Default, the first column defines the stop location.

Special columns can be defined using the following syntax :

- `#comment`: row contains comment only and is ignored when reading the timetable
- <blank>: row is extension of row above
- `#path`: defines train path
- `#consist`: defines train consist
- `#start`: defines time when train is started
- `#note`: defines general notes and starting control commands for this train
- `#dispose`: defines how train is handled after it has terminated
- `#speed`, `#speedmph`, or `#speedkph`: defines train speed behavior in meters per second, miles per hour, or kilometers per hour, respectively; only one kind of speed row can be used in a single timetable file
- `#restartdelay`: defines randomized delays for a train
- `#briefing`: row contains briefing text for each train and is ignored when reading the timetable

11.3.4 Timing details

Each cell which is at an intersection of a train column and a location row, can contain timing details for that train at that location. *Timing commands* can be set at locations where the train stops, but can also be set for locations where no timing is inserted as the train passes through that location without stopping.

11.4 Timetable Data Details

11.4.1 Timetable Description

Although `#comment` rows and columns are generally ignored, the contents of the cell at the intersection of the first `#comment` row and first `#comment` column is used as the timetable description. This appears as the timetable's name in the Open Rails menu and is used to reference trains from other timetables.

11.4.2 Train Details

The train name as defined in the first row must be unique for each train in a timetable file. This name is also used when referencing this train in a train command; see details below.

The sequence of trains is not important.

11.4.3 Location Details

At present, the possible locations are restricted to 'platforms' as defined in the MSTS Route Editor.

Each location must be set to the 'Station Name' as defined in the platform definitions.

The name used in the timetable must exactly match the name as used in the route definition (*.tdb file), otherwise the location cannot be found and therefore cannot be processed.

Also, each location name must be unique, as otherwise its position in the train path could be ambiguous.

The sequence of the locations is not important, as the order in which the stations are passed by a train is defined in that train's path. For the same reason, a train's path can be set to just run in between some of the locations, or be set to bypass certain stations.

11.4.4 Timing Details

Each cell at an intersection of train and location can contain the timing details of that train at that location.

Times are defined as HH:mm, and the 24-hour clock must be used.

If a single time is inserted it is taken as the departure time (except at the final location).

If both arrival and departure time are to be defined, these must be separated by '-'.

Additional *timing commands* can be included. Such commands can also be set for locations where the train does not stop and therefore has no timing details, but the train must pass through that location for the commands to be effective.

Although a location itself can be defined more than once in a timetable, it is not possible to define timing details for trains for a location more than once. If a train follows a route which takes it through the same location more than once, the train must be 'split' into separate train entries.

11.4.5 Special Columns

- `#comment` column.

  A column with the #comment definition in the first row is a comment column and is ignored when reading the timetable, except for the cell at the intersection of the first comment column and the first comment row.

- <Blank> column.

  A column with a blank (empty) cell in the first row is taken as a continuation of the preceding column. It can be used to insert control commands which apply to the details in the preceding column. This can be useful when timings are derived automatically through formulas in the spreadsheet as inserting commands in the timing cell itself would exclude the use of such formulas.

11.4.6 Special Rows

- `#comment` row.

  A row with the #comment definition in the first column is a comment row and is ignored when reading the timetable, except for the cell at the intersection of the first comment column and the first comment row.

- <Blank> row.

  A row with a blank (empty) cell in the first column is taken as a continuation of the preceding row.

- `#path` row.

  The #path row defines the path of that train. The path must be a *.pat file as defined by the MSTS Activity Editor or by Trackviewer, and must be located in the route's Path directory. This field is compulsory.

  The timetable uses the same paths as those defined for activities.

  However, waiting points must not be defined in paths for use in timetables as the processing of waiting points is not supported in the timetable concept. Waiting points within a timetable must be defined using the specific control commands.

  The `#path` statement can take a qualifier: `/binary`.

  Large timetables can require many paths, and loading those paths can take considerable time (several minutes). To reduce this loading time, the paths can be stored in a processed, binary format. This format is the same as used in the 'save' command. Note that the binary path information cannot be directly accessed by the user, either for reading or for writing. When `/binary` is set, the program will check if a binary path exists. If so, it will read that path. If not, it will read the 'normal' path, and will then store this as binary for future use. Binary paths are stored in a subdirectory named `OpenRails` which must be created in the Paths directory of the route.

  Note: If a path or the route is edited, then the binary data will be out of date. If so, it is deleted and re-created automatically when the user starts the route.

- `#consist` row

  The `#consist` row defines the consist used for that train. This field is compulsory.

  However, if the train is run as an AI train and it is 'formed' out of another train (see below), the consist information is ignored and the train uses the consist of the train out of which it was formed.

  For the player train, the consist is always used even if the train is formed out of another train. The consist definition must be a *.con file as defined by the MSTS Activity Editor or by the TSRE5 consist editor, and must be stored in the defined consist directory.

  Also a more complex syntax of the consist definition is possible, as described below.

  This allows a consist definition to be not just a single string directly referring to a file, but a combination of strings, with the possibility to use (part of) the consist in reverse.

The general syntax is:

```
consist [$reverse] [+ consists [$reverse] [+ ...] ]
```

Example: a loco-hauled train, using the same set of coaches, running in both directions. Two consists are defined: `c_loco` and `c_wagons`. The consist definitions which can now be used are:

`c_loco + c_wagons`, and for reverse:

`c_loco $reverse + c_wagons $reverse`

Please note that `$reverse` always applies only to the sub-consist with which it is defined, not for the complete combined consist.

If this train sometimes has some additional wagons, e.g. during rush hours, the consists can be defined as follows (with `c_add` the definition of the additional wagons):

`c_loco + c_wagons + c_add`, and for reverse:

`c_loco $reverse + c_add $reverse + c_wagons $reverse`

Clearly, this can save on the definition of the total required consists, and in particular saves the tedious task of having to define 'reverse' consists. When using multiple units, this is even more useful.

Suppose there are two sets of multiple units, running either as single trains or combined. Normally, six different consists would be required to cover all trains, but now only two will suffice : `set_a` and `set_b`. The various combinations are:

`set_a`, reverse `set_a $reverse`.

`set_b`, reverse `set_b $reverse`.

`set_a + set_b`, reverse `set_b $reverse + set_a $reverse`.

Consist strings which contain '+' or '$' can be used in timetables but must be enclosed by < >. For instance :

`<loco+wagon>+<$loco+wagon>$reverse`

- `#start` row

  The `#start` row defines the time at which the train is started. It must be defined as HH:mm, and the 24 hour clock must be used. This field is compulsory.

  Use of start time for AI trains :

  - When a train is formed out of another train and this other train is included to run in the timetable, the time defined in #start is only used to define when the train becomes active.

  Use of start time for player train :

  - The time as defined in `#start` is normally used as the start time of the timetable 'activity'.

  If a train is formed out of another train and this train is included in the timetable, then if this train is delayed and has not arrived before the defined start time, the starting of this train is also delayed until the train out of which it is formed has arrived. This applies to both AI and player train. This means that the start of the player activity can be delayed.

  The `#start` field also accepts a number of *start commands*.

  For details on starting and running of trains around midnight see the paragraph *below*.

- `#note` row

  The `#note` row can be used to define *note commands* which are not location related but apply to the full run of the train. It can also be used to set commands for trains which do not stop at or pass through any defined location. This row is optional.

- `#dispose` row

  The `#dispose` row defines what happens to an AI train when it has reached the end of its run, i.e. it has reached the end of the defined path. The information in the `#dispose` row can detail if the train is to be formed into another train, and, if so, how and where. For details see the *dispose commands* as described further down.

  This row is optional and if included, the use per train is also optional. If the row is not included or the field is not set for a particular train, the train is removed from the activity after it has terminated.

  The #dispose row presently does not affect the end of the run for the player train.

- `#speed` row

  This optional field defines maximum speed for trains, which may restrict the train to lower speed as would otherwise be allowed. Note that any value defined here will never be applied if it exceeds the maximum speed as set through speedposts or signals, or as set in the consist file.

  If specified, only one `#speed` (m/s), `#speedkph`, or `#speedmph` row can be present in a single timetable file.

  This row also accepts a number of *speed commands*.

- `#restartdelay` row

  Delays are applied when restarting a train from a stop, e.g. at a station or a signal. Default random delays are set for each train. The default values may optionally be overruled using *delay commands* in the `#restartdelay` field.

  The random delay is calculated as fixed part $+ Random($variable part$)$, where all values are in seconds.

- `#briefing` row

  The `#briefing` row is optional and contains text which describes the train operation for the user. This text appears in the Open Rails main window along with description of the route and the loco.

  The user can also see it in-game in the Briefing tab of the Help Window (F1).

  A similar entry in the `#comment` column provides text which describes the entire timetable.

  The timetable-or file does not allow the fields to contain line-breaks but if HTML breaks “<br>” are inserted into the `#briefing` field, these will be converted to line-breaks.

11.4.7 Control Commands

General

Control commands can be set to control train and signaling behaviour and actions.

Command Syntax

All commands have the same basic syntax. A command consists of:

- Syntax name : defines the control command.
- Syntax value : set the value related to the command. Not all commands take a value.
- Syntax qualifiers : adds additional information to the command. Not all commands have qualifiers. Some qualifiers may be optional but others may be compulsory, or compulsory only in combination with other qualifiers.
- Syntax qualifier values : a qualifier may require a value

Command syntax:

```
$name = value /qualifier=value
```

Multiple values may be set, separated by '+'. Note that any qualifiers always apply to all values.

Train Reference

Many commands require a reference to another train. This reference is the other train's name as defined in the first row.

If the target train is in a separate timetable of the same timetable group, the reference is in the form of `train name:timetable description`, where the description is the text at the intersection of the first `#comment` row and `#comment` column in the other timetable file.

Station Commands

Station commands apply to all stops for a given station row. They are inserted directly after the station name in the first column.

`$hold, $nohold and $forcehold`

> If `$hold` is set, it defines that the exit signal for that location must be held at danger up to 2 minutes before train departure.
>
> An exit signal is allocated to a platform if this signal is beyond the end platform marker (in the direction of travel), but is still within the same track node - so there must not be any points etc. between the platform marker and the signal.
>
> **By default, the signal will not be held.**
>
> `$forcehold` will set the first signal beyond the platform as the 'hold' signal, even if this signal is not allocated to the platform as exit signal. This can be useful at locations with complex layout where signals are not directly at the platform ends, but not holding the signals could lead to delay to other trains.

`$forcewait`

> Force the train to wait if the next signal is at danger even if this signal is not recognized as the exit signal for that platform.

`$nowaitsignal`

> Normally, if a train is stopped at a station and the next signal ahead is still at danger, the train will not depart. But, there are situations where this should be overruled.
>
> Some stations are 'free line' stations - that is, they are not controlled by signals (usually small halts, without any switches). The next signal probably is a 'normal' block signal and may be some distance from the station. In that situation, the train does not have to wait for that signal to clear in order to depart.
>
> Other situation are for freight trains, light engines and empty stock, which also usually do not wait for the signal to clear but draw up to the signal so as to take as little as time as possible to exit the station.

`$terminal`

> The `$terminal` command changes the calculation of the stop position, and makes the train stop at the terminating end of the platform. Whether the platform is really a terminating platform, and at which end it terminates, is determined by a check of the train's path.
>
> If the platform is in the first section of a train's path, or there are no junctions in the path leading up to the section which holds the platform, it is assumed the train starts at a terminal platform and the end of the train is placed close to the start of the platform.

If the platform is in the last section if the path or there are no junctions beyond the section which holds the platform, it is assumed the platform is at the end of the train's path and the train will run up to near the end of the platform in its direction of travel.

If neither condition is met, it is assumed it is not a terminal platform after all, and the normal stop position is calculated.

The `$terminal` option can be set for a station, or for individual trains. If set for a station it cannot be overruled by a train.

However, because of the logic as described above, if set for a station which has both terminal platforms as well as through platforms, trains with paths continuing through those platforms will have the normal stop positions.

`$closeupsignal`

Sets a reduced clearance on approach to signal to maximize use of available platform length.

`$extendplatformtosignal`

Sometimes the platform marker is placed some distance from the actual end of the platform where the signal is located, e.g. in case of switches along the platform. Normally this would cause trains to stop far from the end of the platform and then block the switches to the rear. This parameter will place the 'end of platform' position not at the position of the platform marker but just ahead of the signal position.

`$restrictplatformtosignal`

Sometimes the platform marker is placed beyond the exit signal for that platform. If the signal is at danger, the train will stop at the signal and if this is a long train, this stop will not be seen as the station stop as the train has not reached the required platform stop position. This parameter will place the 'end of platform' position not at the position of the platform marker but just ahead of the signal position.

`$stoptime`

Syntax : `$stoptime=n` (n is time in seconds)

Sets the required default stop time at this platform, overriding the stoptime definition set in the track database.

`$closeup`

The train will stop close to another train already in the platform. Can only be used if the `$callon` timing command is also set for that train.

`$keepclear`

Defines that the stop position must be such that the length of platform as indicated in the command must be kept clear ahead of or behind the train. This may be essential if another train is to be attached or if another train is to be taken into the same platform.

Parameters :

`rear = <n>` (n in meter)

The stop location must be such that he minimal distance behind the train is n meter. If the platform has an exit signal, the train will stop in front of the signal even if this means that less than n meter is clear, unless the `/force` parameter is set as well. In this situation, the path of the train must continue beyond the exit signal.

Note that the train will never proceed beyond the end of its path.

`front = <n>` (n in meter)

The stop location must be such that the minimal platform length available ahead of the train is not less than n meter. If the rear of the train would be outside the platform, the location is calculated such that the rear of the

train is at the platform end even if this means that less than n meter is clear, except when the `/force` parameter is set as well.

`force`

Forces front or rear section to be kept clear even if train must pass exit signal (for `rear` parameter), or rear of train does not fit into platform (for `front` parameter).

`$endstop`

When the path of the train continues beyond the station position (e.g. when setting `$keepclear /rear /force`), the stop is considered to be the end of the path even if the train has not reached the actual final position.

Timing Commands

These commands can be set for each timing cell, i.e. at each intersection of train column and location row, or in the `#note` row. The commands will apply at and from the location onward (if applicable).

For instance, a `$wait` command can be set for a station without a stop. The actual wait location can be that station itself, but it could also be a loop or junction somewhere beyond that station.

`$wait`

Syntax : `$wait=<train> /maxdelay=n /notstarted /atstart /owndelay=n`

Defines that a train is to wait for the referenced train to allow this train to proceed first. The referenced train can be routed in the same or the opposite direction as this train itself. A search is done for the first track section which is common to both trains, starting at the location where the `$wait` is defined, or at the start of the path if defined in the `#note` row.

If the start location is already common for both trains, then first a search is done for the first section which is not common to both trains, and the wait is applied to the next first common section beyond that.

If the wait is set, the section will not be cleared for this train until the referenced train has passed this section. This will force the train to wait. The referenced train must exist for the wait to be valid.

However, if /notstarted is set, the wait will also be set even if the referenced train has not yet been started. This can be used where the wait position is very close to the start position of the referenced train, and there is a risk that the train may clear the section before the referenced train is started.

Care should be taken when defining a $wait at a location where the train is to reverse. As the search is performed for the active subpath only, a $wait defined at a location where the train is to reverse will not be effective as the common section will be in the next subpath after the reversal. In such a situation, the train should be 'split' into two separate definitions, one up to the reversal location and another starting at that location.

Command value : referenced train, this is compulsory.

Command qualifiers :

`/maxdelay=n`: n is the maximum delay (in minutes) of the referenced train for which the wait is still valid.

This delay is compensated for any delay of the train which is to wait, e.g. if maxdelay is 5 minutes, the referenced train has a delay of 8 minutes but this train itself has a delay of 4 minutes, the compensated delay is 4 minutes and so the wait is still valid.

This parameter is optional, if not set a maxdelay of 0 minutes is set as default.

`/notstarted`: the wait will also be applied if the referenced train has not yet started.

`/atstart`: the wait is activated at the present position rather than the first non-common position.

May be used where a train in opposite direction is to terminate in the same location as this train is started and there may not be any possible passing locations between this starting position and the present position of the other train.

`/owndelay=n` (n is delay in minutes); the owndelay qualifier command makes the command valid only if the train in question is delayed by at least the total minutes as set for the owndelay qualifier.

This can be used to hold a late-running train such that is does not cause additional delays to other trains, in particular on single track sections.

`/trigger=HH:MM`

Experimental option: Restricts this command to trigger only after the specified time.

`/endtrigger=HH:MM`

Experimental option: Restricts this command to trigger only before the specified time.

`$follow`

Syntax : `$follow=<train> /maxdelay=n /notstarted /owndelay=n`

This command is very similar to the $wait command, but in this case it is applied to each common section of both trains beyond a part of the route which was not common. The train is controlled such that at each section where the paths of the trains re-join after a section which was not common, the train will only proceed if the referenced train has passed that position. The command therefore works as a $wait which is repeated for each such section.

The command can only be set for trains routed in the same direction. When a wait location is found and the train is due to be held, a special check is performed to ensure the rear of the train is not in the path of the referenced train or, if it is, the referenced train has already cleared that position. Otherwise, a deadlock would result, with the referenced train not being able to pass the train which is waiting for it.

Command value: referenced train, this is compulsory.

Command qualifiers:

`/maxdelay=n`: n is the maximum delay (in minutes) of the referenced train for which the wait is still valid. This delay is compensated by any delay of the train which is to wait, e.g. if maxdelay is 5 minutes, the referenced train has a delay of 8 minutes but this train itself has a delay of 4 minutes, the compensated delay is 4 minutes and thus the wait is still valid.

This parameter is optional, if not set a maxdelay of 0 minutes is set as default.

`/notstarted`: the follow will also be applied if the referenced train has not yet started.

`/owndelay=n` (n is delay in minutes): the owndelay qualifier command makes the command valid only if the train in question is delayed by at least the total minutes as set for the owndelay qualifier.

This can be used to hold a late-running train such that is does not cause additional delays to other trains, in particular on single track sections.

`/trigger=HH:MM`

Experimental option: Restricts this command to trigger only after the specified time.

`/endtrigger=HH:MM`

Experimental option: Restricts this command to trigger only before the specified time.

`$connect`

Syntax : `$connect=<train> /maxdelay=n /hold=h`

Defines that a train is to wait at a station until another train has arrived, so as to let passengers make the connection between the trains.

The train will be timetabled to allow this connection, and the `$connect` command is set to maintain this connection if the arriving train is running late.

Note that the `$connect` command will not lock the signal. If the paths of this train and the arriving train conflict before the arriving train reaches the station, additional `$wait` or `$hold` commands must be set to avoid deadlock.

Command value: reference to train which is to be waited for, this is compulsory.

Command qualifiers :

`/maxdelay=n` : n is the maximum delay (in minutes) of the arriving train for which this train is held.

If the delay of the arriving train exceeds this value the train will not wait. The maximum delay is independent from this train's own delay.

This qualifier and its value are compulsory.

`/hold=n` : n is the time (in minutes) the train is still held after the other train has arrived, and relates to the time required by the passengers to make the connection.

This qualifier and its value are compulsory.

`$waitany`

Syntax : `$waitany=<path> /both /opposite`

This command will set a wait for any train which is on the path section as defined.

If the qualifier /both is set, the wait will be applied for any train regardless of its direction, otherwise the wait is set only for trains heading in the same direction as the definition of the path.

The path defined in the waitany command must have a common section with the path of the train itself, otherwise no waiting position can be found.

This command can be set to control trains to wait beyond the normal signal or deadlock rules. For instance, it can be used to perform a check for a train which is to leave a siding or yard, checking the line the train is to join for any trains approaching on that line, for a distance further back than signalling would normally clear, so as to ensure it does not get into the path of any train approaching on that line.

With the /both qualifier set, it can be used at the terminating end of single track lines to ensure a train does not enter that section beyond the last passing loop if there is another train already in that section as this could lead to irrecoverable deadlocks.

With the /opposite qualifier set, the command searches only for trains in the opposite direction of the defined path.

`$callon`

This will allow a train to 'call on' into a platform occupied by another train.

For full details, see the *discussion above* on the relationship between signalling and timetable.

`$hold, $nohold and $forcehold`

These commands are functionally identical to (and take precedence over) their respective station *commands*, but apply only to the current train.

`$forcewait`

Identical to the station *command*, but applies only to the current train.

`$nowaitsignal`

Identical to the station *command*, but applies only to the current train.

`$waitsignal`

Can be used to override and negate a `$nowaitsignal` station *command* for the current train.

`$noclaim`

Experimental option: The `$noclaim` command inhibits the train from claiming track circuit sections if the train is held at a signal. A train with the $noclaim command would always be last in the queue at busy junctions, always giving priority to any other train.

`$detach`

Syntax : `$detach <detach parameters> <forms parameters>`

Set details for train to detach a portion of that train.

Parameters to define the portion to be detached :

`/power`

Will detach the power unit. The system will check for power unit at front or rear, if both are found, front will prevail. If there is no power unit at either end, nothing is detached.

`/leadingpower`

Will detach the front power unit only. If there is no power unit at the front, nothing is detached.

`/allleadingpower`

Will detach all power units at the front of the train. If there are no power units at the front, nothing is detached.

`/trailingpower`

Will detach the power unit which is the rearmost unit on the train. If the rear unit is not a power unit, nothing is detached.

`/alltrailingpower`

Will detach all power units from the rear of the train. If there are no power units at the rear of the train, nothing is detached.

`/nonpower`

All units which are not power units will be detached from the train. The system will determine at which end of the train power units are located, and will then detach all non power units from the other end of the train.

If neither end has power units, units will be detached from the rear. If both ends are power units, nothing is detached.

`/units=n` (n may be <0 or >0 but n=0 is not allowed)

Number of units to be detached.

If n>0, the units will be detached at the front of the train. If n<0, the units will be detached at the rear of the train. If n exceeds the actual length of the train, n is reduced such that one unit remains on the train.

`/consist=<consist>[+<consist>[+...]]`

Name of consist(s) to be detached. For use of consist names in detach command, see *note on consist names* below.

The consist to be detached must be at either end of the train, i.e. it must be the front portion or the rear portion of the train.

If a list of consists is defined, it must be in the sequence of the consists to be detached, from the outside looking inward, i.e. if the units are to be detached at the front, the first consist in the list must be the front portion, but if the units are to be detached at the rear the first consist in the list must be the rear portion.

If neither front nor rear portion matches the consist or first consist as defined, nothing is detached.

Parameters for formed train :

`/forms=<train>`

Detached portion will form train as indicated.

`/static`

Detached portion will form a static consist.

`$attach`

Syntax : `$attach=<train>`

This train will attach to train as indicated, and will therefore cease to exist.

If used at station stop, there is no use to define anything beyond this stop, and nothing can be defined in the `#dispose` field either.

If the other train to which this train must attach is not at the location where the attach is to take place, this train will terminate without the attach taking place. It is therefore advisable to use a `$wait` command to ensure the other train is in the location as required.

If the `/firstin` or `/setback` parameter is set, it should be the other way round, in that case a `$wait` command should be set for the other train to ensure this train is indeed first in.

Parameters (only valid at station stop) :

`/firstin`

This train is in first, and will wait for arrival of the second train to perform the attach. The other train may come in ahead of this train through a switch or from the opposite direction.

`/setback`

This train is in first, and will wait for the other train to come in behind. When the other train has arrived, this train will set back to perform the attach.

This should not be used if an engine is to be detached from the other train as this train will not wait for the engine to clear before performing the attach.

`$pickup`

Syntax : `$pickup=<train> /static`

This train will pick up the train as defined in the command, or will pick up the static consist which is on the location where the pickup is defined.

The train which is picked up will cease to exist. The full train is picked up, no changes are made to the consists of either trains (except if combined with $triggers command in #dispose field).

If there is no train to pick up at the required location, the train will continue as defined.

`$transfer`

Syntax : `$transfer=<train> /static <transfer parameters>`

This train (the "active" train) will transfer units with the train as indicated, or with a static consist placed at the location where the transfer is defined (the "passive" train).

With a transfer, units will be transferred from one train to another, but both trains will continue to exist. At least one power unit must remain on the "active" train, this power unit must not be part of the portion to be transferred. The "passive" train need not have power units, or all power units may be detached as part of the transfer.

Parameters defining the type of transfer :

`/give`

This train is to give the defined units to the other train, that is units as defined for the "active" train will be moved to the "passive" train.

`/take`

This train is to take the defined units from the other train, that is units as defined for the "passive" train will be moved to the "active" train.

`/keep`

All units except the units as defined for the "active" train will be transferred to the "passive" train.

`/leave`

All units except the units as defined for the "passive" train will be transferred to the "active" train.

Parameters defining the units to transfer or to keep on the train :

`/onepower` : One power unit only.

`/allpower` : All power units.

`/nonpower` : All units which are not power units.

`/units=<n>`

If the portion is defined for the "active" train, and `<n>` exceeds the length of that train, the number is reduced such that one unit will remain on the train.

`/consist=<consist>[+consist[+...]]`

Consists names of portions to keep or to transfer. The consist names must be in sequence, and the first (or only) consist name must match the portion at the applicable end of the train.

`$activate`

Syntax : `$activate=<train>`

Will activate the train as indicated, either when the train starts, when the train is at the indicated stop or when it is terminated.

Start Commands

`$create`

Syntax : `$create[=<time>] [/ahead=<train>]`

The `$create` command will create that train at the time as indicated. If no time is set, the train will be created before the start of the first train. The train will be 'static' until the time as set as start time. The normal rules for train placement still apply, so a train cannot be placed onto a section of track already occupied by another train.

However, storage sidings often hold multiple trains. To allow for this, and to ensure the trains are stored in proper order (first one out up front), the parameter `[/ahead=<train>]` must be used.

The train will now be placed ahead of the referenced train, in the direction of the train's path. Multiple trains can be stored on a single siding, but care must be taken to set the proper references. The reference must always be to the previous train - two trains cannot reference the same train in the `/ahead` parameter as that would cause conflict.

If the total length of all trains exceeds the length of the sidings, the trains will 'spill out' onto whatever lies beyond.

Note that a train referenced in an `/ahead` parameter must be created before or at the same time as the train which uses that reference.

`$pool`

Syntax : `$pool=<poolname> [/direction=forward|backward]`

Train originates from the defined pool.

For trains starting from a pool, the path must start at or near the end of one of the access paths as defined for that pool. If the path starts earlier than the last track section defined for the access path, it must not deviate from that path.

For turntable pools, the direction in which the train exits from the turntable can be set using the direction qualifier. If not set, the train will reverse.

`$next`

Start time is after 00:00 at the end of the timetable. May be used to start train running after midnight.

`$static`

Syntax : `$static [/pool=<pool>] [/ahead=<train>]`

This train will spawn as a static train.

`/pool=<pool>`

Train is created in referenced pool. For a pool to have trains, these must be defined using this command.

The path must be a storage path as defined for that pool. Note that the train may be placed on one of the other storage paths as defined for that pool, this is defined through the pool logic.

If more trains are created in a pool than the pool can hold, a warning is issued.

`/ahead=<train>`

As above for the `$create` command.

`$activated`

The train is activated through the `$activate` command from another train. The `$activate` command may be sent before or after the defined start time of this train.

A train can be activated by only one other train.

Note Commands

The note row defines commands applicable to when the train is started. In addition to the exclusive `#note` commands listed below, this row also accepts all *timing commands*.

The program uses average acceleration and deceleration values for all trains (different values for freight, passenger and high speed trains). But these values are not always adequate, especially for modern trains. This can lead to delays when trying to run to a real life timetable.

Using the `$acc` and `$dec` commands, the values used can be modified. Note that these commands do not define an actual value, but define a factor; the default value will be multiplied by this factor. However, setting a higher value for acceleration and deceleration does not mean that the trains will always accelerate and decelerate faster according to the set value. **Most of the time, the train behaviour is controlled through the physics.** But especially the `$dec` factor does have an important side effect. The deceleration value is also used to calculate the expected required braking distance. Setting a higher deceleration will reduce the required braking distance, allowing the train to continue to run at maximum allowed speed for longer distances. This can have a significant effect on the timing. Take care, though, not to set the value too high - the calculated braking distance must of course be sufficient to allow for proper braking, otherwise the train cannot stop in time resulting in SPADs etc.

A typical value for modern stock for the `$dec` command is 2 or 3.

`$acc`

Syntax : `$acc=<value>`

Sets the required acceleration for this train. <value> is a multiplier for the default acceleration.

`$dec`

Syntax : `$dec=<value>`

Sets the required deceleration for this train. <value> is a multiplier for the default deceleration.

`$doo`

Defines the train as “Driver Only Operated”. If set, there will be no departure sound (whistle, bell or whatever) on departure from a station.

`$forcereversal`

Normally, when a reversal is made and there is a signal in the train’s path as leading from the reversal point, the actual reversal position is placed such that the train will be fully passed that signal before reversing, and the reverse move is therefor controlled by that signal.

Setting `$forcereversal` will allow the train to reverse as soon at it is clear of the reverse position. This is useful when shunting in yards when there is no need to fully exit the yard to reverse and the entry signal.

Speed Commands

`$max`

Syntax : `$max=<value>`

Overall maximum speed for this train.

`$cruise`

Syntax : `$cruise=<value>`

Maximum speed at which train will normally operate when it is running on time.

When the actual delay exceeds the defined maximum delay (as set in `$maxdelay`), the train will accelerate to maximum speed.

`$maxdelay`

Syntax : `$maxdelay=<m>`

Maximum delay (in minutes) for cruise control. When this delay is exceeded, the train will accelerate to maximum speed.

`$creep`

Syntax : `$creep=<value>`

Creep speed is the minimum speed on the final approach to a signal at danger or station stop location.

`$attach`

Syntax : `$attach=<value>`

Speed at which the train will attach to another train.

`$detach`

Syntax : `$detach=<value>`

Speed at which the train will detach from another train.

`$movingtable`

Syntax : `$movingtable=<value>`

Speed at which the train will navigate turntables.

Delay Commands

All delay commands, except for the `$reverse` *command*, are in the form of `$command [/fix=<f>] [/var=<v>]`, where `<f>` represents the fixed component of the time delay and `<v>` represents the variable component of the time delay, both in seconds.

`$new`

Set the train's delay after spawning into the simulator.

The fixed delay defaults to 0 seconds, while the variable delay defaults to 10 seconds.

`$path`

Set the train's delay after stopping for an obstacle along its path, such as a stop signal or a reversed switch.

The fixed delay defaults to 1 second, while the variable delay defaults to 10 seconds.

`$station`

Set the train's delay after making a station stop.

The fixed delay defaults to 0 seconds, while the variable delay defaults to 15 seconds.

`$follow`

Set the train's delay when following another train.

The fixed delay defaults to 15 seconds, while the variable delay defaults to 10 seconds.

`$attach`

Set the train's delay after attaching to another train.

The fixed delay defaults to 30 seconds, while the variable delay defaults to 30 seconds.

`$detach`

Set the train's delay after detaching one of its portions.

The fixed delay defaults to 5 seconds, while the variable delay defaults to 20 seconds.

`$movingtable`

Set the train's delay after using a turntable.

The fixed delay defaults to 1 second, while the variable delay defaults to 10 seconds.

`$reverse`

Syntax : `$reverse /additional=<value>`

When reversing, an additional delay is added to reflect the time required for the driver to walk through or along the train to the other end. This delay defaults to 0.5 seconds per meter of train, a value that can be overridden with this command.

For trains which are pushed on reversal, e.g. for shunt moves of freight trains, it is advisable to set the reversing delay to 0.

Dispose Commands

Dispose commands can be set in the `#dispose` row to define what is to be done with the train after it has terminated. See special notes below on the behaviour of the player train when it is formed out of another train by a dispose command, or when the player train itself has a dispose command.

`$forms`

Syntax : `$forms=<train> <qualifiers>`

`$forms` defines which new train is to be formed out of this train when the train terminates. The consist of the new train is formed out of the consist of the terminating train and any consist definition for the new train is ignored. The new train will be 'static' until the time as defined in #start row for that train. This means that the new train will not try to clear its path, signals etc., and will not move even if it is not in a station.

If the incoming train is running late, and its arrival time is later as the start time of the new train, the start of the new train is also delayed but the new train will immediately become active as soon as it is formed.

For locomotive-hauled trains, it can be defined that the engine(s) must run round the train in order for the train to move in the opposite direction. The runround qualifier needs a path which defines the path the engine(s) is to take when performing the runround. If the train has more than one leading engine, all engines will be run round. Any other power units within the train will not be moved.

For specific rules and conditions for runround to work, see *discussion* on the relationship between signalling and the timetable concept.

If runround is defined, the time at which the runround is to take place can be defined. If this time is not set, the runround will take place immediately on termination of the incoming train.

Command value : referenced train, this is compulsory.

Command qualifiers:

`/runround=<path>`: `<path>` is the path to be used by the engine to perform the runround.

This qualifier is optional; if set, the value is compulsory.

For finer control over the runround maneuver, it is suggested to use the `$detach` and `$attach` commands instead.

`/rrtime=time`: time is the definition of the time at which the runround is to take place. The time must be defined in HH:mm and must use the 24 hour clock.

This qualifier is only valid in combination with the /runround qualifier, is optional but if set, the value is compulsory.

/setstop: if this train itself has no station stops defined but the train it is to form starts at a station, this command will copy the details of the first station stop of the formed train, to ensure this train will stop at the correct location.

For this qualifier to work correctly, the path of the incoming train must terminate in the platform area of the departing train.

This qualifier is optional and takes no values.

/atstation: The final position of the train is calculated as if the train is stopping at the station where the new train starts, even if no station stop is defined for this train.

/closeup: Final position of train will be close up to end of track or other train.

/speed=<v>: This qualifier can only be used with the $runround parameter. It defines the maximum speed for the runround move in m/s.

$triggers

Syntax : $triggers=<train> <qualifiers>

$triggers also defines which new train is to be formed out of this train when the train terminates.

However, when this command is used, the new train will be formed using the consist definition of the new train and the existing consist is removed.

Command value : referenced train, this is compulsory.

Command qualifiers:

/runround=<path>: <path> is the path to be used by the engine to perform the runround.

This qualifier is optional; if set, the value is compulsory.

/rrtime=time: time is the definition of the time at which the runround is to take place. The time must be defined in HH:mm and must use the 24 hour clock.

This qualifier is only valid in combination with the /runround qualifier, is optional but if set, the value is compulsory.

/setstop: if this train itself has no station stops defined but the train it is to form starts at a station, this command will copy the details of the first station stop of the formed train, to ensure this train will stop at the correct location.

For this qualifier to work correctly, the path of the incoming train must terminate in the platform area of the departing train.

This qualifier is optional and takes no values.

/atstation: The final position of the train is calculated as if the train is stopping at the station where the new train starts, even if no station stop is defined for this train.

/closeup: Final position of train will be close up to end of track or other train.

/speed=<v>: This qualifier can only be used with the $runround parameter. It defines the maximum speed for the runround move in m/s.

$static

Syntax : $static /closeup

The train will become a 'static' train after it has terminated.

Command value : none.

Command qualifiers:

/closeup: Final position of train will be close up to end of track or other train.

`$stable`

Syntax: `$stable /out_path=<path> /out_time=time /in_path=<path> /in_time=time /static /runround=<path> /rrtime= time /rrpos=<runround position> /forms=<train> /triggers=<train> /speed=<v> /name=<name>`

`$stable` is an extended form of either $forms, $triggers or $static, where the train is moved to another location before the related command is performed. In case of /forms or /triggers, the train can move back to the same or to another location where the new train actually starts. Note that in these cases, the train has to make two moves, outward and inward.

A runround can be performed in case /forms is defined.

If `/triggers` is defined, the change of consist will take place at the 'stable' position. Any reversal(s) in the inward path, or at the final inward position, are taken into account when the new train is build, such that the consist is facing the correct direction when the new train is formed at the final inward position.

The `$stable` can be used where a train forms another train but when the train must clear the platform before the new train can be formed to allow other trains to use that platform. It can also be used to move a train to a siding after completing its last duty, and be 'stabled' there as static train.

Separate timings can be defined for each move; if such a time is not defined, the move will take place immediately when the previous move is completed.

If timings are defined, the train will be 'static' after completion of the previous move until that required time.

If the formed train has a valid station stop and the return path of the stable command (in_path) terminates in the area of the platform of the first station stop of the formed train, the 'setstop' check (see setstop qualifier in `$forms` command) will automatically be added

Command value : none.

Command qualifiers :

`/out_path=<path>`: <path> is the path to be used by the train to move out to the 'stable' position. The start of the path must match the end of the path of the incoming train.

`/out_time = time`: time definition when the outward run must be started. Time is defined as HH:mm and must use the 24 hour clock.

`/in_path=<path>`: <path> is the path to be used by the train for the inward run from the 'stable' position to the start of the new train. The start of the path must match the end of the out_path, the end of the path must match the start of the path for the new train.

`/in_time = time`: time definition when the inward run must be started. Time is defined as HH:mm and must use the 24 hour clock.

`/closeup`: Final position of train will be close up to end of track or other train.

`/callon`: This train is allowed to proceed into the platform even if that platform is occupied.

> This option requires the TrainHasCallOn or TrainHasCallOn_Restricted function to be implemented for the signal which protects the platform.

`/runround=<path>`: <path> is the path to be used by the engine to perform the runround. For details, see the $forms command definition of the time at which the runround is to take place. The time must be defined in HH:mm and must use the 24 hour clock.

`/rrtime=time`: time is the definition of the time at which the runaround is to take place. The time must be defined in HH:mm and must use the 24 hour clock.

`/rrpos = <runround position>`: the position within the 'stable' move at which the runround is to take place.

Possible values:

- out: the runround will take place before the outward move is started.
- stable: the runround will take place at the 'stable' position.
- in: the runround will take place after completion of the inward move.

`/speed=<v>`: This qualifier can only be used with the `$runround` parameter. It defines the maximum speed for the runround move in m/s.

`/name=<name>`: This qualifier can only be used with the `$runround` parameter. It defines the name the train will carry during the stable move. This is the name shown in F7 info, in the dispatcher hud info and in the dispatcher window.

`/static`: train will become a 'static' train after completing the outward move.

`/forms=<train>`: train will form the new train after completion of the inward move. See the $forms command for details.

`/triggers=<train>`: train will trigger the new train after completion of the inward move. The train will change to the consist of the new train at the 'stable' position. See the $triggers command for details.

Use of command qualifiers :

In combination with /static:

- /out_path: compulsory
- /out_time: optional

In combination with /forms:

- /out_path: compulsory
- /out_time: optional
- /in_path: compulsory
- /in_time: optional
- /runround: optional
- /rrtime: optional, only valid if /runround is set
- /rrpos: compulsory if /runround is set, otherwise not valid

In combination with /triggers :

- /out_path: compulsory
- /out_time: optional
- /in_path: compulsory
- /in_time: optional

`$pool`

Syntax : `$pool=<poolname> [/direction=forward|backward]`

Train enters the defined pool when it terminates.

For turntable pools, the direction in which the train enters from the turntable can be set using the direction qualifier. If not set, the train will reverse.

`$attach`

Equivalent to the *timing command* of the same name.

`$detach`

Equivalent to the *timing command* of the same name.

`$pickup`

Equivalent to the *timing command* of the same name.

`$transfer`

Equivalent to the *timing command* of the same name.

`$activate`

Equivalent to the *timing command* of the same name.

11.5 Additional Notes on Timetables

11.5.1 Static Trains

A static train can be defined by setting `$static` in the top row (e.g. as the 'name' of that train). Consist and path are still required - the path is used to determine where the consist is placed (rear end of train at start of path). No start-time is required. The train will be created from the start of the timetable - but it cannot be used for anything within a timetable. It cannot be referenced in any command etc., as it has no name. At present, it is also not possible to couple to a static train - see below for details.

Note that there are some differences between timetable and activity mode in the way that static trains are generated. In activity mode, the train is an instance of the Train class, with type STATIC.

In timetable mode, the train is an instance of the TTTrain class (as are all trains in timetable mode), with type AI, movement AI_STATIC. This difference may lead to different behaviour with respect to sound, smoke and lights.

11.5.2 Processing of #dispose Command For Player Train

When the player train terminates and a #dispose command is set for that train to form another train (either $form, $trigger or $stable), the train will indeed form the next train as detailed, and that next train will now be the new player train. So the player can continue with that train, for instance on a return journey.

On forming the new train, the train will become 'Inactive'. This is a new state, in which the train is not authorized to move.

Note that the *F4 Track Monitor* information is not updated when the train is 'Inactive'. The *Next Station* display in the *F10 Activity Monitor* will show details on when the train is due to start. The train will become 'active' at the start-time as defined for the formed train. For information, the Activity Monitor window shows the name of the train which the player is running.

11.5.3 Termination of a Timetable Run

On reaching the end of a timetable run, the program will not be terminated automatically but has to be terminated by the player.

11.5.4 Calculation of Running Delay

An approximate value of the delay is continuously updated. This approximation is derived from the booked arrival time at the next station. If the present time is later as the booked arrival, and that difference exceeds the present delay, the delay is set to that difference. The time required to reach that station is not taken into account.

This approximation will result in better regulation where /maxdelay or /owndelay parameters are used.

11.5.5 No Automatic Coupling

There is logic within the program which for any stopped train checks if it is close enough to another train to couple to this train. It is this logic which allows the player train to couple to any static train.

However, this logic contains some actions which do not match the processing of timetable trains. Therefore, coupling of trains is not possible in timetable mode except for maneuvers specified explicitly with commands, such as `$attach` and `$detach`.

11.5.6 Use of Consists in Shunting Commands

Any wagon on the simulation must have been placed somewhere as a 'new' train. When a 'new' train is placed, it is formed as defined in the consist definition for that train.

Each wagon will remember this 'original consist' throughout its entire life on the simulation.

This 'original consist' name can be used in any $detach or $transfer command, even if the portion involved has changed trains.

So, for instance, if a freight train is placed which consists of multiple portions, each with their own consist name (using the multiple consist definition), each wagon in that train will always remember its original consist. When this train is taken apart, portions are taken into other trains etc., the original consist name can still be used.

When using this facility it is important to keep track of where and in which train the various portions are moved. As a list of consists must be defined in the correct sequence, it is also important to keep track of the configuration of the formed trains. The advantage of this method is that one does not need to keep count of the number of units in each train and each portion.

Note that the consist information can not be used if the unit is started at a pool, if that pool can hold different consists. In that situation, it is not defined which consist will form the actual train.

11.5.7 Signalling Requirements and Timetable Concept

General

The timetable concept is more demanding of the performance of the signalling system than 'normal' activities. The main reason for this is that the timetable will often have AI trains running in both directions, including trains running ahead of the player train in the same direction as the player train. There are very few activities with such situations as no effort would of course be made to define trains in an activity which would never be seen, but also because MSTS could not always properly handle such a situation.

Any flaws in signalling, e.g. signals clearing the path of a train too far ahead, will immediately have an effect on the running of a timetable.

If signals clear too far ahead on a single track line, for instance, it means trains will clear through passing loops too early, which leads to very long waits for trains in the opposite direction. This, in turn, can lead to lock-ups as multiple trains start to converge on a single set of passing loops.

Similar situations can occur at large, busy stations - if trains clear their path through such a station too early, it will lead to other trains being kept waiting to enter or exit the station.

If `$forms` or `$triggers` commands are used to link reversing trains, the problem is exacerbated as any delays for the incoming train will work through on the return working.

Call On Signal Aspect

Signalling systems may allow a train to 'call on', i.e. allow a train onto a section of track already occupied by another train (also known as permissive working).

The difference between 'call on' and 'permissive signals' (STOP and PROCEED aspects) is that the latter is also allowed if the train in the section is moving (in the same direction), but 'call on' generally is only allowed if the train in the section is at a standstill.

When a signal allows 'call on', AI trains will always pass this signal and run up to a pre-defined distance behind the train in the section.

In station areas, this can lead to real chaos as trains may run into platforms occupied by other trains such that the total length of both trains far exceeds the platform length, so the second train will block the 'station throat' stopping all other trains. This can easily lead to a complete lock-up of all traffic in and around the station.

To prevent this, calling on should be blocked in station areas even if the signalling would allow it. To allow a train to 'call on' when this is required in the timetable, the `$callon` command must be set which overrules the overall block. This applies to both AI and player train

In case the train is to attach to another train in the platform, calling on is automatically set.

Because of the inability of AI trains in MSTS to stop properly behind another train if 'called on' onto an occupied track, most signalling systems do not support 'call on' aspects but instead rely on the use of 'permission requests'. AI trains cannot issue such a request, therefore in such systems `$callon` will not work.

In this situation, attach commands can also not work in station areas.

Note that the 'runround' command also requires 'call on' ability for the final move of the engine back to the train to attach to it. Therefore, when performed in station areas, also the runround can only work if the signalling supports 'call on'.

Special signalling functions are available to adapt signals to function as described above, which can be used in the scripts for relevant signals in the sigscr file.

The function "TRAINHASCALLON()" will return 'true' if the section beyond the signal up to the next signal includes a platform where the train is booked to stop, and the train has the 'callon' flag set. This function will also return 'true' if there is no platform in the section beyond the signal.

The function "TRAINHASCALLON_RESTRICTED" returns 'true' in similar conditions, except that it always returns 'false' if there is no platform in the section beyond the signal.

Both functions must be used in combination with BLOCK_STATE = BLOCK_OCCUPIED.

Wait Commands and Passing Paths

From the location where the 'wait' or 'follow' is defined, a search is made for the first common section for both trains, following on from a section where the paths are not common.

However, on single track routes with passing loops where 'passing paths' are defined for both trains, the main path of the trains will run over the same tracks in the passing loops and therefore no not-common sections will be found. As a result, the waiting point cannot find a location for the train to wait and therefore the procedure will not work.

If waiting points are used on single track lines, the trains must have their paths running over different tracks through the passing loop in order for the waiting points to work properly.

It is a matter of choice by the timetable creator to either pre-set passing locations using the wait commands, or let the system work out the passing locations using the passing paths.

Wait Commands and Permissive Signals

The 'wait' and 'follow' commands are processed through the 'blockstate' of the signal control. If at the location where the train is to wait permissive signals are used, and these signals allow a 'proceed' aspect on blockstate JN_OBSTRUCTED, the 'wait' or 'follow' command will not work as the train will not be stopped.

Running Trains Around Midnight

A timetable can be defined for a full 24 hour day, and so would include trains running around midnight.

The following rules apply for the player train:

- Train booked to start before midnight will be started at the end of the day, but will continue to run if terminating after midnight.
- Trains formed out of other trains starting before midnight will NOT be started if the incoming train is delayed and as a result the start time is moved after midnight. In this situation, the activity is aborted.
- Trains booked to start after midnight will instead be started at the beginning of the day, unless the `$next` *command* is used.

The following rules apply for AI trains:

- Trains booked to start before midnight will be started at the end of the day, but will continue to run if terminating after midnight.
- Trains formed out of other trains starting before midnight will still be started if the incoming train is delayed and as a result the start time is moved after midnight.
- Trains booked to start after midnight will instead be started at the beginning of the day, unless the `$next` *command* is used.

Viewing the Other Active Trains in the Timetable

To change the train that is shown in the external views, click `<Alt+F9>` to display the *Train List* and select the desired train from the list of active trains, or click `<Alt+9>` as described in *Changing the View* to cycle through the active trains.

11.5.8 Known Problems

- If a #dispose command is processed for the player train , and the new train runs in the opposite direction, the reverser will 'jump' to the reverse state on forming that new train.
- A run-round command defined in a #dispose command cannot yet be processed. It will be necessary to switch to Manual to perform that run-round.
- If two trains are to be placed on a single siding using $create with /ahead qualifier, but the trains have paths in opposite directions, the trains may be placed in incorrect positions.
- If the /binary qualifier is set for #path, but the OpenRails subdirectory in the Paths directory does not exist, the program will not be able to load any paths.

11.6 Storing Trains with Pools

Pools can be used to store trains before or in between active duties, or when all duties have been performed. Trains can be defined to be placed in a pool at the start of the timetable. When required, the train can be extracted from the pool. When the duty has terminated, the train can be returned to the pool. There is no need to define the exact storage of the train, nor is there need to sort out the various duties so as to avoid trains being locked in by other trains which are only required at a later time. When using pools, the system will take care of actual storage location and will select the first available train when a train is required.

A pool will consist of one or more tracks which are used to stable the trains. Access tracks must also be defined. (For details, see below.) A special type of pool is the turntable pool. In a turntable pool, all storage tracks are connected to a turntable. The access paths are also connected to the turntable. When extracting or storing a train, the train will run unto the turntable and the turntable, with the train on it, will be turned to the required position.

Pools can be used for both AI and player trains. When a train which is extracted from a pool is selected as the player train, the first available train will be selected and set as player train. When a train which is the player train is send to a pool, the train will terminate in the pool. The player can remain with the train until its next duty, but there is no way to tell what or when that duty will be, as that depends on other actions set up for that pool.

11.6.1 Additional Notes

A pool can only contain trains which are equivalent in usage. The trains need not all be same type, but their use must be exchangeable. It is not possible to select a specific type of train from a pool.

Attach, detach or transfer is not possible for trains stored in a pool. Only fixed formations (single or multiple engines, or MU's) can be extracted from or send to a pool. If multiple units are required, these must be extracted separately and coupled together after exiting from the pool. If multiple units are to be sent to a pool these must be detached before send to the pool. As attach, detach or transfers are not possible, pools can only be used by engines and MU's, i.e. for units which can move on their own. Pools can not be used for coaches and wagons or trains without power.

Pool "overflow" can occur when a train is send to a pool but the storage area is full to capacity. In this situation, the train will terminate at the access point to the pool, and will be removed.

Pool "underflow" can occur when a train is requested from a pool but the storage area is empty and no units are available. In this situation, if the flag "force creation" is set for this pool, the train will be created and will start at the access point. If this flag is not set, the train is cancelled. A warning is issued to the logfile in case of pool underflow.

11.7 Pool Definition

Pools are defined in a file similar to a timetable file, i.e. a csv spreadsheet saved as a unicode text file. The files must be stored in the same directory as the timetable files (`<route>\Activities\OpenRails`).

The layout of a pool file is considerably different compared to that of a timetable file. All parameters are located in the first column, and only one value may be defined per row. The very first row is ignored.

The file extension for normal pools is `.pool-or`; for turntable pools it is `.turntable-or`.

A file can repeat parameters to define multiple pools, which need not be related in any way.

Note that there are some key differences between non-turntable pools and turntable pools:

- For non-turntable pools, each storage path must have at least one access path; for turntable pools, access paths are independent of the storage paths.

- For non-turntable pools, storage paths are defined in the outbound direction; for turntable pools, storage paths are defined as leading away from the turntable, i.e. in the inbound direction.

11.7.1 Non-Turntable Pools

Parameters for non-turntable pools:

`#comment`

Comment only, value is ignored.

`#name`

Name of the pool. This is the name which must be used in the timetable `$pool` commands for creating, extracting or storing trains for this pool. This field is compulsory, and *must precede all other parameters*.

`#storage`

A path that defines a storage track. At least one storage track must be defined for a pool.

The path must be defined in the *outbound* direction, that is, the direction of the train when it leaves the pool.

A storage path can only be a single section; it cannot pass over switches or crossings.

`#access`

A path that defines access to a storage track. Each storage track definition must be followed by one or more access path definitions.

The path must be defined in the *outbound direction*, that is, the direction of the train when it leaves the pool.

An access path can pass over switches or crossings but can not contain any reversal points.

`#maxunits`

For each storage track, the maximum number of units which can be stored on that track can be defined. This field is optional.

Note that this defines only the maximum number of units. The effective number may be less if the length of the storage track is not sufficient to hold this number of units.

`#settings`

Contains special flags for pool usage. Currently, only one value is allowed: `force creation`, which forces trains to spawn on the access point if the pool is *underflowing*.

Additional Notes

It is not possible to define "run-through" storage areas. Access paths to storage tracks can only be defined at one end of the storage track, and trains will always enter and exit the pool at the same end.

Although each storage path has its own access path(s), it is advisable that all access paths end at the same point, such that all storage tracks are accessible from that location. It is possible to have multiple access points but then it is still advisable that all storage paths can be reached from all points.

If only part of the storage paths can be accessed from an access point, there is a risk that the trains can not be spread adequately over the full storage area. Worst case, if all trains are always send to one access point and always extracted from another access point and these points do not access all storage tracks, there may be a continuous series of pool "overflow" and "underflow" as the engines send to the pool can not be extracted.

11.7.2 Turntable Pools

Parameters for turntable pools:

`#comment`

Comment only, value is ignored.

`#name`

Name of the pool. This is the name which must be used in the timetable `$pool` commands for creating, extracting or storing trains for this pool. This field is compulsory, and *must precede all other parameters*.

`#worldfile`

The filename of the world file in which the turntable is located.

`#uid`

The uid of the turntable in the worldfile. Together with #worldfile, this defines the turntable on which the pool is based.

The `#worldfile` and `#uid` values must be the same as the related values in the turntable.dat file which defines the working timetables.

`#storage`

A path that defines a storage track. This path must be defined in the direction *leading away from the turntable*. At least one storage track must be defined.

The start position of the path must be outside the turntable area. A storage path can only be a single section; it cannot pass over switches or crossings.

`#access`

A path that defines access to a storage track. This path must be defined in the direction *leading away from the turntable*. At least one access path must be defined. The access path is not linked to a specific storage track but applies to all storage tracks as these are always accessed via the turntable.

The start position of the path must be outside the turntable area. The path can pass over switches or crossings but can not contain any reversal points.

`#maxunits`

For each storage track, the maximum number of units which can be stored on that track can be defined. This field is optional.

Note that this defines only the maximum number of units. The effective number may be less if the length of the storage track is not sufficient to hold this number of units.

`#speedmph` and `#speedkph`

These parameters define the maximum speed of train when accessing the turntable, in mph or kph. This speed will also apply to the storage tracks.

On exiting on the turntable on access paths, the train will automatically revert to the maximum speed which applied on the approach to the turntable.

With these commands, there is no need to place speedposts in the route to limit the speed on the turntable.

`#framerate`

This parameter defines the frame rate for turning the turntable. See *Turntables and Frame Rate* for details.

`#approachclearance`

This parameter sets the distance, in meters, at which the engine will stop in front of the turntable when it approaches the turntable but has to wait for turntable to align. It is also the distance at which the restricted turntable speed is applied.

`#releaseclearance`

This parameter sets the distance, in meters, at which the turntable will be released after the engine has moved off of the turntable. This is also the distance at which the turntable speed restriction will be lifted if the engine is leaving the pool.

`#settings`

Equivalent to the non-turntable pool *command* of the same name.

Using the Turntable

Do not at any time move the turntable using manual controls.

When the player train is extracted from the pool, the turntable will turn to the required position. The player train can either wait or move slowly toward the turntable. When the player train approaches the turntable on an access path and the turntable is not in the required position, stop just short of the turntable and wait until the table is in position. There will be a screen notification when the turntable is ready.

When moving onto the turntable, proceed until the engine is fully positioned on the turntable. There will be a screen notification when the engine is correctly positioned.

When the engine is positioned, set the throttle to 0% and set the reverser to neutral (or 0% for steam engines). The turntable will start to move when both conditions are met. Do not move the engine while the table is turning.

When the turntable is in the required position, the train can be moved off the table.

AI Turntable Behavior

The turntable will always move to the required position over the shortest angle.

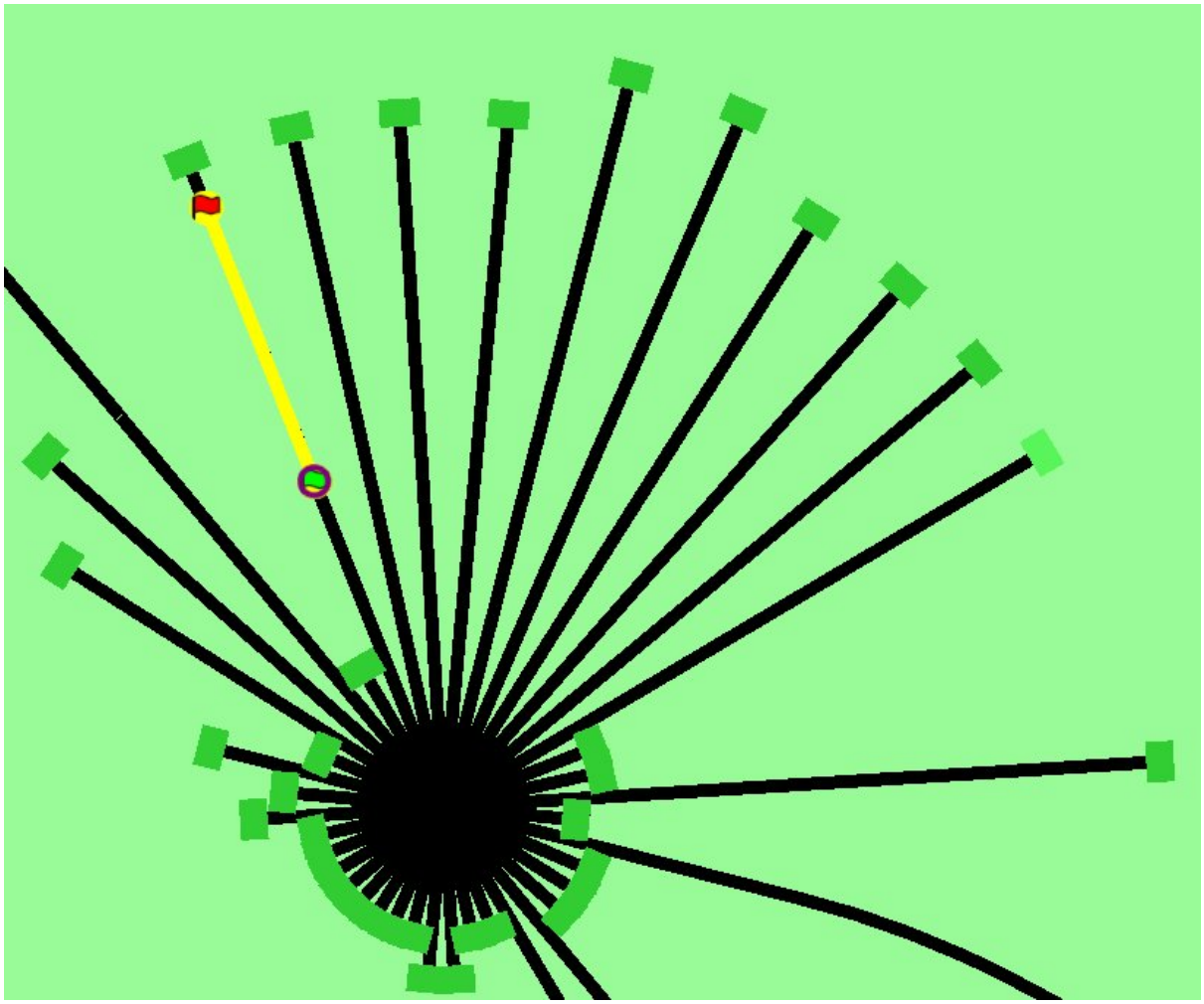
When a train requests the turntable but the turntable is already activated or occupied by another train, the request is queued. The turntable is released when the occupying engine moves off the turntable and is a short distance clear of it. If no other requests are queued, the turntable will remain in that position until the next request.

When an AI train approaches the turntable on an access path and the turntable is not in the required position, the train will stop just short of the turntable and will request the turntable to move to that position.

When an AI train is requested to exit from a storage track and the turntable is not in that position, it will request the turntable to move but will not start to move toward the turntable until the turntable is in position.

Turntable Paths

The Track Viewer will show paths leading through the turntable. Turntable paths, however, must not pass through the actual turntable itself, but rather start outside the turntable area, as shown in this image:



It is advisable to have separate access paths for extracting trains from the pool and sending trains to the pool, especially if the turntable is shared by multiple pools. Otherwise, if a train is send to the turntable at approximately the same time as another is extracted, there is a risk of a deadlock situation. The program cannot resolve this, as it cannot see that both trains are bound to proceed onto the same track while the train that is being extracted is still waiting for the turntable or is being turned.

Turntables and Frame Rate

Normally, the turntable frame rate (speed at which the table rotates) is taken from the shape file of the turntable.

However, as AI trains can use a turntable anywhere on a route, it may be that the shape file of a particular turntable which is not in view has not been loaded, and therefor the frame rate can not be derived in that way. The value as defined for the pool is used as substitute.

If at any time the turntable is used when its shape file is loaded, this substitute value is replaced by the value as defined in the shape file. One frame per second relates to a rotation speed of 0.1 degrees per second. This parameter is optional. If not defined, a default value of 30 frames per second is used, which gives a default rotation speed of 3 degrees per second.

11.8 Changing Weather

The current cloud cover, precipitation, and visibility can be varied over the course of the timetable day with weather files. Weather files reside within a special `WeatherFiles` subdirectory of the route's folder and they have the file extension `*.weather-or`. They are selected by the player from the timetable mode menu.

A weather file is a JSON file that consists of a single array, named "Changes", each item of which represents a weather event that activates at a specific time. Each event is a JSON object whose "Type" property identifies the kind of weather event. Concretely, a weather file follows the format:

```
{
    "Changes": [
        {
            "Type": "<type>",
            "<property>": "<value>"
        }
    ]
}
```

There are three types of events: `Clear`, `Precipitation`, and `Fog`, each with their own individual sets of properties.

11.8.1 "Clear" event

A Clear event removes any precipitation or fog while also setting the prevailing overcast conditions. Clear events contain the following JSON properties:

Property	Type	Description
Time	string	The 24-hour time this event activates.
Overcast	number	The overcast intensity as a percentage from 0 to 100.
OvercastVariation	number	The variation in overcast intensity as a percentage from 0 to 100.
OvercastRateOfChange	number	The rate of change of overcast intensity as a scaling factor from 0 to 1.
OvercastVisibility	number	The resulting visibility, in meters. The value must be in the range from 10000 to 60000. (For lower values, use a Fog event.)

11.8.2 "Precipitation" event

A Precipitation event represents a rain spell followed by a clear spell, with smooth transitions into, out of, and between both phases.

Property	Type	Description
Time	string	The 24-hour time this event activates.
Phase 1: Build up to precipitation		
OvercastPrecipitationStart	number	The overcast intensity during the build up to the precipitation spell as a percentage from 0 to 100.
OvercastBuildUp	number	The rate of change of overcast intensity in advance of the precipitation spell as a scaling factor from 0 to 1.
PrecipitationStartPhase	number	The duration of the precipitation build up phase, in seconds. Must be in the range from 30 to 240.
Phase 2: Precipitation spell		
PrecipitationType	string	The type of precipitation. Must be one of `Snow` or `Rain`.
PrecipitationDensity	number	The precipitation intensity as a scaling factor from 0 to 1.
PrecipitationVariation	number	The variability of the precipitation intensity as a scaling factor from 0 to 1.
PrecipitationProbability	number	The probability of the precipitation event as a percentage from 0 to 100.
PrecipitationSpread	number	The number of distinct periods of showers during the spell. Must be in the range from 1 to 1000.
PrecipitationVisibilityAtMinDensity	number	The visibility at minimum precipitation density.
PrecipitationVisibilityAtMaxDensity	number	The visibility at maximum precipitation density.
Phase 3: Dispersion after precipitation		
OvercastDispersion	number	The rate of change of overcast intensity after the precipitation spell as a scaling factor from 0 to 1.
PrecipitationEndPhase	number	The duration of the precipitation dispersion phase, in seconds. Must be in the range from 30 to 360.
Phase 4: Clear spell		
Overcast	number	The overcast intensity as a percentage from 0 to 100.
OvercastVariation	number	The variation in overcast intensity as a percentage from 0 to 100.
OvercastRateOfChange	number	The rate of change of overcast intensity as a scaling factor from 0 to 1.
OvercastVisibility	number	The resulting visibility, in meters. The value must be in the range from 10000 to 60000.

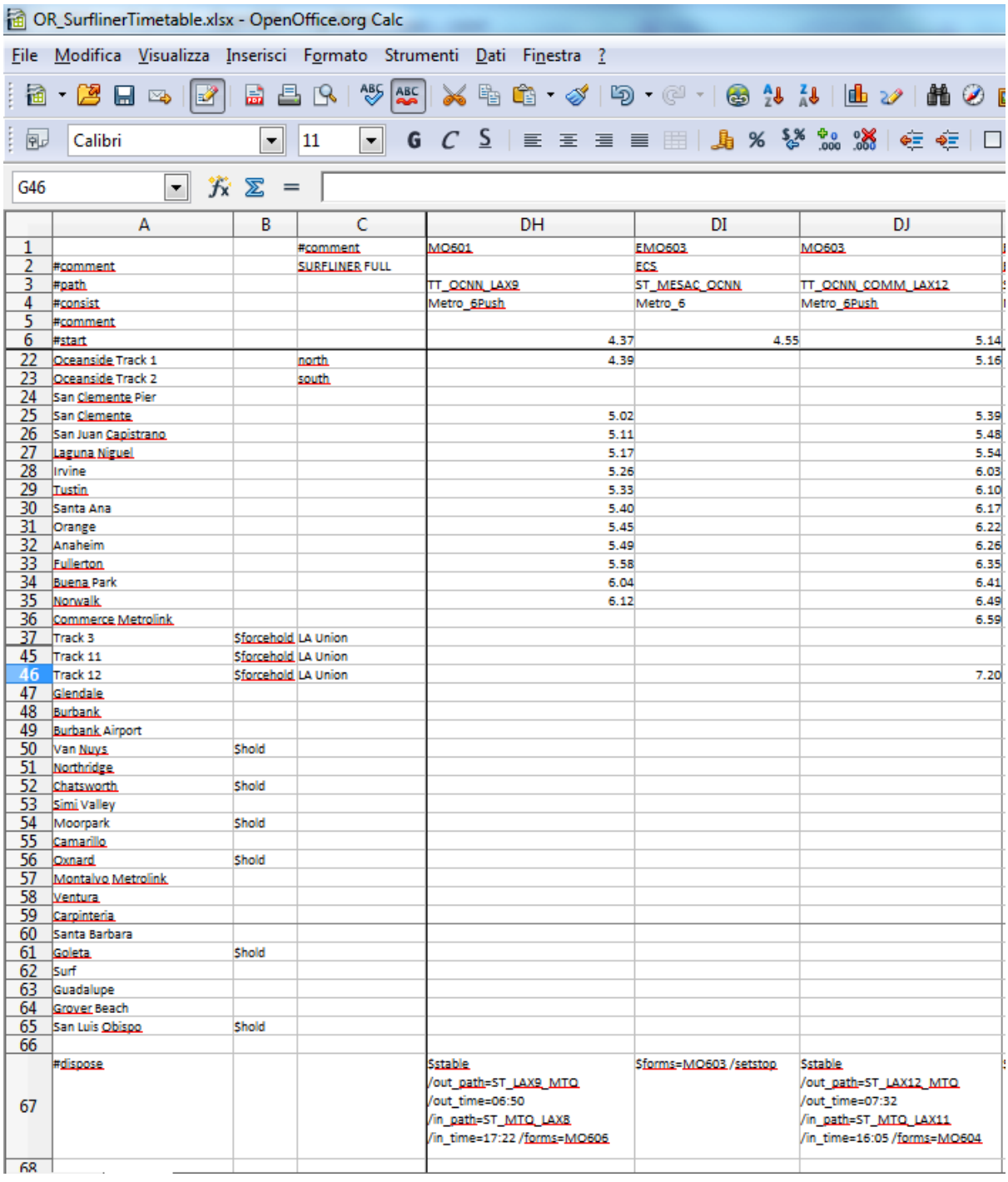
11.8.3 "Fog" event

A Fog event greatly reduces the prevailing visibility. It features smooth transitions into and out of the fog, from the previous weather event and to the next weather event.

Property	Type	Description
Time	string	The 24-hour time this event activates.
FogVisibility	number	The resulting visibility, in meters. Maximum value 1000. (For higher values, use a Clear event.)
FogSetTime	number	The transition time for fog to set in, in seconds. The value must be in the range from 300 to 3600.
FogLiftTime	number	The transition time for fog to lift, in seconds. The value must be in the range from 360 to 3600.
FogOvercast	number	The resulting overcast intensity after the fog lifts as a percentage from 0 to 100.

11.9 Example of a Timetable File

Here is an excerpt of a timetable file (shown in Excel):


OR_SurflinerTimetable.xlsx - OpenOffice.org Calc

File Modifica Visualizza Inserisci Formato Strumenti Dati Finestra ?

Calibri 11

G46

	A	B	C	DH	DI	DJ
1			#comment	MO601	EMO603	MO603
2	#comment		SURFLINER FULL		ECS	
3	#path			TT_OCNN_LAX9	ST_MESAC_OCNN	TT_OCNN_COMM_LAX12
4	#consist			Metro_6Push	Metro_6	Metro_6Push
5	#comment					
6	#start			4.37	4.55	5.14
22	Oceanside Track 1		north	4.39		5.16
23	Oceanside Track 2		south			
24	San Clemente Pier					
25	San Clemente			5.02		5.39
26	San Juan Capistrano			5.11		5.48
27	Laguna Niguel			5.17		5.54
28	Irvine			5.26		6.03
29	Tustin			5.33		6.10
30	Santa Ana			5.40		6.17
31	Orange			5.45		6.22
32	Anaheim			5.49		6.26
33	Fullerton			5.58		6.35
34	Buena Park			6.04		6.41
35	Norwalk			6.12		6.49
36	Commerce Metrolink					6.59
37	Track 3	$forcehold	LA Union			
45	Track 11	$forcehold	LA Union			
46	Track 12	$forcehold	LA Union			7.20
47	Glendale					
48	Burbank					
49	Burbank Airport					
50	Van Nuys	$hold				
51	Northridge					
52	Chatsworth	$hold				
53	Simi Valley					
54	Moorpark	$hold				
55	Camarillo					
56	Oxnard	$hold				
57	Montalvo Metrolink					
58	Ventura					
59	Carpinteria					
60	Santa Barbara					
61	Goleta	$hold				
62	Surf					
63	Guadalupe					
64	Grover Beach					
65	San Luis Obispo	$hold				
66						
67	#dispose			$stable /out_path=ST_LAX9_MTQ /out_time=06:50 /in_path=ST_MTQ_LAX8 /in_time=17:22 /forms=MO606	$forms=MO603 /setstop	$stable /out_path=ST_LAX12_MTQ /out_time=07:32 /in_path=ST_MTQ_LAX11 /in_time=16:05 /forms=MO604
68						

11.10 What tools are available to develop a Timetable?

It is recommended to use a powerful stand-alone program (Excel is not required), called Timetable Editor. It is included in the OR pack, and accessed from the *Tools* button on the OR menu.

CHAPTER 12

Open Rails Multi-Player

12.1 Goal

The Multi-Player mode implemented in this stage is intended for friends to play OR together, each assuming the role of a train engineer operating a train. There is a built-in way to compose and send text messages, but there is no built-in tool for chatting, thus players are encouraged to use Skype, Teamspeak or other tools to communicate vocally.

Each player must start and run OR on his computer. The network server may be either a special *public server* so you may not need to set up a network server from your own computer, or the dispatcher computer (see below).

12.2 Getting Started

One player starts as the dispatcher (from a network point of view his computer may be network client or network server, as explained above), and then the others start as standard players. They are always network clients and therefore they are also simply called clients. Each player (dispatcher included) will choose and operate his own consist (and locomotive), but also can jump to watch others' consists, or couple with others to work as lead and DPU through a tough route, or even act as a dispatching aid to control signals and switches manually.

12.3 Requirements

The dispatcher can start an activity or choose to explore. Clients MUST choose to explore (or a simple activity with timetable but no AI trains).

The client must select the same route played by the dispatcher.

It is not required for everyone to have the same set of paths, rolling stocks and consists.

12.4 Technical Issues

If you start the server at home, it will be necessary for you to learn your public IP address. You may also need to configure your router for port forwarding. Details to accomplish these are given in sections that follow.

It is recommended that you do not run a server for a prolonged period as the code has not been tightened for security. Only tell people you trust that you have a server started.

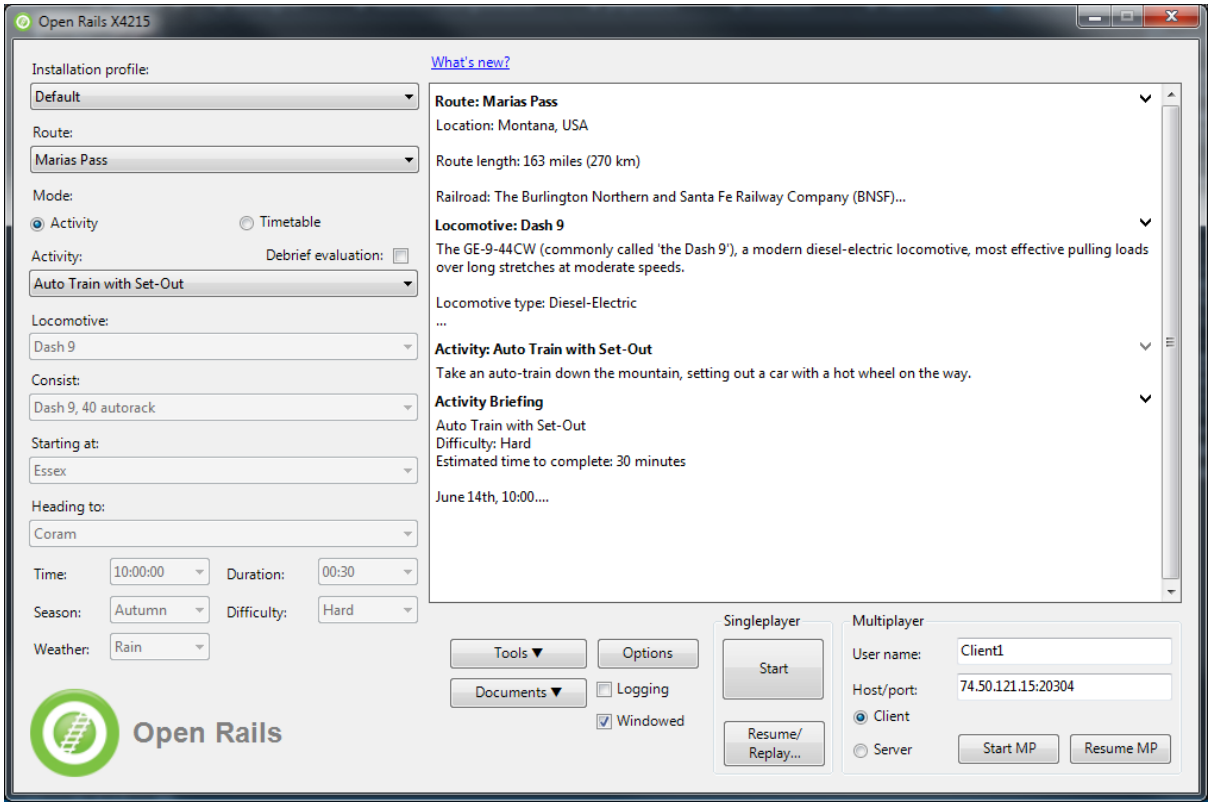
12.5 Technical Support

You can ask questions in the following forums: trainsim.com, elvastower.com, uktrainsim.com, etc.

A web forum has been set for you to post questions and announce servers. You can also request a private club so that only your friends know of your server. The forum is free to join and post: http://www.tsimserver.com/forums

12.6 Starting a Multi-Player Session

12.6.1 Starting as Dispatcher



To start as Dispatcher you must be the first player to enter the multiplayer session.

On the OR main menu you select in a standard way as described in the *Getting started* chapter on the left side Route, activity or explore route, and in case of explore route you select as usual locomotive, consist, path, time, season and weather.

On the lower right side you enter your User Name and the host and port address. If you want to run as standalone local server, or if you want to have more than one instance of OR running in MP mode on

the same computer, you must set *Host/port* to `127.0.0.1:30000`. 30000 is the default port, but you can change to any integer between 10000 and 65536.

If you want to run in a local area network usually valid host addresses are `192.168.1.2` or `192.168.1.1`.

If you use the special *public server* you need to check the *Client*"* radio button. If instead your computer acts as server, you check the *Server*"* radio button.

After having inserted the *Username* and *Host/port* data you click on *Start MP*.

Note that when using the special *public server* ,

After start, Windows Firewall may ask if you want to allow OR access to the Internet. If so, click *Allow*. If you use other firewall software, you may need to configure it to allow OpenRails to access the Internet.

There is no built-in limit of how many players can connect; a server with good Internet upload bandwidth can be expected to handle at least 10 client connections.

12.6.2 Starting as Client

On the left side of the main menu you must enter only route, path and consist. The other parameters are received from the server.

On the right side you enter your username, IP address and port of the server, you check the *Client* radio button and the you click on *Start MP*.

12.7 In-Game Controls

Once the dispatcher and clients have started and connected, to display the MultiPlayer status, you must press `<Shift+9>` to display the MultiPlayer Info window, at the bottom of it you will see the information. You can watch how many players and trains are present and how far away you are from others. You can also look if you are acting as dispatcher or as client and the username of each one.



A player joined will have the same weather, time and season as the dispatcher, no matter what are the original choices.

The player train may join the world and find that it is inside another train. Don't panic, you have two minutes to move your train out before OR thinks you want to couple with that train.

AI trains are added by the dispatcher computer (if it runs in activity mode) and broadcast to all players. As a client, do not start an activity with AI trains; moreover it is recommended that you start in Explore mode on the client.

You can jump to see other trains in sequence by pressing <Alt+9>. OpenRails will cycle through all active trains with each key press. If you are running an activity OpenRails will include in the cycle any static consists that are used in the activity. As some trains may be far away, OpenRails may need a few seconds to load the surrounding scenery. Thus you may temporarily see a blank screen. You can press F7 to see train names. You can press 9 to return to seeing your own train.

Locations of trains from other players are sent over the Internet. Because Internet routings vary moment to moment there may be some lag, and trains may jump a bit as OpenRails tries to update the locations with information received.

You can couple/decouple as usual. As coupling is controlled in the dispatcher computer, a player needs to drive slowly so that the dispatcher computer will have accurate information of train positions. If two player trains couple together, one of them will become a helper, and a message will be shown on the left indicating that the player is in Helper mode. A player in Helper mode cannot control his consist as it falls under control of the lead locomotive. By pressing <Shift+E> you can swap Helper status with another player on the train. Always press <\> and <Shift+/> to reset brakes each time after coupling/uncoupling. Note that two trains can't couple together before two minutes are passed from their presence in the Dispatcher's computer.

Players can uncouple their own trains. Players in the uncoupled trains may need to press <Shift+E> to gain control; otherwise, the uncoupled trains may become a loose consist. Always stop completely before uncoupling, otherwise weird things may happen. Players may also need to press keys for resetting brake state after uncoupling (see *here*).



Players can throw switches by pressing <G> or <Shift+G>, and the switch state will change for all players on the server. The server has a choice to disallow clients to throw switches manually.

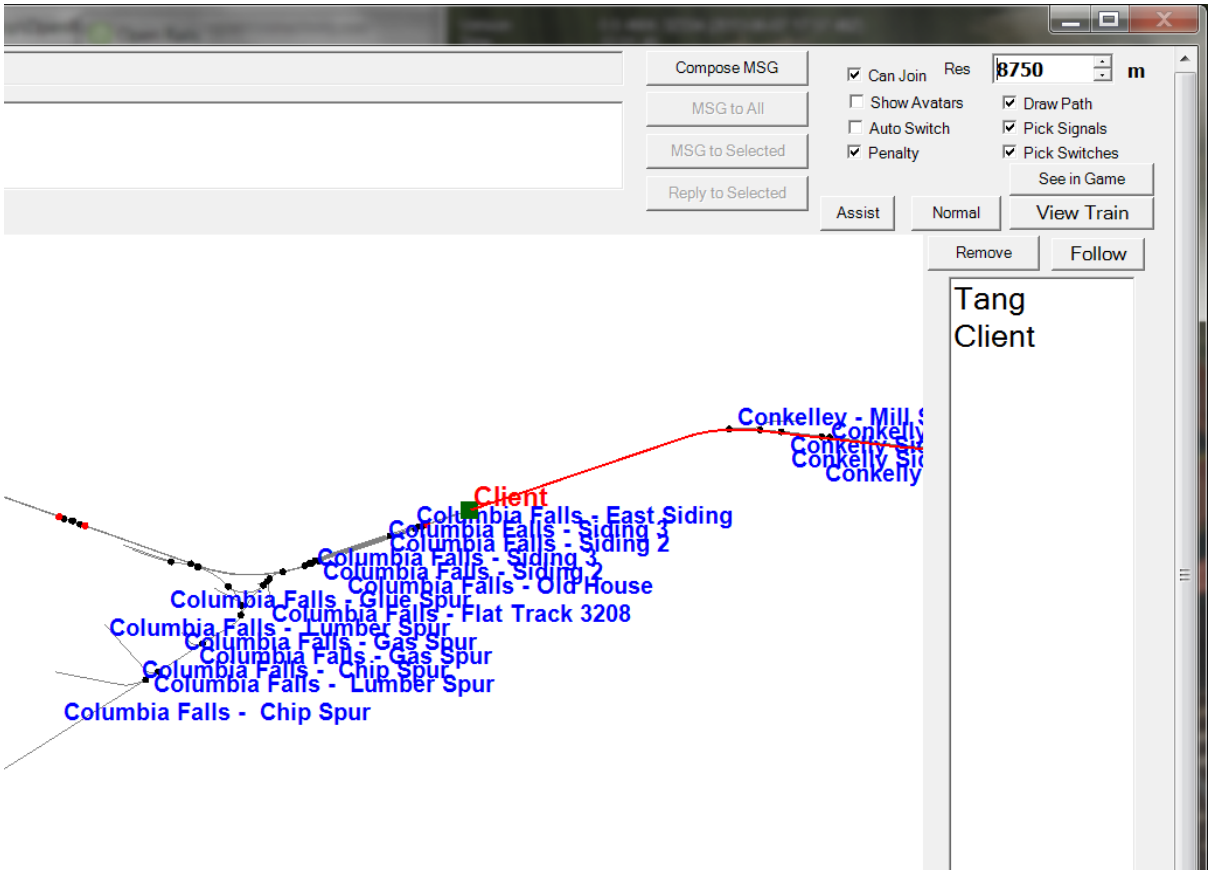
Both switches and signals are synchronized through the server (default every 10 seconds).

Player actions, such as sounding the horn or bell, turning on or off headlights, moving the pantograph up and down, opening and closing doors, moving the mirrors are broadcast to other players. Currently only the player controlled train has the cone of light shown.

A separate *Dispatcher Window* (also shown below) showing the route, signals and trains can be activated by pressing Ctrl+9. By default, it is minimized and you must click on it on the Taskbar to make it active. You can hide it by pressing <Ctrl+9> again or by pressing <Esc> when that window has the focus. This window is an extended version of the Dispatcher Window.

You can zoom in and out by rotating the mouse wheel, or by holding both the left and right mouse button and moving the mouse (if you do not have a mouse wheel). You can hold shift key while clicking the mouse in a place in the map, which will quickly zoom in with that place in focus. You can hold Ctrl while

clicking the mouse in a place in the map, which will zoom out to show the whole route. Holding Alt and clicking will zoom out to show part of the route.



A red line will be drawn for each train so you can find its intended path.

You can select a train either by clicking on the name in the right bar, or in the map by clicking the green train body. After that, you can click the *Remove* button to delete that train from the game.

You can pan the window by dragging it with the left mouse button.

One can click a switch (or signal) and press `<Ctrl+Alt+G>` to jump to that switch with the free-roam camera.

The Dispatcher player can click a switch (black dot) and choose *Main Route* or *Side Route* to switch. He can also click on a signal (green, red or orange dot) and choose to change the signal status.

The Dispatcher can choose a player and give the player right to throw switches and change signals, by clicking the button *Assist*. The right can be revoked by click the *Normal* button.

The Dispatcher can choose a player from the avatar list and remove that player from the game.

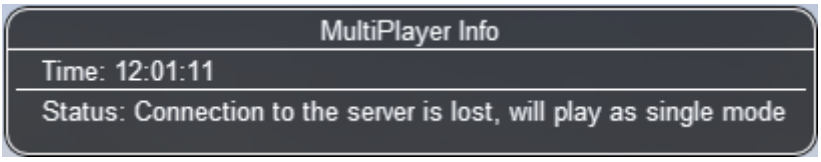
You can send a text message by typing in the top left text input area, and view the most recent 10 messages from the viewing area. You can send message to all after finishing it, or select some avatars and send a message to those selected.

12.8 Summary of Multi-Player Procedures

1. Dispatcher can start an activity or Explore. Clients must choose to Explore the route or start with an activity without AI trains.
2. Missing rolling stock in other players' consists will be automatically replaced by existing cars from local directory. This can lead to awkward consists.
3. You have two minutes after joining the game to move your train out of other trains.
4. Use <Alt+9> to see other trains, 9 to see your own train, <Ctrl+9> to view/hide the dispatcher window. Use the mouse wheel to zoom and left mouse button to pan the dispatcher window.
5. You can send and read messages from the dispatcher window
6. Use <Ctrl+Alt+F11> to see the path trains will follow, and <F7> to see train names
7. Move trains slowly when trying to couple. Trains don't couple in their first two minutes of life.
8. Use <\> and <Shift+/> (on English keyboards) just after your train is coupled or uncoupled, or when you just gain back the control of your own train.
9. Use <Shift+E> to gain control of your own train after uncoupling.
10. Use other communication tools (such as Ventrillo or Skype) to communicate with other players.
11. Always completely stop before uncoupling trains with two players coupled together

12.9 Possible Problems

- A server may not be able to listen on the port specified. Restart the dispatcher and the clients and choose another port.
- If you cannot connect to the server, verify you have the correct IP address and port number, and that the server has the port opened.
- If other players have rolling stock you do not have, that train will automatically replace cars from your own folder, and this replacement may make the consist 'interesting'.
- You may join the game and see you've selected the same start point as someone else and that your train is inside another train. Move the trains apart within two minutes and it will be fine.
- If your train is moving too quickly when trying to couple, the process may not work and weird things can happen.
- As the dispatcher has absolute control, clients may notice the switch just changed will be changed back a few seconds later if the dispatcher controlled train wants to pass it.
- Coupling/uncoupling the same set of trains may end up with weird things.
- <Ctrl+E> locomotive switch may have train cars flipped.
- When the server connection is lost, a message appears in the MultiPlayer Info window:



12.10 Using the Public Server

A special public server is deployed so that you do not need to use your own computer as the server, avoiding the setup problems you may encounter. You can find the IP and port numbers here.

To connect to this public server you must act as described *here*, using IP and port numbers as found on the above link, with only a difference: the first player entering the session has to enter by clicking on *Client* and not on *Server*, even if he intends to be the dispatcher. If the port has no player yet, whoever connects first will be declared the dispatcher, others connected later will be normal players.

The public server runs a special code that is not part of OR. If you plan to run such a server for free, please contact the email listed in http://tsimserver.com/forums/showthread.php?2560.

12.10.1 Additional info on using the Public Server

- If the computer of the player acting as dispatcher crashes or if the connection with it breaks down, the public server will try to appoint another player as dispatcher. Such player will receive on his monitor the following message: *You are the new dispatcher. Enjoy!*
- If a client crashes or loses the connection, its position is held by the server for 10 minutes. If the client re-enters the game within such time frame, it will re-enter the game in the position where he was at the moment of the crash.

12.11 Save and resume

Networked games may be prone to crashes, and it is not nice if you get a crash in the middle of a long-lasting game and you have to restart the game from its beginning.

Therefore also for multiplayer mode the *Save and resume* feature is available: it is advisable that the dispatcher regularly saves the session by pressing F2 during game.

If a crash occurs, the procedure to resume game is described here below. When the dispatcher wants to resume the session from the last save, all players must be off the game.

The dispatcher must have in his main menu path and consist as in the saved session. He clicks the *Resume MP* button and in the resume window he selects the session he wants and clicks on the *Resume* bùtton therein. When he is again in the game, he will see in the dispatcher window that the other player trains are shown in grey on the route. Now the other players have 10 minutes to re-enter the game where they were when the game was saved. They too must have in their main menu their original path and the consist they had in the moment the game was saved. They must select *Start MP* to re-enter the game. They will re-enter the game in the place where they were and with the consist they had when the game was saved. If however the train proceeded less than 1 Km from game start, the player will re-enter the game at the beginning of the path with his original consist.

As there are many possible cases, it may be possible that some of them are not covered.

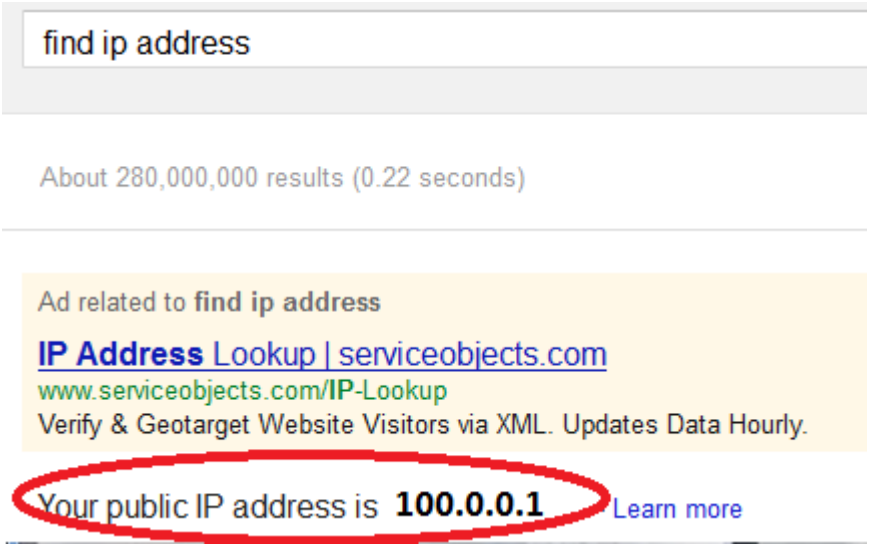
12.12 Setting up a Server from Your Own Computer

As any online game, you need to do some extra work if you want to host a multiplayer session.

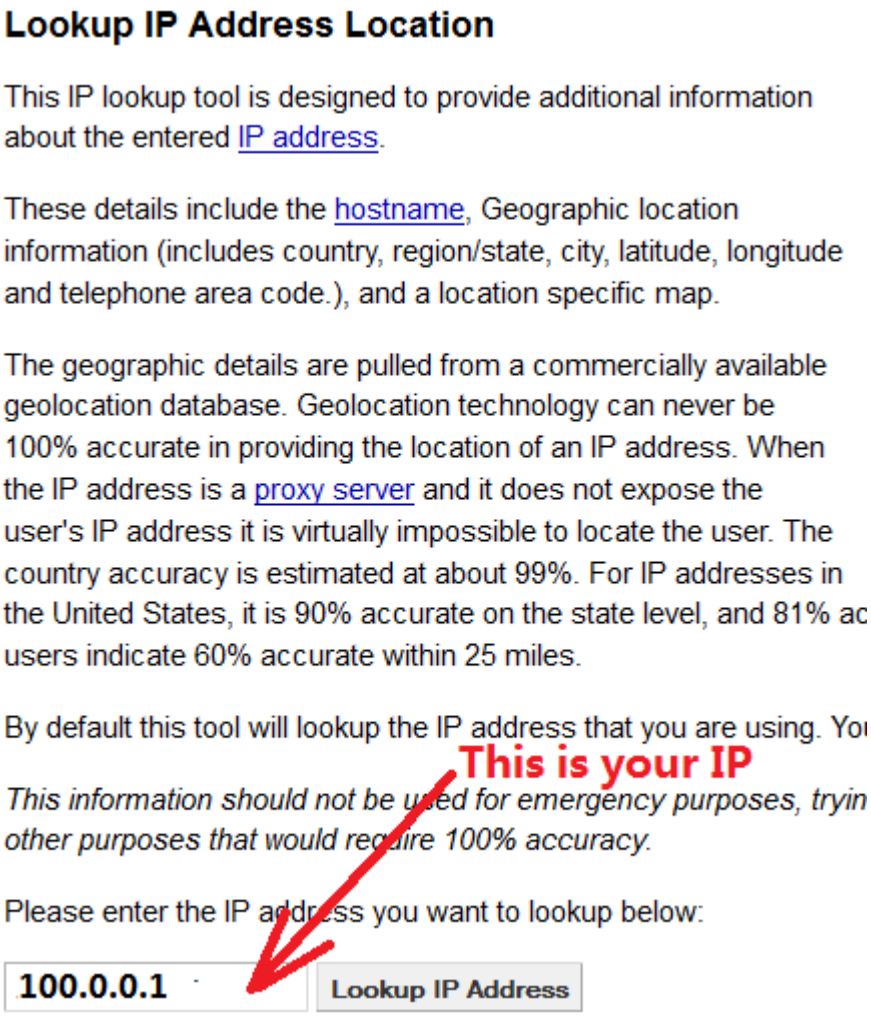
12.12.1 IP Address

If you are running at home and use a router, you may not have a permanent IP. Thus before you start as a server, you must find your IP. The quickest ways are the following:

- Using Google: type in `find ip address`, then Google will tell you



- If the above does not work, try http://whatismyipaddress.com/ip-lookup/, which shows your IP in the middle of the page.



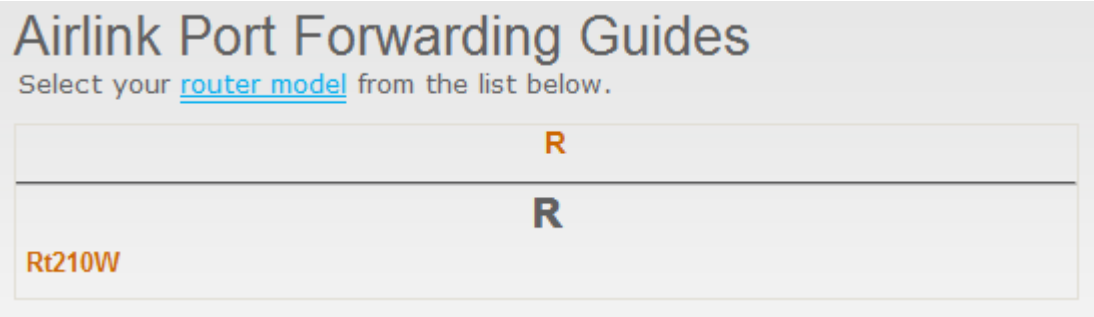
12.12.2 Port Forwarding

If you are using a router at home with several computers, your router needs to be told which computer on your home network should receive the network data OpenRails needs. This is done by enabling Port Forwarding on the router. The default port OpenRails uses is 30,000. If you change that port number in the game you'll need to change the forwarded port number in the router as well. Your router must be told to forward data arriving from the internet on the correct port to the network IP address of the computer running OpenRails. For more information on Network Address Translation (NAT) and how Port Forwarding works, see this site: http://www.4remotesupport.com/4content/remote_support_NAT.html Here the following are the steps:

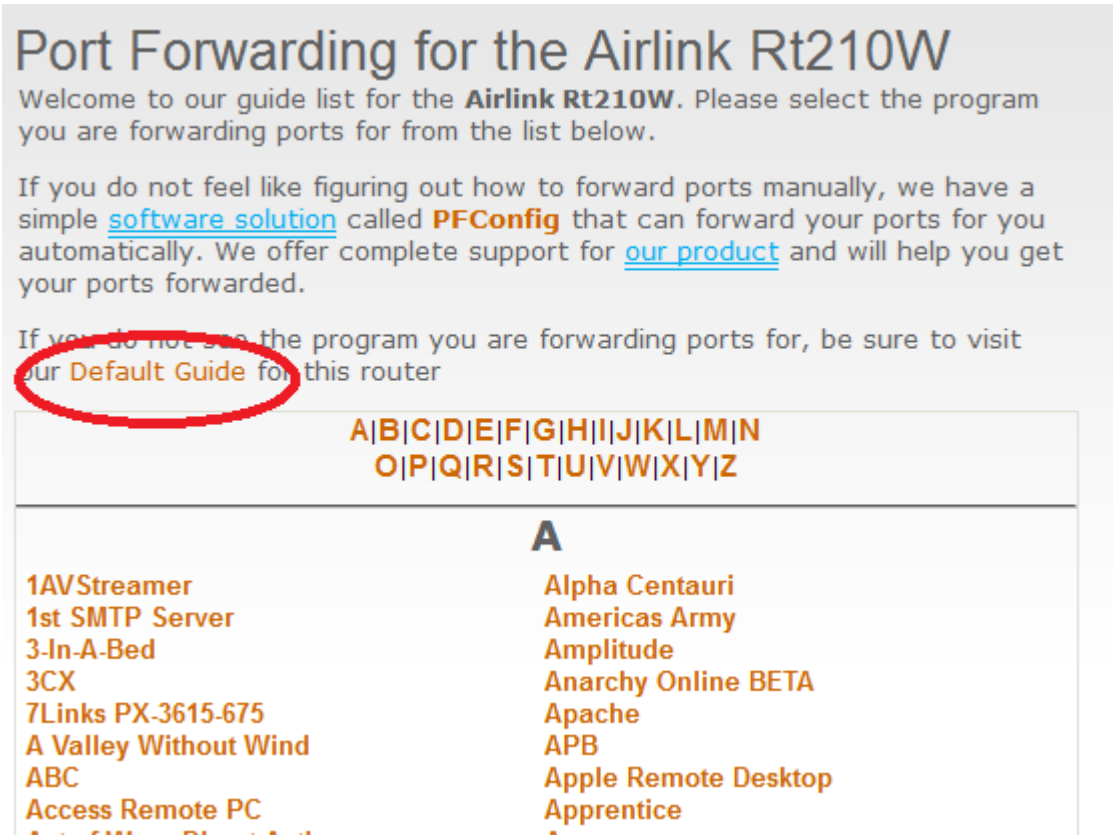
1. Go to http://portforward.com/english/routers/port_forwarding/, which contains a lot of ads - just focus on the center of this page.
2. Locate the name of the manufacturer of your router, i.e. Airlink and click it:



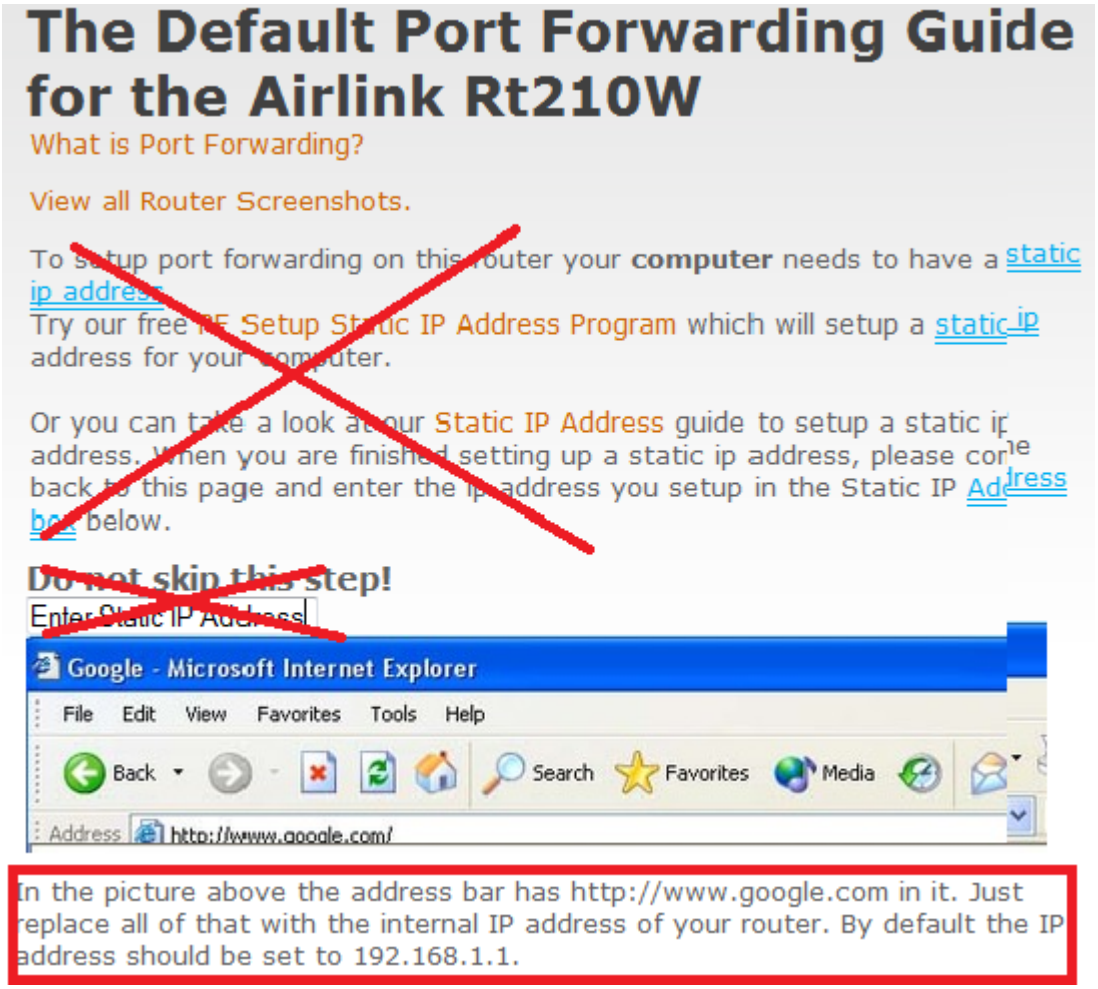
3. A page may appear allowing you to select your specific model of router:



4. It then shows all the programs (games) for which you want to forward ports. Just click 'Default Guide':



5. A page like the following should appear. Ignore the part crossed-out but pay special attention to the part enclosed in red:



6. Then follow the steps listed on the screen. Remember you want to forward port 30000 by default, but if you change that you'll have to forward the correct port.

If you still cannot get others connected to your computer, please go to http://www.tsimserver.com/forums and ask questions.

CHAPTER 13

Open Rails Sound Management

13.1 OpenRails vs. MSTS Sound Management

OR executes .sms files to a very high degree of compatibility with MSTS.

13.2 .sms Instruction Set

OR recognizes and manages the whole MSTS .sms instruction set, in a way generally compatible with MSTS. The differences are described below.

The `Activation ()` instruction behaves differently from MSTS with regard to cameras (`CabCam`, `ExternalCam` and `PassengerCam`): in general OR does not consider which cameras are explicitly activated within the .sms files. Instead, it uses a sort of implicit activation, that as a general rule works as follows:

- when in an inside view (cabview or passenger view) the related inside .sms files are heard, plus all external .sms files (with the exception of those related to the trainset where the camera is in that moment): the volume of those external files is attenuated by a 0.75 factor.
- when in an external view all external .sms files are heard.

For an .sms file to be heard, it must be within the activation distance defined in the related instruction.

A hack is available so as to hear only in the cabview some .sms files residing outside the cabview trainset. This can be used e.g. to implement radio messages. For this to work the related .sms file must be called within a .wag file, must contain an `Activation ( CabCam )` statement, and the related wagon must be within a loose consist, within a not yet started AI train or within the consist where the cabview trainset resides.

The `ScalabiltyGroup ()` instruction behaves differently from MSTS for AI trains. While MSTS uses `ScalabiltyGroup ( 0 )` for AI trains, OR uses for AI trains the same `ScalabiltyGroup` used for player trains. This way AI train sound can profit from the many more triggers active for AI trains in ORTS. For instance, Variable2 trigger is not active in MSTS for AI trains, while it is in ORTS.

If a `Stereo()` line is present within a `ScalabiltyGroup`, and a mono .wav sound is called, MSTS will play the sound at double speed. In order to have it play at the correct speed, a frequency curve halving the speed has to be inserted. OR behaves the same as MSTS in this case.

13.2.1 Discrete Triggers

Unlike MSTS, OR does not restrict the operation of some discrete triggers related to locomotives to the cabview related .sms file (usually named …cab.sms file). On OR they are all also active in the file related to the external view (usually named …eng.sms file).

OR manages the following MSTS discrete triggers:

Trigger	Function
2	DynamicBrakeIncrease (currently not managed)
3	DynamicBrakeOff
4	SanderOn
5	SanderOff
6	WiperOn
7	WiperOff
8	HornOn
9	HornOff
10	BellOn
11	BellOff
12	CompressorOn
13	CompressorOff
14	TrainBrakePressureIncrease
15	ReverserChange
16	ThrottleChange
17	TrainBrakeChange
18	EngineBrakeChange
20	DynamicBrakeChange
21	EngineBrakePressureIncrease
22	EngineBrakePressureDecrease
23	EnginePowerOn (requires MSTS Bin option)
24	EnginePowerOff (requires MSTS Bin option)
27	SteamEjector2On
28	SteamEjector2Off
30	SteamEjector1On
31	SteamEjector1Off
32	DamperChange
33	BlowerChange
34	CylinderCocksToggle
36	FireboxDoorChange
37	LightSwitchToggle
38	WaterScoopDown
39	WaterScoopUp
41	FireboxDoorClose
42	SteamSafetyValveOn
43	SteamSafetyValveOff
44	SteamHeatChange
45	Pantograph1Up
46	Pantograph1Down
47	Pantograph1Toggle
48	VigilanceAlarmReset
54	TrainBrakePressureDecrease
56	VigilanceAlarmOn
57	VigilanceAlarmOff
58	Couple
59	CoupleB (currently not managed)
60	CoupleC (currently not managed)

continues on next page

Table 1 – continued from previous page

Trigger	Function
61	Uncouple
62	UncoupleB (currently not managed)
63	UncoupleC (currently not managed)
66	Pantograph2Up (requires MSTS Bin option)
67	Pantograph2Down (requires MSTS Bin option)

MSTS .sms files for crossings (`crossing.sms`), control error and permission announcements (`ingame.sms`) together with their triggers, and for fuel tower are managed by OR.

MSTS triggers for derailment are currently not managed by OR.

MSTS .sms files related to weather (`clear_ex.sms`, `clear_in.sms`, `rain_ex.sms`, `rain_in.sms`, `snow_ex.sms`, `snow_in.sms`) are managed by OR.

The signal file (`signal.sms`) and its discrete trigger 1 is managed by OR.

Moreover, OR manages the extended set of discrete triggers provided by MSTSbin.

13.2.2 OR-Specific Discrete Triggers

OR manages the following set of new discrete triggers that were not present under MSTS. If MSTS (or MSTSbin) executes an .sms where such discrete triggers are used, it simply ignores the related statements.

In addition, OpenRails extends triggers 23 and 24 (electric locomotive power on/power off), that were introduced by MSTSbin, to diesel engines. Keys `<Shift+Y>` (for diesel player engine) and `<Ctrl+Y>` (for diesel helpers), apart from physically powering on and off the diesel engines, trigger the above triggers.

Trigger	Function
101	GearUp : for gear-based engines, triggered by the `<E>` key, propagated to all gear-based diesel engines of a train and run also for AI trains
102	GearDown : for gear-based engines, triggered by the `<Shift+E>` key, propagated to all gear-based diesel engines of a train and run also for AI trains
103	ReverserToForwardBackward : reverser moved towards the forward or backward position
104	ReverserToNeutral : reverser moved towards the neutral position
105	DoorOpen : triggered by the `<Q>` and `<Shift+Q>` keys and propagated to the wagons of the consist
106	DoorClose : triggered by the `<Q>` and `<Shift+Q>` keys and propagated to the wagons of the consist
107	MirrorOpen : triggered by the `<Shift+Q>` key
108	MirrorClose : triggered by the `<Shift+Q>` key

Triggers from 109 to 118 are used for TCS scripting, as follows:

Trigger	Function
109	TrainControlSystemInfo1
110	TrainControlSystemInfo2
111	TrainControlSystemActivate
112	TrainControlSystemDeactivate
113	TrainControlSystemPenalty1
114	TrainControlSystemPenalty2
115	TrainControlSystemWarning1
116	TrainControlSystemWarning2
117	TrainControlSystemAlert1
118	TrainControlSystemAlert2

Triggers from 121 to 136 are used to synchronize steam locomotive chuffs with wheel rotation. The sixteen triggers are divided into two wheel rotations. Therefore every trigger is separated from the preceding one by a rotation angle of 45 degrees.

Triggers 137 and 138 are used for the cylinder cocks of steam locomotives:

Trigger	Function
137	CylinderCocksOpen : triggered when cylinder cocks are opened
138	CylinderCocksClose : triggered when cylinder cocks are closed

Triggers from 139 to 143 can be used to make looped brake sounds:

Trigger	Function
139	TrainBrakePressureStoppedChanging : for rolling stock equipped with train brakes, to use with triggers 14 and 54, triggered when the automatic brake pressure stops changing
140	EngineBrakePressureStoppedChanging : for locomotives with engine/independent brakes, to use with triggers 21 and 22, triggered when the engine brake pressure stops changing
141	BrakePipePressureIncrease : for rolling stock equipped with train brakes, triggered when brake pipe/brakeline pressure increases
142	BrakePipePressureDecrease : for rolling stock equipped with train brakes, triggered when brake pipe/brakeline pressure decreases
143	BrakePipePressureStoppedChanging : for rolling stock equipped with train brakes, triggered when brake pipe/brakeline pressure stops changing

Trigger	Function
145	WaterScoopRaiseLower
146	WaterScoopBroken

Trigger	Function
147	SteamGearLeverToggle : Toggles when steam gear lever is moved.
148	AIFiremanSoundOn : AI fireman mode is on.
149	AIFiremanSoundOff : AI fireman mode is off, ie in Manual Firing mode.

Triggers from 150 to 158 are used for the circuit breaker sounds.

The following triggers are activated when the state of the circuit breaker changes:

Trigger	Function
150	CircuitBreakerOpen
151	CircuitBreakerClosing
152	CircuitBreakerClosed

The following triggers are activated when the driver moves the buttons or switches in the cab:

Trigger	Function
153	CircuitBreakerClosingOrderOn
154	CircuitBreakerClosingOrderOff
155	CircuitBreakerOpeningOrderOn
156	CircuitBreakerOpeningOrderOff
157	CircuitBreakerClosingAuthorizationOn
158	CircuitBreakerClosingAuthorizationOff

Trigger 161 is activated when the cab light is switched on or off.

The following triggers are activated when the state of the cab radio changes (see *here*):

Trigger	Function
162	Cab radio switched on
163	Cab radio switched off

The following triggers are activated when the state of the engines different from the first one change state in a diesel locomotive (see *here*):

Trigger	Function
167	Second engine power on
168	Second engine power off

Following triggers are activated when a 3rd and a 4th Pantograph are present on the locomotive:

Trigger	Function
169	Pantograph3Up
170	Pantograph3Down
171	Pantograph4Up
172	Pantograph4Down

Additional triggers:

Trigger	Function
173	HotBoxBearingOn
174	HotBoxBearingOff
175	BoilerBlowdownOn
176	BoilerBlowdownOff

Triggers from 189 to 198 are activated when the driver moves the following buttons or switches in the cab (related to power supplies):

Trigger	Function
189	BatterySwitchOn
190	BatterySwitchOff
191	BatterySwitchCommandOn
192	BatterySwitchCommandOff
193	MasterKeyOn
194	MasterKeyOff
195	ServiceRetentionButtonOn
196	ServiceRetentionButtonOff
197	ServiceRetentionCancellationButtonOn
198	ServiceRetentionCancellationButtonOff

The following triggers are used to activate the gear positions:

Trigger	Function
200	GearPosition0
201	GearPosition1
202	GearPosition2
203	GearPosition3
204	GearPosition4
205	GearPosition5
206	GearPosition6
207	GearPosition7
208	GearPosition8

Additional triggers for vacuum brakes:

Trigger	Function
210	LargeEjectorOn
211	LargeEjectorOff
212	SmallEjectorOn
213	SmallEjectorOff

Triggers from 214 to 222 are used for the traction cut-off relay sounds of Diesel locomotives.

The following triggers are activated when the state of the traction cut-off relay changes:

Trigger	Function
214	TractionCutOffRelayOpen
215	TractionCutOffRelayClosing
216	TractionCutOffRelayClosed

The following triggers are activated when the driver moves the buttons or switches in the cab:

Trigger	Function
217	TractionCutOffRelayClosingOrderOn
218	TractionCutOffRelayClosingOrderOff
219	TractionCutOffRelayOpeningOrderOn
220	TractionCutOffRelayOpeningOrderOff
221	TractionCutOffRelayClosingAuthorizationOn
222	TractionCutOffRelayClosingAuthorizationOff

Triggers from 223 to 226 are used for the electric train supply sounds.

The following triggers are activated when the state of the electric train supply changes:

Trigger	Function
223	ElectricTrainSupplyOn
224	ElectricTrainSupplyOff

The following triggers are activated when the driver moves the buttons or switches in the cab:

Trigger	Function
225	ElectricTrainSupplyCommandOn
226	ElectricTrainSupplyCommandOff

Triggers from 227 to 235 are activated for passenger cars (and locomotives when custom power supply scripts are used):

Trigger	Function
227	PowerConverterOn
228	PowerConverterOff
229	VentilationHigh
230	VentilationLow
231	VentilationOff
232	HeatingOn
233	HeatingOff
234	AirConditioningOn
235	AirConditioningOff

Triggers from 240 to 243 associated to the two generic items (see *here*) :

Trigger	Function
240	GenericItem1On
241	GenericItem1Off
242	GenericItem2On
243	GenericItem2Off

13.2.3 Variable Triggers

ORTS

The sound objects attached to a vehicle (wagon or loco) can respond in volume and frequency to changes in the vehicle's properties. There are 7 properties:

- distance squared from a sound source (m^2)
- speed (m/s)
- pressure in the brake cylinder (psi)
- centrifugal force due to traversing a curve (N)
- 3 variables in range 0 - 1:
  - Variable1 reflects the throttle
  - Variable2 reflects the engine's RPM (diesel) or Tractive Force (electric) or cylinder pressure (steam)
  - Variable3 reflects the dynamic brake (diesel | electric) or fuel rate (steam)

Note: Separately, for a whole route, sounds for all curves below a certain radius can be automatically triggered as vehicles pass - see *Automatic switch and curve squeal track sound* below.

Comparison with MSTS

OR manages all of the variable triggers managed by MSTS. There can be some difference in the relationship between physical locomotive variables (e.g. Force) and the related variable. This applies to Variable2 and Variable3.

New variables introduced by OR:

- BrakeCyl, which contains the brake cylinder pressure in PSI. Like the traditional MSTS variables, it can be used to control volume or frequency curves (`BrakeCylControlled`) and within variable triggers (`BrakeCyl_Inc_Past` and `BrakeCyl_Dec_Past`).

- CurveForce, in Newtons when the rolling stock is in a curve. Can be used for curve flange sounds, with two volume curves: one is `SpeedControlled`, which makes the sound speed dependent too, and `CurveForceControlled`. Of course `CurveForce_Inc_Past`, and `CurveForce_Dec_Past` are also available for activating and deactivating the sound.

13.2.4 Sound Loop Management

Sound loop management instructions are executed as follows by OR:

- `StartLoop` / `ReleaseLoopRelease`: the .wav file is continuously looped from beginning to end; when the ReleaseLoopRelease instruction is executed, the .wav file is played up to its end and stopped.
- `StartLoopRelease` / `ReleaseLoopRelease`: the .wav file is played from the beginning up to the last CuePoint, and then continuously looped from first to last CuePoint; when the `ReleaseLoopRelease` instruction is executed, the .wav file is played up to its end and stopped.
- `StartLoopRelease` / `ReleaseLoopReleaseWithJump`: the .wav file is played from the beginning up to the last CuePoint, and then continuously looped from the first to the last CuePoint. When the `ReleaseLoopReleaseWithJump` instruction is executed, the .wav file is played up to the next CuePoint, then jumps to the last CuePoint and stops. It is recommended to use this pair of instructions only where a jump is effectively needed, as e.g. in horns; this because this couple of instructions is more compute intensive and can lead to short sound breaks in the case of high CPU loads.

13.2.5 Testing Sound Files at Runtime

The *sound debug window* is a useful tool for testing.

13.3 Discrete triggers for container cranes

Trigger	Function
1	CraneXAxisMove
2	CraneXAxisSlowDown
3	CraneYAxisMove
4	CraneYAxisSlowDown
5	CraneZAxisMove
6	CraneZAxisSlowDown
7	CraneYAxisDown (triggers when grabber hits container)

13.4 Automatic switch and curve squeal track sound

With this feature a specific track sound is played when a train passes over any switch or crossover, or over a curve with a low radius, which highly enhances the sound experience. If this feature is enabled there is no more need to lay down specific sound regions around or sound sources above every switch or over curves. This is a lengthy task, and in fact most of the routes aren't equipped with such sound regions or sound sources. Three automatic sounds are supported:

```
- switch sound
- curve squeal sound
- curve + switch sound (when wagon is both on curve and switch).
```

It is possible to define also only one or two of these automatic sounds. If switch and curve squeal sound are defined, and no curve + switch sound is defined, the curve squeal sound is played when a wagon is both on curve and switch. The curve radius threshold below which the curve squeal sound is played is 350 meters for freight wagons and 301 meters for all other trainsets.

To enable this feature steps here below must be followed:

1. Suitable external and internal automatic sounds must be available (.sms files); usually you find them in the root's SOUND. It often occurs that switch track and curve squeal sounds are available in modern routes. If not, they must be created or searched on the web. A test sound set may be downloaded from here.

2. For every route it must be checked whether a reference to the three automatic track sounds are present in the route's `ttype.dat` file. If they are, you can proceed to next step. Else you must insert three new lines at the end of `ttype.dat`, adding the reference to the automatic track sounds, and you must add 3 to the number on top of the file. Here below an example of a default `ttype.dat` can be found, where three new lines referring to the above test sound have been added in last position:

```
SIMISA@@@@@@@@@@JINX0t1t______

13
TrackType ( "Default" "EuropeSteamTrack0In.sms" "EuropeSteamTrack0Ex.sms" )
TrackType ( "Concrete Supported"   "EuropeSteamTrack1In.sms" "EuropeSteamTrack1Ex.sms
↪" )
TrackType ( "Wood Supported"       "EuropeSteamTrack2In.sms" "EuropeSteamTrack2Ex.sms
↪" )
TrackType ( "In Tunnel" "EuropeSteamTrack3In.sms" "EuropeSteamTrack3Ex.sms" )
TrackType ( "Steel Bridge" "EuropeSteamTrack4In.sms" "EuropeSteamTrack4Ex.sms" )
TrackType ( "Girder Bridge" "EuropeSteamTrack5In.sms" "EuropeSteamTrack5Ex.sms" )
TrackType ( "Under Bridge" "EuropeSteamTrack6In.sms" "EuropeSteamTrack6Ex.sms" )
TrackType ( "Concrete Bridge" "EuropeSteamTrack7In.sms" "EuropeSteamTrack7Ex.sms" )
TrackType ( "Crossing Platform" "EuropeSteamTrack8In.sms" "EuropeSteamTrack8Ex.sms" )
TrackType ( "Wooden Bridge" "EuropeSteamTrack9In.sms" "EuropeSteamTrack9Ex.sms" )
TrackType ( "Switch" "DemoAutoSound/switchtrackin.sms" "DemoAutoSound/switchtrackex.
↪sms"     )
TrackType ( "Squeal Curve" "DemoAutoSound/curvesquealtrackin.sms" "DemoAutoSound/
↪curvesquealtrackex.sms"    )
TrackType ( "Squeal Switch" "DemoAutoSound/curveswitchtrackin.sms" "DemoAutoSound/
↪curveswitchtrackex.sms"    )
```

3. For every route you must tell OR which of the ttype sound files are those related to automatic sounds. This is done by inserting following line in the route's `.trk` file:

```
ORTSSwitchSMSNumber ( 10 )
ORTSCurveSMSNumber ( 11 )
ORTSCurveSwitchSMSNumber ( 12 )
```

   A better solution, because it leaves the `.trk` file unaltered, is to create an `OpenRails` subfolder within the route's folder, and to put in it an integration `.trk` file, named like the base one, and with following sample content (supposing the base .trk file is named `ITALIA13.trk`:

```
-> BLANK LINE HERE <-
include ( "../ITALIA13.trk" )
   ORTSDefaultTurntableSMS ( turntable.sms )
   ORTSSwitchSMSNumber ( 10 )
   ORTSCurveSMSNumber ( 11 )
   ORTSCurveSwitchSMSNumber ( 12 )
```

Note that a blank line must be present above the `include` line, but that is difficult to reproduce in this manual.

Note also that with the same integration `.trk` file also the default turntable sound is defined, in case this route has turntables or transfertables.

As already stated, you can also define in `ttype.dat` and in the `.trk` file only one or only two types of automatic sounds.

13.5 Override % of external sound heard internally for a specific trainset

External sounds are reproduced at a lower volume when heard within a cab or passenger view. The % of external sound heard internally is defined in the `Audio Options` menu window.

This percentage may be overridden for any trainset inserting in the Wagon section of any .eng or .wag file (or in their "include" file as explained *here*) following line:

```
ORTSExternalSoundPassedThroughPercent ( 50 )
```

where the number in parenthesis may be anyone from 0 (nothing heard internally) to 100 (external sound reproduced at original volume).

CHAPTER 14

Open Rails Cabs

OR supports both MSTS-compatible 2D cabs as well as native 3D cabs, even on the same locomotive.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

14.1 2D Cabs

OR supports with a high degree of compatibility all functions available in MSTS for 2D cabs, and provides some significant enhancements described in the next paragraphs.

14.1.1 ETCS circular speed gauge

You can add to the cabview a circular speed gauge accordingly to the European standard train control system ETCS.



For content developers

The gauge is added by the insertion of a block like the following into the .cvf file:

```
Digital (
    Type ( SPEEDOMETER DIGITAL )
    Style ( NEEDLE )
    Position ( 160 255 56 56 )
    ScaleRange ( 0 250 )
    Units ( KM_PER_HOUR )
)
```

It is also possible to display the full ETCS display using the following block instead:

```
ScreenDisplay (
        Type ( ORTS_ETCS SCREEN_DISPLAY )
        Position ( 280 272 320 240 )
        Units ( KM_PER_HOUR )
        Parameters (
                Mode FullSize
        )
)
```

The following DMI size variants are available: FullSize (displays the whole DMI), SpeedArea (displays only the left part with information about distance and speed) and PlanningArea (displays only the planning area and navigation buttons).

The information displayed in the DMI is controlled via the TCS script. For more details, see *C# engine scripting - Train Control System*.

14.1.2 Battery switch

The *battery switch* controls the low voltage power supply of the locomotive.

The following controls are available for the cabview:

- `ORTS_BATTERY_SWITCH_COMMAND_SWITCH` can be used if the switch is directly controlled from the cab
- `ORTS_BATTERY_SWITCH_COMMAND_BUTTON_CLOSE` and `ORTS_BATTERY_SWITCH_COMMAND_BUTTON_OPEN` can be used if the switch is controlled with two pushbuttons (one to close the switch and the other to open it)
- `ORTS_BATTERY_SWITCH_ON` can be used to control a light on the cab showing the state of the battery switch

Other controls can be hidden if the low voltage power supply is not available using the following parameter:

```
TwoState (
   Type ( ORTS_CIRCUIT_BREAKER_CLOSED TWO_STATE)
   ...
   DisabledIfLowVoltagePowerSupplyOff ( 1 )
)
```

14.1.3 Master key

The *master key* controls the power supply of the cab.

The following controls are available for the cabview:

- `ORTS_MASTER_KEY` can be used in order to control the master key
- `ORTS_CURRENT_CAB_IN_USE` can be used to indicate that the current cab is active
- `ORTS_OTHER_CAB_IN_USE` can be used to indicate that another cab of the train is active

Other controls can be hidden if the cab power supply is not available using the following parameter:

```
TwoState (
   Type ( ORTS_CIRCUIT_BREAKER_CLOSED TWO_STATE)
   ...
```

(continues on next page)

(continued from previous page)

```
   DisabledIfCabPowerSupplyOff ( 1 )
)
```

14.1.4 Service retention

The *service retention* can be used to disable a cab without cutting the power on the train. It can only be used with a power supply script that uses this functionality.

The following controls are available for the cabview:

- `ORTS_SERVICE_RETENTION_BUTTON` can be used in order to enable the service retention
- `ORTS_SERVICE_RETENTION_CANCELLATION_BUTTON` can be used in order to cancel the service retention

14.1.5 Electric train supply

The *electric train supply* controls the power line that supplies the passenger cars with electricity.

The following controls are available for the cabview:

- `ORTS_ELECTRIC_TRAIN_SUPPLY_COMMAND_SWITCH` can be used to control the electric train supply switch
- `ORTS_ELECTRIC_TRAIN_SUPPLY_ON` can be used to indicate that the electric train supply line is powered on

14.1.6 Controls to switch on and off diesel engines

The keyboard keys to switch on and off diesel engines are following ones:

- Ctrl+Y switches on and off the first diesel engine of the player locomotive
- Shift+Y switches on and off the other diesel engines of the player locomotive, plus all diesel engines of the further locomotives in the train, if they are MUed, (that is under control of the player locomotive) which is the default.

Following cabview controls are available:

ORTS_PLAYER_DIESEL_ENGINE: the first frame is displayed when the diesel engine of the player locomotive is in states stopped or stopping, while the second frame is displayed when it is in states running or started. The control may be used with the mouse and starts/stops the (first) diesel engine of the player locomotive, and is useful when a single two state lever is used to start/stop the engine.

Example:

```
TwoState (
        Type ( ORTS_PLAYER_DIESEL_ENGINE TWO_STATE)
        Position ( 150 446 27 26 )
        Graphic ( graphic1.ace )
        NumFrames ( 2 2 1 )
        Style ( ONOFF )
        MouseControl ( 1 )
)
```

ORTS_HELPERS_DIESEL_ENGINES: the first frame is displayed when further diesel engines of the player locomotive and/or the diesel engines of the helper locomotives are in states stopped or stopping, while the second frame is displayed when they are in states running or started. The control may be used with the mouse and starts/stops further diesel engines of the player locomotive and the diesel engines of

the helper locomotives, and is useful when a two state button or lever is used. Note therefore that this command can be used also for player locomotives with more than one engine.

Example:

```
TwoState (
        Type ( ORTS_HELPERS_DIESEL_ENGINES TWO_STATE)
        Position ( 190 446 27 26 )
        Graphic ( graphics2.ace )
        NumFrames ( 2 2 1 )
        Style ( ONOFF )
        MouseControl ( 1 )
)
```

ORTS_PLAYER_DIESEL_ENGINE_STATE: this control respectively selects frames 0, 1, 2, 3 for the player locomotive engine states Stopped, Starting, Running and Stopping. It is a display-only control.

Example:

```
MultiState (
        Type ( ORTS_PLAYER_DIESEL_ENGINE_STATE TRI_STATE)
        Position ( 270 446 39 40 )
        Graphic ( cd_363_zberace.ace )
        NumFrames ( 4 4 1 )
        Style ( NONE )
        MouseControl ( 1 )
        Orientation ( 0 )
        DirIncrease ( 1 )
)
```

ORTS_PLAYER_DIESEL_ENGINE_STARTER: it displays the second frame when the player diesel engine is in starting status, and the first one in all other cases. It may be used with the mouse and it can only start the engine, therefore it is useful in conjunction with ORTS_PLAYER_DIESEL_ENGINE_STOPPER when starting and stopping the engine is done with separate commands (e.g. 2 buttons).

Example:

```
TwoState (
        Type ( ORTS_PLAYER_DIESEL_ENGINE_STARTER TWO_STATE)
        Position ( 310 446 27 26 )
        Graphic ( graphics3.ace )
        NumFrames ( 2 2 1 )
        Style ( PRESSED )
        MouseControl ( 1 )
)
```

ORTS_PLAYER_DIESEL_ENGINE_STOPPER: it displays the second frame when the player diesel engine is in stopping status, and the second one in all other cases. It may be used with the mouse and it can only stop the engine, therefore it is useful when starting and stopping the engine is done with separate commands (e.g. 2 buttons).

Example:

```
TwoState (
        Type ( ORTS_PLAYER_DIESEL_ENGINE_STOPPER TWO_STATE)
        Position ( 350 446 27 26 )
        Graphic ( Bell.ace )
        NumFrames ( 2 2 1 )
        Style ( PRESSED )
        MouseControl ( 1 )
)
```

14.1.7 Cab radio

OR supports the cab radio cabview control. Pressing keys Alt+R switches on and off the cab radio. Switching on and off the cab radio enables discrete sound triggers 162 and 163, as explained *here*. Here is an example of a cab radio control block within the .cvf file:

```
            TwoState (
Type ( CAB_RADIO TWO_STATE )
                  Position ( 150 425 30 21 )
                  Graphic ( Horn.ace )
                  NumFrames ( 2 2 1 )
                  Style ( ONOFF )
                  MouseControl ( 1 )
)
```

14.1.8 Cab light

OR supports the cab light cabview control. Pressing key L switches on and off the cab light under the same conditions applicable to MSTS. Switching on and off the cab light enables discrete sound trigger 161, as explained *here*. Here is an example of a cab light control block within the .cvf file:

```
            TwoState (
Type ( ORTS_CABLIGHT TWO_STATE )
                  Position ( 120 425 30 21 )
                  Graphic ( Horn.ace )
                  NumFrames ( 2 2 1 )
                  Style ( ONOFF )
                  MouseControl ( 1 )
)
```

14.1.9 Dedicated buttons for brake controllers

In addition to the BailOff keyboard command, a cabview control named ORTS_BAILOFF is available. It is used to release the brakes of the engine while keeping the train brakes applied.

In some brake controllers, there is a button that provides a full and quick release of the train brake when pressed. OR supports this via the ORTS_QUICKRELEASE cabview control.

Some brake controllers have a dedicated button to overcharge the brake pipe. The ORTS_OVERCHARGE cabview control can be used for this purpose.

Here is an example of one of this controls within the .cvf file:

```
            TwoState (
Type ( ORTS_BAILOFF TWO_STATE )
                  Position ( 120 425 30 21 )
                  Graphic ( BailOff.ace )
                  NumFrames ( 2 2 1 )
                  Style ( PRESSED )
                  MouseControl ( 1 )
)
```

14.1.10 Signed Traction Braking control

This cabview control shows the signed value of the force (+ve or -ve, that is tractive or due to dynamic braking) as displayed in many real loco cabs. The control is ORTS_SIGNED_TRACTION_BRAKING. For comparison, the MSTS-compatible TRACTION_BRAKING cabview control shows the absolute value of the force. Here is an example of a cab light control block within the .cvf file:

```
Dial (
    Type ( ORTS_SIGNED_TRACTION_BRAKING DIAL )
    Position ( 319 223 3 32 )
    Graphic ( ../../Common.Cab/CabE464/AgoDin.ace )
    Style ( NEEDLE )
    ScaleRange ( -761 1600 )
    ScalePos ( 190 70 )
    Units ( AMPS )
    Pivot ( 36 )
    DirIncrease ( 0 )
    )
```

14.1.11 Signed Traction Total Braking control

ORTS_SIGNED_TRACTION_TOTAL_BRAKING control behaves and is defined like ORTS_SIGNED_TRACTION_BRAKING, with the only difference that the braking force does include also the train brake force in addition to the dynamic brake force.

14.1.12 Odometer controls

Following cabview controls are available:

- ORTS_ODOMETER: used to digitally display the odometer value
- ORTS_ODOMETER_RESET: used to reset the odometer
- ORTS_ODOMETER_DIRECTION_CHANGE: used to change direction (up/down) of the odometer.

Following units of measure are available for ORTS_ODOMETER:

- KILOMETRES
- METRES
- MILES
- FEET
- YARDS

The operation of the odometer is explained *here*.

Here is an example of use of the odometer control blocks within a .cvf file:

```
TwoState (
  Type ( ORTS_ODOMETER_RESET TWO_STATE )
  Position ( 320 70 24 22 )
  Graphic ( OdoResetButton.ace )
  NumFrames ( 2 2 1 )
  Style ( WHILE_PRESSED )
  MouseControl ( 1 )
)
TwoState (
  Type ( ORTS_ODOMETER_DIRECTION TWO_STATE)
```

(continues on next page)

(continued from previous page)

```
  Position ( 320 100 13 15 )
  Graphic ( OdoDirectionSwitch.ace )
  NumFrames ( 2 2 1 )
  Style ( ONOFF )
  MouseControl ( 1 )
)
Digital (
  Type ( ORTS_ODOMETER DIGITAL)
  Position ( 377 100 26 17 )
  ScaleRange ( 0 100000 )
  Accuracy ( 0 )
  AccuracySwitch ( 0 )
  LeadingZeros ( 0 )
  Justification ( 1 )
  PositiveColour ( 1
   ControlColour ( 255 255 255 )
  )
  NegativeColour ( 0 )
  DecreaseColour ( 0 )
  Units ( FEET )
)
```

14.1.13 Distributed Power

The principles of Distributed Power are described *here* .

Distributed Power data can be displayed using control ORTS_DISTRIBUTED_POWER. Here an example of use:

```
     ScreenDisplay (
   Type ( ORTS_DISTRIBUTED_POWER SCREEN_DISPLAY )
   Position (  164.4 286.5 136 52 )
             Parameters (
      FullTable True
      LoadUnits AMPS
             )
   Units ( KM_PER_HOUR )
             ORTSDisplay ( 1 )
             ORTSScreenPage ( "2300-0" )
)
```

Here below an example of the output of the above control.



When parameter FullTable is set to False, only the first 6 lines are displayed. Optional parameter LoadUnits defines which is the UoM used for the Load field. Default is AMPS in a metric environment and KILO_LBS in the other cases. Selectable LoadUnits are AMPS, NEWTONS, KILO_NEWTONS, LBS and KILO_KBS.

The screen display can be rotated in 2D cabs adding parameter ORTSAngle (number) in the ScreenDisplay block. The angle is in degrees.

Info specific for 3D cabs can be found *here* .

For every keyboard command related to Distributed Power, a cabview control is also available. Here's a list of the cabview controls:

```
- ORTS_DP_MOVE_TO_FRONT
- ORTS_DP_MOVE_TO_BACK
- ORTS_DP_IDLE
- ORTS_DP_TRACTION
- ORTS_DP_BRAKE
- ORTS_DP_MORE
- ORTS_DP_LESS
```

Here an example of use of one of the controls:

```
TwoState (
             Type ( ORTS_DP_MOVE_TO_FRONT TWO_STATE )
             Position ( 163.2 378.4 13.75 10 )
             Graphic ( "..\\..\\Common.Cab\\ES44v3\\softkey1trans.ace" )
             NumFrames ( 2 2 1 )
             Style ( WHILE_PRESSED )
             MouseControl ( 1 )
             ORTSDisplay ( 1 )
             ORTSScreenPage ( "2300-0" )
     )
```

14.1.14 EOT (End of Train device)

See *here* for full description of EOT features.

Following EOT controls are available for EOT management:

- ORTS_EOT_BRAKE_PIPE : displays the value of the brake pipe pressure at last wagon. The display is always enabled (even if the EOT is disarmed), because this display could be available also in other ways; however it is possible to mask the display using a texture driven by the EOT state.
- ORTS_EOT_STATE_DISPLAY : may have values from 0 to 5, corresponding to the states listed *here*
- ORTS_EOT_ID : the EOT ID is generated as a 5-digit random number and can be displayed in the cab using this control; entering the ID by the train driver is not supported, as the .cvf files don't support as of now digital data entry
- ORTS_EOT_COMM_TEST : driver command that starts the communication test between locomotive and EOT
- ORTS_EOT_ARM_TWO_WAY : driver command passes the EOT from ArmNow to ArmedTwoWay
- ORTS_EOT_DISARM : passes the EOT to disarmed state
- ORTS_EOT_EMERGENCY_BRAKE (on-off): lets the EOT venting the brake pipe from the last train car.

These controls are available only using the mouse; only The last one can also be operated by the `<Ctrl+Backspace>` key combination.

An example of implementation of the above controls can be seen *in this picture*

The `ORTS_EOT_EMERGENCY_BRAKE` control can be implemented in the cab by an ON-OFF switch.

An example of implementation of the above controls in a .cvf file follows:

```
TwoState (
        Type ( ORTS_EOT_COMM_TEST TWO_STATE )
        Position ( 474 385 16.25 10 )
        Graphic ( "..\\..\\Common.Cab\\ES44v3\\softkey5trans.ace" )
        NumFrames ( 2 2 1 )
        Style ( WHILE_PRESSED )
        MouseControl ( 1 )
        ORTSDisplay ( 0 )
        ORTSScreenPage ( "2100-0" )
)
TwoState (
        Type ( ORTS_EOT_DISARM TWO_STATE )
        Position ( 493 385 16.25 10 )
        Graphic ( "..\\..\\Common.Cab\\ES44v3\\softkey5trans.ace" )
        NumFrames ( 2 2 1 )
        Style ( WHILE_PRESSED )
        MouseControl ( 1 )
        ORTSDisplay ( 0 )
        ORTSScreenPage ( "2100-0" )
)
TwoState (
        Type ( ORTS_EOT_ARM_TWO_WAY TWO_STATE )
        Position ( 511.7 385.7 16.25 10 )
        Graphic ( "..\\..\\Common.Cab\\ES44v3\\softkey7trans.ace" )
        NumFrames ( 2 2 1 )
        Style ( WHILE_PRESSED )
        MouseControl ( 1 )
        ORTSDisplay ( 0 )
        ORTSScreenPage ( "2100-0" )
)
MultiStateDisplay (
        Type ( ORTS_EOT_STATE_DISPLAY MULTI_STATE_DISPLAY )
        Position ( 516 314.5 17 5.15 )
        Graphic ( "..\\..\\Common.Cab\\ES44v3\\CommTest.ace" )
        States ( 2 2 1
                State (
                        Style ( 0 )
                        SwitchVal ( 0 )
                )
                State (
                        Style ( 0 )
                        SwitchVal ( 2 )
                )
        )
        ORTSDisplay ( 0 )
        ORTSScreenPage ( "2100-0" )
)
        Digital (
        Type ( ORTS_EOT_ID DIGITAL )
        Position ( 421 313 22 8 )
        ScaleRange ( 0 999999 )
        Accuracy ( 0 )
        AccuracySwitch ( 0 )
        LeadingZeros ( 0 )
        Justification ( 1 )
        PositiveColour ( 1
                ControlColour ( 255 255 255 )
```

(continues on next page)

(continued from previous page)

```
        )
        NegativeColour ( 1
                ControlColour ( 255 255 0 )
        )
        DecreaseColour ( 0
                ControlColour ( 0 0 0 )
        )
        Units ( KILO_LBS )
        ORTSFont ( 6 0 "Arial" )
        ORTSDisplay ( 0 )
        ORTSScreenPage ( "2100-0" )
)
MultiStateDisplay (
        Type ( ORTS_EOT_STATE_DISPLAY MULTI_STATE_DISPLAY )
        Position ( 513.5 328 22.66 5.15 )
        Graphic ( "..\\..\\Common.Cab\\ES44v3\\EOTStatus2.ace" )
        States ( 4 4 1
                State (
                        Style ( 0 )
                        SwitchVal ( 0 )
                )
                State (
                        Style ( 0 )
                        SwitchVal ( 2 )
                )
                State (
                        Style ( 0 )
                        SwitchVal ( 4 )
                )
                State (
                        Style ( 0 )
                        SwitchVal ( 5 )
                )
        )
        ORTSDisplay ( 0 )
        ORTSScreenPage ( "2100-0" )
)
MultiStateDisplay (
        Type ( ORTS_EOT_STATE_DISPLAY MULTI_STATE_DISPLAY )
        Position (  431.4 292.1 9 5  )
        Graphic ( "..\\..\\Common.Cab\\ES44v3\\MaskEOT.ace" )
        States ( 2 2 1
                State (
                        Style ( 0 )
                        SwitchVal ( 0 )
                )
                State (
                        Style ( 0 )
                        SwitchVal ( 2 )
                )
        )
)
TwoState (
        Type ( ORTS_EOT_EMERGENCY_BRAKE TWO_STATE )
        Position ( 53.5 344.2 21.4 42.8 )
        Graphic ( ..\\..\\Common.Cab\\ES44v3\\EOTEmergency.ace )
```

(continues on next page)

(continued from previous page)

```
        NumFrames ( 2 2 1 )
        Style ( ONOFF )
        MouseControl ( 1 )
)
```

14.1.15 Animated 2D Wipers

This control animates the wipers as seen from a 2D cab. Animation is triggered on/off through key V.

Here is an example of a 2D wipers control block within the .cvf file:

```
ORTSAnimatedDisplay  (
                Type ( ORTS_2DEXTERNALWIPERS MULTI_STATE_DISPLAY )
                Position ( 155 0 331.875 236.25 )
                Graphic ( ..//..//Common.Cab//CabE464_DMI//e464Tergicristallo9.ace )
                ORTSCycleTime ( 1.35 )
                States ( 9 3 3
                        State (
                                Style ( 0 )
                                SwitchVal ( 0 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.11 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.22 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.33 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.44 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.55 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.66 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.77 )
                        )
                        State (
                                Style ( 0 )
                                SwitchVal ( 0.88 )
                        )
                )
        )
```

ORTSCycleTime is expressed in seconds. The .ace file must contain only the frames related to half cycle, that is if e.g. the wiper moves from left to right and back, only the frames related to the motion from left to right have to be included. For the reverse motion the same frames are used from last to first. SwitchVal can vary from 0 to 1.

14.1.16 Control Labels

The string appearing on the screen when the mouse browses over a command control can be customized with following line, to be added within the control block in the .cvf file:

```
ORTSLabel ( "string" )
```

14.1.17 Multiple screen pages on displays

Modern locomotives have one or more displays in their cabs, and often in such displays it is possible to switch among more screen pages. Fields and controls described in this paragraph enable the implementation of .cvf files with such functionality, for both 2D and 3D cabs.

In the .cvf control blocks following further fields may be optionally present:

```
ORTSDisplay ( numeric ), indicating the display ID number (from 0 to 7)
to which the control is linked; if such field is missing, display ID number
zero is assumed;

ORTSScreenPage ( alphanumeric-string ) indicating the screen ID string to
which the control is linked; that means that the control is displayed/may be
operated only if its screen is active in that moment; a missing entry
indicates that the control is displayed independently from the selected screen page;
at game start such controls are enabled, plus the ones with line
ORTSScreenPage ( "default" ); more ORTSScreenPage() entries in a single control
are possible.
```

A new on/off control, called ORTS_SCREEN_SELECT is available, which, in addition to the usual fields and to the optional fields ORTSDisplay and ORTSScreenPage contains one or more of following fields:

```
ORTSNewScreenPage ( alphanumeric-string numeric ): when the control is clicked,
the controls with field ORTSScreenPage equal to the string of this field and
with field ORTSDisplay equal to the numeric will be displayed on such display
in place of the ones displayed up to that moment. if the numeric is missing within
ORTSNewScreenPage, the involved display is the one referenced in field ORTSDisplay
of ORTS_SCREEN_SELECT.
```

A further control is available, named ORTS_STATIC_DISPLAY, which is specially devoted to the loading of the background of screen pages (their static part). Here is an example of usage of it:

```
MultiStateDisplay (
        Type ( ORTS_STATIC_DISPLAY MULTI_STATE_DISPLAY )
        Position ( 246 151 105 16 )
        Graphic ( sempronioStatic.ace )
        States ( 1 1 1
                State (
                        Style ( 0 )
                        SwitchVal ( 0 )
                )
        )
                ORTSScreenPage ( "sempronio" )
        )
```

With this block, the static part of the "sempronio" screen page is loaded on display 0 when such screen becomes the active one.

.cvf files not using fields and controls listed in this paragraph work as usual, with no changes needed.

14.1.18 Further OR cab controls

OR supports the cabview control to open/close the left doors, the right doors and the mirrors.

The control blocks are like the one shown for the cab light. The Type strings are ORTS_LEFTDOOR, ORTS_RIGHTDOOR and ORTS_MIRRORS.

14.1.19 Cab controls for generic items

OR supports the cabview controls for two generic two-state items. The cabview controls aree called <ORTS_GENERIC_ITEM1> and <ORTS_GENERIC_ITEM2>. Their state can be toggled also by respectively clicking keys <Shift+.> and <Shift+,>.

Sound events are associated, that is:

```
240: GenericItem1On
241: GenericItem1Off
242: GenericItem2On
243: GenericItem2Off
```

Animations within the .s file of the locomotive, either stopped/moving or two-state can be associated to the item state. Linked stopped/moving (wiper type) animations are named <ORTSITEM1CONTINUOUS> and <ORTSITEM2CONTINUOUS>. Linked two-state animations (doors type) are named <ORTSITEM1TWOSTATE> and <ORTSITEM2TWOSTATE>. The default animation speed for stopped/moving type animations is 8 FPS. It may be modified with following parameter in the .sd file:

```
ESD_CustomAnimationSpeed ( 8 )
```

Examples of use are fan control, open/close of aerodynamic coverages of couplers in high speed trains, menu pages switching.

Animations within the 3D cab .s file are also available, as follows:

```
ORTS_ITEM1CONTINUOUS
ORTS_ITEM2CONTINUOUS
ORTS_ITEM1TWOSTATE
ORTS_ITEM2TWOSTATE
```

in analogy to the four animations for the locomotive .s file.

14.1.20 High-resolution Cab Backgrounds and Controls

In MSTS the resolution of the cab background image is limited to 1024x1024; this limitation does not apply in OR as a result of OR's better handling of large textures.

2D cab backgrounds can reach at least to 3072x3072; however very fine results can be obtained with a resolution of 2560x1600. The image does not have to be square.

2D cab animations have also been greatly improved; you are reminded here that there are two types of animated rotary gauges, i.e. normal gauges and general animations using multiple frames. In this second case in MSTS all of the frames had to be present in a single texture with a max resolution of 640x480. In OR these frames can be as large as desired and OR will scale them to the correct size. In general it is not necessary to use a resolution greater than 200x200 for every frame.

The syntax to be used in the .cvf file is the standard one as defined by MSTS.

To clarify this, the position parameters of a sample needle block are described here.

In the `Position` statement, the first 2 numbers are the position of the top left-hand side of the needle texture in cabview units with the needle in the vertical position. In the `Dial` type the last 2 numbers are the size of the needle texture. The last number (50 in the example) controls the scaling of the needle texture, i.e. changing this changes the size of the needle that OR displays.

```
Dial (
    Type ( SPEEDOMETER DIAL )
    Position ( 549 156 10 50 )
    Graphic ( Speed_recorder_needle_2.01.ace )
    Style ( NEEDLE )
    ScaleRange ( 0 140 )
    ScalePos ( 243 115 )
    Units ( KM_PER_HOUR )
    Pivot ( 38 )
    DirIncrease ( 0 )
)
```

Next is an example of a control animation, this one is a simple 3 frame animation. The examples shown in the following images are the two rotary switches to the right of the two lower brake gauges, both being 3 position. (The left most switch is for the headlights). For these animations the graphic was done at 1600x1600; when each frame was finished it was scaled down to 200x200 and placed into the animation texture. Note the extreme sharpness of these controls in the inset image.

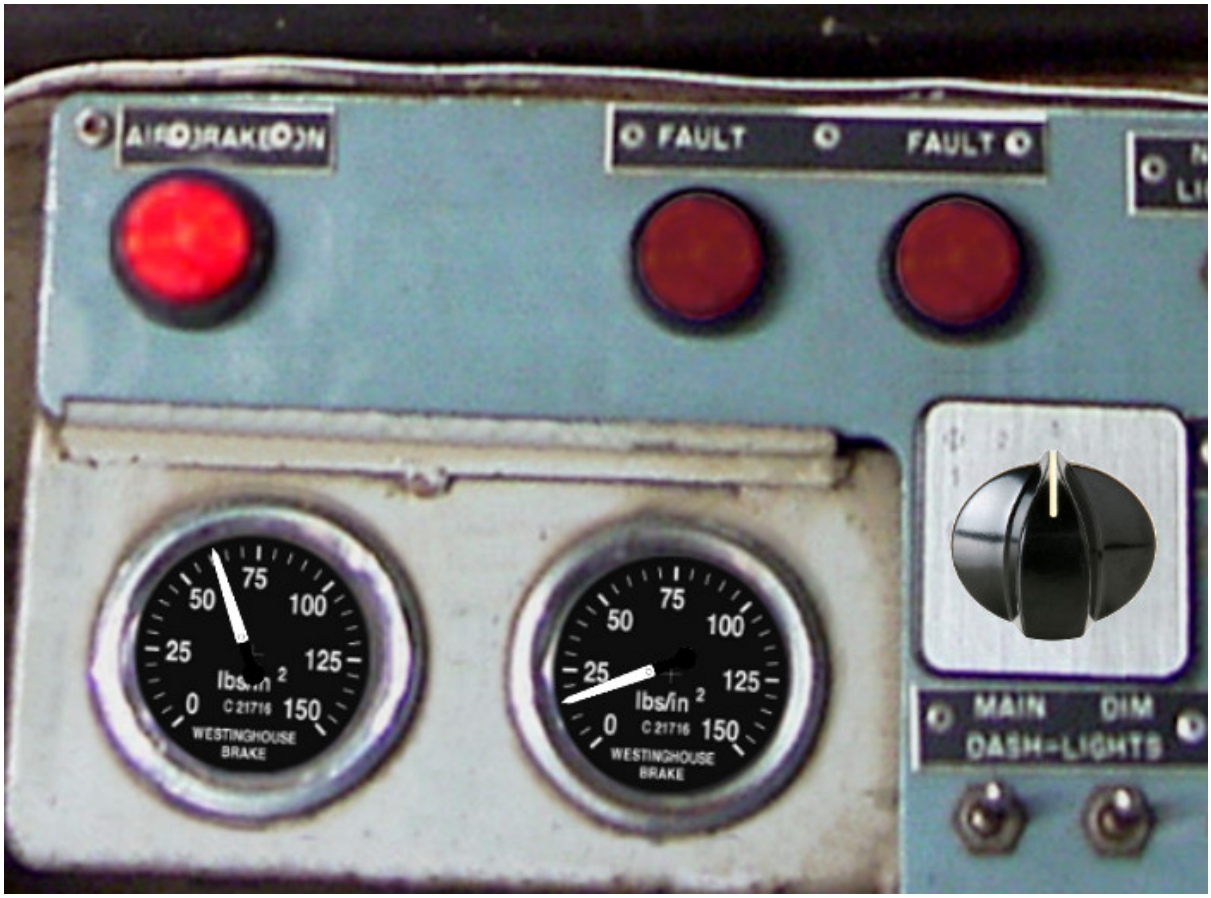
Adding a slight amount of 2x2 pixel blur helps the animation blend into the background better (this has been done to the gauge needles).

Below is the appropriate part of the CVF. The scaling is controlled by the last two digits of the `Position` statement:

```
TriState (
    Type ( DIRECTION TRI_STATE )
    Position ( 445 397 35 35 )
    Graphic ( Switch_nob_3.0_Transmission.ace )
    NumFrames ( 3 3 1 )
    Style ( NONE )
    MouseControl ( 1 )
    Orientation ( 0 )
    DirIncrease ( 0 )
)
```

Note that the “Airbrake On” light (on the panel upper left) has also been animated. This is a simple 2 frame animation.





Shown above are two pictures of one hi-res 2D cabview, one showing the whole cab, and the other one showing the detail of some controls. In this example the cab background image used was cut down to 2560x1600. The texture for the Speed Recorder needle is 183x39 and for the brake gauge needles is 181x29, Note the odd number for the width. This is required as OR (and MSTS) assume the needle is in the center of the image. The Reversing and Headlight switch animation frames are 116x116.

There are as yet no specific tools to create these cabviews; a standard image manipulation program to do all textures is required, and to create any new items, e.g. the gauge faces, a standard drawing program can be used. To actual set up the cabview and to position the animations the .cvf file is modified with a standard text editor, and OR is used as a viewer, using a straight section of track on a quick loading route. Through successive iterations one arrives quite quickly at a satisfactory result.

14.1.21 Configurable Fonts

OR supports a configurable font family, with font size selection, and a choice of regular or bold style. More than one font or size can be used in the same cabview. This does not affect the display in MSTS.

An optional line of the form `ORTSfont ( fontsize fontstyle "fontfamily" )` must be inserted into the .cvf block of the digital control or digital clock, where *fontsize* is a float (default value 10), *fontstyle* an integer having the value 0 (default) for regular and 1 for bold, and *fontfamily* is a string with the font family name (ex. "Times New Roman"). The default is "Courier New". A convenient font, if available, is "Quartz MS" or "Quartz", which models a 7-segment display.

Here is an example that displays the digital clock with a 12 pt. bold font using the Sans Serif font family:

```
DigitalClock (
    Type ( CLOCK DIGITAL_CLOCK )
    Position ( 40 350 56 11 )
    Style ( 12HOUR )
    Accuracy ( 1 )
    ControlColour ( 255 255 255 )
    ORTSFont ( 12 1 "Sans Serif" )
)
```

It is acceptable if only the first parameter of ORTSFont is present, or only the first two, or all three. Note that you cannot use the MS Cabview editor on the .cvf file after having inserted these optional lines, because the editor will delete these added lines when the file is saved.

14.1.22 Rotation of Gauges and Digital controls

One of the drawbacks of rendering a cabview in 2D is that some parts of it are not shown with a frontal, precisely vertical or horizontal, view. Displaying a vertical gauge or a horizontal digital control on it generates an unrealistic effect. This is the rationale of following entry, to be added within a Gauge or Digital cabview control block in the .cvf file:

```
ORTSAngle ( 5 )
```

The number in parenthesis is the angle in degrees with respect to the horizontal (or to the vertical for vertical gauges). Positive values produce counterclockwise rotation.

At the left of the picture an example of a white vertical gauge that has been rotated by 12 degrees



Here an example of a red max speed indication that has been rotated by 5 degrees



Gauges may have Style POINTER or SOLID.

Rotation may be applied, with the same syntax, also to DigitalClock cab controls.

14.2 3D cabs

If the locomotive has a 3D cab, it will be selected by default by the simulator. You can press key <1> to enter the cab. In case locomotive has both 2D and 3D cabs provided, the key <Alt+1> can be used in order to switch between 2D and 3D cabs.

14.2.1 Development Rules

- The 3D cab is described by an .s file, the associated .ace or .dds files, and a .cvf file having the same name as the .s file. All these files reside in a folder named CABVIEW3D created within the main folder of the locomotive.
- If the .cvf file cannot be found in the CABVIEW3D folder, the 3D cab is associated with the .cvf file of the 2D cab.
- Instruments are named with the same conventions as 2D cabs, i.e. FRONT_HLIGHT, SPEEDOMETER, etc.
- A cab can have multiple instances of the same instruments, for example multiple clocks or speedometers.
- Instruments are sorted based on the order of their appearance in the .cvf file, for example SPEEDOMETER:0 corresponds to the first speedometer in the .cvf file, SPEEDOMETER:1 corresponds to the second one.
- An instrument can have multiple subgroups to make the animation realistic, for example, TRAIN_BRAKE:0:0 and TRAIN_BRAKE:0:1 belong to the instrument TRAIN_BRAKE:0. However, if the instrument is a digital device, the second number will be used to indicate the font size used, for example SPEEDOMETER:1:14 means the second speedometer (which is digital as defined in .cvf) will be rendered with 14pt font. This may be changed in future OR releases. The important information for a digital device is its location, thus it can be defined as an object with a small single face in the 3D model.
- Animation ranges must be in agreement with the .cvf file
- Within the Wagon section of the .eng file a block like the following one has to be generated:

```
ORTS3DCab(
    ORTS3DCabFile ( Cab.s )
    ORTS3DCabHeadPos ( -0.9 2.4 5.2 )
    RotationLimit ( 40 60 0 )
    StartDirection ( 12 0 0 )
)
```

- It is also possible to animate the wipers, by inserting into the .s file an animation named EXTERNALWIPERS:0:0
- Gauges of solid type have to be named AMMETER:1:10:100; where the three numbers indicate that this is the second ammeter, that it has a width 10 mm, and a maximum length of 100 mm. The color and direction/orientation follow those defined in .cvf files.
- Digits for 3D cabs can now use custom ACE files; e.g. name the part as CLOCK:1:15:CLOCKS. This will draw the second clock with 15mm font dimension, with the CLOCKS.ACE file in CABVIEW3D containing the font. If no ace is specified, the default will be used.
- Mirrors and doors can be operated from 3D cabs. The names used are LEFTDOOR, RIGHTDOOR and MIRRORS.
- like the 2D cabs, also 3D cabs can have a night version. Night textures, named like the corresponding day textures, must be located within a NIGHT subfolder of the CABVIEW3D folder. To enable night cabs an .sd file with the same name as the shape file of the 3D cab must be present in the CABVIEW3D folder. This .sd file has a standard format and must contain following line:

```
ESD_Alternative_Texture ( 256 )
```

- How to control the view in a 3D cab is described *here*.

A demo trainset with a 3Dcab, that may be useful for developers, can be downloaded from: http://www.tsimserver.com/Download/Df11G3DCab.zip

14.2.2 A Practical Development Example For a Digital Speedometer

Let's suppose you wish to create a digital speedometer using a size 14 font.

To explain it in *gmax* language, you must have an object called `SPEEDOMETER` in the cab view and it must be comprised of at least one face.

As the sample cab has only one digital speedometer, it can be named `SPEEDOMETER_0_14`.

The number 0 indicates that this is the first speedometer gauge in the cab and the number 14 indicates the size of the font to display. Note that an underscore is used to separate the numbers as the LOD export tool does not support the use of colons in object names when exporting. More on this later.

The speed does not display where the face for the `SPEEDOMETER` object is located but where the *pivot point* for the `SPEEDOMETER` object is located. Normally you would place the `SPEEDOMETER` object somewhere in the cab where it will not be seen.

With the `SPEEDOMETER_0_14` object selected in gmax, go to the *Hierarchy* tab, select *Affect Pivot Only* and click *Align to World* to reset the orientation to world coordinates. Then use the *Select and Move* tool to move the pivot to where in the cab you wish the numerals to appear. As you have aligned the pivot point to World coordinates the numerals will display vertically. As most locomotive primary displays are normally angled you may have to rotate the pivot point so that it aligns with the angle of the *display screen*.

Export the .S file for the cab as usually.

You will then have to uncompress the .s file for the cab using Shape File Manager or the .S file decompression tool of your choice.

Then open the .S file with a text editor and search for the letters "speed" until you find the first instance of `SPEEDOMETER_0_14` and change it to be `SPEEDOMETER:0:14`. Search again and find the second instance of `SPEEDOMETER_0_14` and change that also to `SPEEDOMETER:0:14`. Save the .S file in the text editor.

Now just one more thing. Download the `DF11G3DCab` demo trainset. In the `CABVIEW3D` folder of that download you will find an ace file called `SPEED.ACE`. Copy that file and paste it into the `CABVIEW3D` folder for your model.

Now, open OR and test your speedometer.

14.2.3 FUEL_GAUGE for steam locomotives

The FUEL_GAUGE dial is available also for steam locomotives. It may be used both to display a fuel level for oil burning steam locomotives (also in 2D cabs), and to animate the coal level in a tank loco. Default unit of measure is Kg; alternate unit of measure may be LBS. Here below is an example of an entry for a 3D cab:

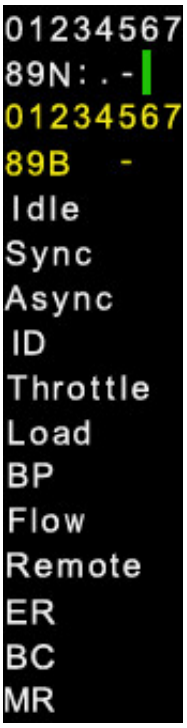
```
Dial (
Type ( FUEL_GAUGE DIAL )
Style ( POINTER )
ScaleRange ( 0 5000 )
Units ( LBS )
)
```

14.2.4 Distributed power display

Following info applies to the creation of a distributed power display in 2D cabs, in addition to what is described *here* for 2D cabs.

In the 3Dcab .s file an ORTS_DISTRIBUTED_POWER object must be defined, with the same syntax rules of the digitals, so e.g. ORTS_DISTRIBUTED_POWER:0:8:DPI , where 8 is the selected character font size and DPI is the DPI.ace texture associated.

In the folder where the 3D cab files are located (usually CABVIEW3D) such file DPI.ace must be present. A sample file for that can be found in `Documentation\SampleFiles\Manual\DPI.zip` . Here is how such file looks like



Customizations for such file are possible following these rules:

1. Horizontal/vertical ratio must be kept
2. The first four lines must have the characters centered in their rectangle.
3. From the 5th line on characters may be also spaced in a thicker way (as is for the `Idle` string in the above picture)
4. From the 5th line on strings may be replaced bo strings in other national languages, provided that the new strings aren't wider than the original ones.
5. It should be possible to have a transparent background if preferred.

Except for the first column, fields in the 3D distributed power display are always with center alignment.

14.2.5 Alignment for digital controls

For backwards compatibility reasons, `Justification ( 1 )`, `Justification ( 2 )` and `Justification ( 3 )` all lead to a left alignment of the digital in 3Dcabs.

`Justification ( 5 )` must be used for center alignment, and `Justification ( 6 )` must be used for right alignment. `Justification ( 4 )` leads to left alignment.

CHAPTER 15

OR-Specific Route Features

As a general rule and as already stated, Open Rails provides all route functionalities that were already available for MSTS, plus some opportunities such as also accepting textures in .dds format.

15.1 Modifications to .trk Files

Many of the features described in this chapter require additional parameters to be added in the route's .trk file. The additional parameters can be directly added at the end (just above the last parenthesis) of the route's .trk file residing in the route's root folder. Don't add such parameters in other positions of the file, because this would create problems if you want to use the MSTS editors with the related route. However, to avoid modifying the original file, the *Include* method described *here* is also applicable to the .trk file, creating a new .trk file inserted into an `OpenRails` folder in the root folder of the route. As an example, in case of the parameter needed to avoid forest trees on tracks (see *here*), this additional .trk file will contain:

```
include ( ../Surfliner2.trk )
    ORTSUserPreferenceForestClearDistance ( 2 )
```

Only OR will look in the `Openrails` folder.

For a full list of parameters, see *Developing OR Content - Parameters and Tokens*

15.2 Repetition of Snow Terrain Textures

OR provides a simple way to add snow terrain textures: the following default snow texture names are recognized: `ORTSDefaultSnow.ace` and `ORTSDefaultDMSnow.ace`, to be positioned within folder `TERRTEX\SNOW` of the concerned route. For the snow textures that are missing in the `SNOW` subfolder, and only for them, ORTS uses such files to display snow, if they are present, instead of using file `blank.bmp`.

To have a minimum working snow texture set, the file `microtex.ace` must also be present in the `SNOW` subfolder.

15.3 Snow Textures with Night Textures

MSTS did not allow snow textures to be used with night textures. This meant having dark buildings when running an activity at night when the weather is set to snow. It turns out that OR is able to run with snow textures and night textures. To do this, you have to create the `Night\` folder in the `Textures\Snow\` directory and copy the needed textures into the `Night\` folder. Doing this will allow night textures to be visible when operating in snow at night. Keep in mind that the current night textures such as buildings do not include snow so new textures will have to be created.

One warning, if you decide to do this, there is a possiblility of experiencing resource issues.

15.4 Operating Turntables and Transfertables

A cool feature available in OR is the one of operating turntables and transfertables. In MSTS they are static, and can't rotate and transfer trainsets. Turntables and transfertables are managed in a similar way by OR, and share also a significant portion of code. Therefore here reference to turntables will be made, and then only the differences for transfertables will be described.

Caution: Turntables and transfertables can't be directly connected to a leading switch. A track section of at least 1 meter must be laid in between.

15.4.1 Turntables

The best way to get a turntable to be operational is to refer to an example. So here are the instructions and the files to test this function, both for route Catania-Messina (SICILIA 1) and for other routes using `a1t27mturntable.s`. Route Catania-Messina can be downloaded from here . A .ws file within the World subdirectory must be replaced with file `w-005631+014158.zip` available in the Open Rails pack in the `Documentation\SampleFiles\Manual` subfolder. (this has nothing to do with turntables, it's a file that contains incoherent data that can cause a crash). Pls. note that also the other sample files cited in this paragraph are available in such subfolder.

Two test paths, included in file `Turntable_PATHS.zip`, one for each turntable in the route, which can be used either in explore mode or within activities are available in the Open Rails pack. Within the route's folder an OpenRails subfolder must be created, that must contain 2 files. The first one is following file `turntables.dat`, which contains the data needed to OR to locate and specify the turntable.

turntables.dat:

```
2
Turntable(
WFile ( "w-005625+014198.w" )
UiD ( 1280 )
XOffset ( 0 )
ZOffset ( 13.4 )
TrackShapeIndex ( 253 )
Animation ( "TRACKPIECE" )
Diameter ( 27 )
)
Turntable(
WFile ( "w-005631+014158.w" )
UiD ( 638 )
XOffset ( 0 )
ZOffset ( 13.4 )
TrackShapeIndex ( 253 )
```

(continues on next page)

(continued from previous page)

```
Animation ( "TRACKPIECE" )
Diameter ( 27 )
)
```

To generate this file for other routes following has to be taken into account:

- the first line must be blank
- the number in the second line (2 in the above file) is the number of operating turntables within the route
- WFile is the name of the .w file where the turntable is present
- The number in the UiD line is the UiD number of the TrackObj () block within the .w file related to the turntable
- XOffset, YOffset and ZOffset are the offsets of the center of rotation of the turntable with respect to the zero of the turntable shape
- TrackShapeIndex is the index of the TrackShape () block within tsection.dat that refers to the turntable; please note that if a new TrackShape () block for the turntable is needed, it is not necessary to modify tsection.dat; it is possible to proceed as described *here*
- The Animation parameter is the name of the Matrix of the rotating part within the .s file
- the Diameter value is the diameter of the turntable in meters.

The above file refers to turntables using the a1t27mturntable.s shape.

The second file to be inserted within the route's Openrails subfolder is a small integration .trk file that indicates the name of the .sms sound file to be associated to the turntable. For the route SICILIA 1 such file is therefore named `SICILIA 1.trk`, like its parent file. Here is the file content.

SICILIA 1.trk:

```
include ( "../Sicilia 1.trk" )
                    ORTSDefaultTurntableSMS ( turntable.sms )
```

The first line must be empty.

File `a1t27mturntable.s` must be modified to add the animation data, as MSTS has provided it as a static file. To do this, uncompress it with Route Riter or Shapefilemanager and insert just above the last parenthesis the contents of file `a1t27mturntable_animations.zip`. If other .s files have to be used for turntables, or new ones have to be developed, it must be considered that the rotation animation should be as follows:

```
animation ( 3599 30
        anim_nodes ( ..
                ..
                ..
                ..
                anim_node TRACKPIECE (
                        controllers ( ..
                                tcb_rot ( 5
                                        tcb_key ( 0 0 0 0 1 0 0 0 0 0 )
                                        tcb_key ( 900 0 0.7071068 0 0.7071067 0 0 0 0 0 )
                                        tcb_key ( 1800 0 1 0 0.0 0 0 0 0 0 )
                                        tcb_key ( 2700 0 -0.7071068 0 0.7071067 0 0 0 0 0␣
↪)
                                        tcb_key ( 3600 0 0 0 -1 0 0 0 0 0 )
                                )
```

or as follows:

```
animation ( 3599 30
        anim_nodes ( ..
                ..
                ..
                ..
anim_node WHEEL1 (
    controllers ( 1
       tcb_rot ( 5
          slerp_rot ( 0 0 0 0 1 )
          slerp_rot ( 900 0 0.7071068 0 0.7071067 )
          slerp_rot ( 1800 0 1 0 -1.629207E-07 )
          slerp_rot ( 2700 0 -0.7071066 0 0.7071069 )
          slerp_rot ( 3600 0 0 0 1 )
        )
     )
 )
```

The above names of the anim_nodes are of course free choice. The animation rotation direction as defined above must be counterclockwise.

Within the base Sound folder (not the one of the route) the .sms file `turntablesSOUND.zip` has to be added to provide sound when the turntable rotates. It uses the two default MSTS .wav files for the sound. They have a bit a low volume. It is open to everyone to improve such files. Discrete trigger 1 is triggered when the turntable starts turning empty, discrete trigger 2 is triggered when the turntable starts turning with train on board, and discrete trigger 3 is triggered when rotation stops.

To help generating the tsection.dat entries for new turntable types a rough `.xls` spreadsheet (turntable_sectionidxs.xls) can be found in `Documentation\SampleFiles\Manual`. It computes the X, Z and degree parameters to be inserted in the SectionIdx lines of the TrackShape block within the tsection.dat file. You only have to insert the diameter of the turntable and the degree step. Of course you have to take only the lines up to the one preceding the one with degrees = 180.

Already many existing turntables have been successfully animated and many new other have been created. More can be read in this forum thread .

15.4.2 Transfertables

Info for transfertables is stored in file `turntables.dat` too. This file may contain info for transfertables and turntables together. Here is an example of such file for a turntable and a transfertable:

```
2
Turntable(
WFile ( "w-005625+014198.w" )
UiD ( 1280 )
XOffset ( 0 )
ZOffset ( 13.4 )
TrackShapeIndex ( 253 )
Animation ( "TRACKPIECE" )
Diameter ( 27 )
)
Transfertable(
WFile ( "w-005578+014976.w" )
UiD ( 72 )
XOffset ( 0 )
ZOffset ( 15.0)
TrackShapeIndex ( 37300 )
Animation ( "TRACKPIECE" )
```

(continues on next page)

(continued from previous page)

```
Length ( 29.4 )
)
```

Parameters have the same meaning as for turntables. "Length" is the length of the transfer bridge (therefore the length of the track above it or a bit less, depending from the dimensions of the basin of the transfertable).

The integration .trk file format described in preceding paragraph can be used also for transfertables, using the same sound.

In the standard `tsection.dat` there are no usable transfertables defined. Therefore at least a new TrackShape block has to be created. Also in this case it is suggested to define the additional block in the route's specific `tsection.dat`.

Here below is an example for a route's specific `tsection.dat` containing a TrackShape for a transfertable:

```
include ( "../../../Global/tsection.dat" )
_INFO ( Track section and shape addition for transfer table derived from turntable 27m )
TrackSections ( 40000
_SKIP ( No change here )
)
TrackShapes ( 40000
_INFO(TrackShape for for 30 m transfer table derived from turntable 27m)
TrackShape ( 37300
FileName (  A1t30mTransfertable.s )
NumPaths ( 9 )
SectionIdx ( 1 0 -0.18 -1.1 0 339 )
SectionIdx ( 1 4.985 -0.18 -1.1 0 339 )
SectionIdx ( 1 9.97 -0.18 -1.1 0 339 )
SectionIdx ( 1 14.955 -0.18 -1.1 0 339 )
SectionIdx ( 1 19.94 -0.18 -1.1 0 339 )
SectionIdx ( 1 24.925 -0.18 -1.1 0 339 )
SectionIdx ( 1 29.91 -0.18 -1.1 0 339 )
SectionIdx ( 1 34.895 -0.18 -1.1 0 339 )
SectionIdx ( 1 39.88 -0.18 -1.1 0 339 )
)
)
```

The first line must be empty.

The animation block for the above transfertable is as follows:

```
animations ( 1
        animation ( 3600 30
                anim_nodes ( 2
                        anim_node BASIN (
                                controllers ( 0 )
                        )
                        anim_node TRACKPIECE (
                                controllers ( 1
                                        linear_pos ( 2
      linear_key (      0       0       -1.92177        0       )
      linear_key (      3600    39.88   -1.92177        0       )
                                )
                                )
                        )
                )
        )
)
```

3600 is not a mandatory value, however to have a reasonable transfer speed a number of animation keys equal to 60 - 90 every meter should be selected.

15.4.3 Locomotive and wagon elevators

The elevator is managed by ORTS as a vertically moving transfertable. So files needed are the same as used for a transfertable, with content modified where needed.

Info to identify an elevator in a route is stored in file turntables.dat, as it is for turntables and transfertables. The same file can store info for moving tables of different types. Here a turntables.dat file that contains info for an elevator:

```
1
Transfertable(
WFile ( "w-005578+014976.w" )
UiD ( 75 )
XOffset ( 0 )
YOffset ( -0.18 )
ZOffset ( 13.405)
VerticalTransfer ( 1 )
TrackShapeIndex ( 37301 )
Animation ( "TRACKPIECE" )
Length ( 26.81 )
)
```

What identifies this as an elevator is the presence of the VerticalTransfer parameter with value 1. The other difference to a transfertable is the presence of the YOffset parameter, which is the vertical offset of the zero position of the elevator with respect to the shape file zero.

An example of the animation block in the elevator shape file is shown here below:

```
animations ( 1
        animation ( 1800 30
                anim_nodes ( 2
                        anim_node BASIN (
                                controllers ( 0 )
                        )
                        anim_node TRACKPIECE (
                                controllers ( 1
                                        linear_pos ( 2
      linear_key (      0       0       -1.92177        0       )
      linear_key (      1800    0       6.07823 0        )
                                        )
                                )
                        )
                )
       )
)
```

wich generates a vertical movement with a span of 8 meters which is covered in 60 seconds. Of course the 1800 value may be modified to get the desired motion speed.

The elevator must also be defined as a TrackShape in tsection.dat. It is suggested to define it in a route specific `tsection.dat` extension file, which, for the sample elevator, is as follows:

```
include ( "../../../Global/tsection.dat" )
_INFO ( Track section and shape addition for transfer table derived from turntable 27m )

TrackSections ( 40000
```

(continues on next page)

(continued from previous page)

```
_SKIP ( No change here )

)


TrackShapes ( 40000

_INFO(TrackShape for for vertical transfer table derived from turntable 27m)

TrackShape ( 37301
 FileName ( A1t27mVerticalTransfertable.s )
  NumPaths ( 2 )
  SectionIdx ( 1 0 -0.18 0.0000 0 338 )
  SectionIdx ( 1 0 7.82 0.0000 0 338 )
 )
)
```

To insert the elevator in a route using TSRE5 it must be reminded that the latter doesn't look at the tsection.dat file within the Openrails subfolder. So, for the sole time of the editing of the route, the TrackShape() block must be inserted in the global tsection.dat. After route editing is terminated, the block may be removed. Tsection.dat build 38 or higher is required within the main Global folder.

At runtime the elevator is moved with the keys used for transfertables and turntables. Alt-C moves the elevator upwards, while Ctrl-C moves the elevator downwards.

15.4.4 Path laying and operation considerations

By building up a path that enters the turntable or transfertable, exits it from the opposite side and has a reversal point few meters after the end of the turntable or transfertable, it is possible to use the turntable or transfertable in activity mode. The player will drive the consist into the turntable or transfertable and stop it. At that point the reversal point will have effect and will logically lay the consist in the return sub-path. The player will put the consist in manual mode, rotate the turntable (in case he is using a turntable) by 180 degrees and return to auto mode. At this point the consist will be again on the activity path.

If instead the player wants the consist to exit to other tracks, he must drive the consist in manual mode out of the turntable or transfertable. If he later wants to drive back the consist into the turntable or transfertable and rotate or translate the train so that it exits the turntable or transfertable on the track where it initially entered it, he can pass back the train to auto mode after rotation, provided the path is built as defined above.

By using the feature to change *player train* it is possible also to move in and out any locomotive on any track of e.g. a roundhouse or use a shunter to shunt a wagon in and out of a trasfertable.

15.5 .w File modifiers

An `Openrails` subfolder can be created within the route's `World` folder. Within this subfolder .w file chunks can be positioned. ORTS will first read the base .w files, and then will correct such files with the file chunks of the `Openrails` subfolder. This can be used both to modify parameters or to add OR-specific parameters. Here an example of a w. file chunk for USA1 .w file w-011008+014318.w:

```
SIMISA@@@@@@@@@@@JINX0w0t______

Tr_Worldfile (
              CarSpawner (
```

(continues on next page)

(continued from previous page)

```
                        UiD ( 532 )
                        ORTSListName ( "List2" )
                )
                CarSpawner (
                        UiD ( 533 )
                        ORTSListName ( "List3" )
                )
                Static (
                        UiD ( 296 )
                        FileName ( hut3.s )
        )
)
```

With the two CarSpawner block chunks OR interprets the CarSpawners with same UiD present in the base .w file as extended ones (see *here*). With the Static block OR replaces the shape defined in the Static block with same UiD within the base .w file with the one defined in the file chunk. WAny Pickup, Transfer, Forest, Signal, Speedpost, LevelCrossing, Hazard, CarSpawner, Static, Gantry may have parameters modified or added by the "modifying" .w file.

Caution: If the route is edited with a route editor, UiDs could change and so the .w file chunks could be out of date and should be modified.

Caution: Entering wrong data in the .w file chunks may lead to program malfunctions.

15.6 Multiple car spawner lists

With this OR-specific feature it is possible to associate any car spawner to one of additional car lists, therefore allowing e.g. to have different vehicles appearing in a highway and in a small country road.

The additional car lists have to be defined within a file named carspawn.dat to be inserted in an `Openrails` subfolder within the Route's root folder. Such file must have the structure as in following example:

```
SIMISA@@@@@@@@@@JINX0v1t______

3
CarSpawnerList(
ListName ( "List1" )
2
CarSpawnerItem( "car1.s" 4 )
CarSpawnerItem( "postbus.s" 4 )
)
CarSpawnerList(
ListName ( "List2" )
3
CarSpawnerItem( "policePHIL.S" 6 )
CarSpawnerItem( "truck1.s" 13 )
CarSpawnerItem( "postbus.s" 6 )
)
CarSpawnerList(
ListName ( "List3" )
2
CarSpawnerItem( "US2Pickup.s" 6 )
```

(continues on next page)

(continued from previous page)

```
CarSpawnerItem( "postbus.s" 13 )
)
```

The first 3 defines the number of the additional car spawner lists. To associate a CarSpawner block to one of these lists, a line like this one:

```
ORTSListName ( "List2" )
```

has to be inserted in the CarSpawn block, in any position after the UiD line.

If the CarSpawner block does not contain such additional line, it will be associated with the base carspawn.dat file present in the route's root directory.

Caution: If the route is edited with the MSTS route editor modifying the .w files referring to the additional car spawners, the above line will be deleted.

To avoid this problem, two other possibilities are available to insert the additional line. One is described *here*. The other one is to use the OR specific TSRE route editor, that natively manages this feature. Also in the latter case, however, if the route is later edited with the MSTS route editor, the above line will be deleted.

15.7 Car spawners used for walking people

The OR specific TSRE route editor is able to generate car spawner paths also outside roads. This has many applications, one of which is to generate paths for walking people. Walking people have the peculiarity that on an inclined path they don't incline like a vehicle does, instead they remain vertical. To enable OR to handle these car (or better person) spawners specifically, the parameter IgnoreXRotation () has to be inserted in the car spawner list, just after the number of the car spawner items.



Here is an example of a car spawner file specific for walking people to be inserted in the route's Openrails subfolder (see *here*):

```
SIMISA@@@@@@@@@@JINX0v1t______

1
CarSpawnerList(
ListName ( "People1" )
3
IgnoreXRotation ()
CarSpawnerItem( "walkingperson1.s" 3 )
CarSpawnerItem( "walkingperson2.s" 1 )
CarSpawnerItem( "walkingperson3.s" 1 )
)
```

15.8 Route specific TrackSections and TrackShapes

It quite often occurs that for special routes also special TrackSections and TrackShapes are needed. Being file tsection.dat unique for every installation, for such routes a so-called mini-route installation was needed. The present feature overcomes this problem. The route still uses the common tsection.dat,but it can add to it route-specific TrackSections and TrackShapes, and can modify common ones. This occurs by putting in an `OpenRails` subfolder within the route's root folder a route-specific chunk of tsection.dat, which includes the TrackSections and TrackShapes to be added or modified. Here a fictitious example for route USA1 (first line must be blank):

```
include ( "../../../Global/tsection.dat" )
_INFO ( Track sections and shapes specific for USA1   )
_Skip (
Further comments here
)
TrackSections ( 40000
_Skip (
Comment here
)
_SKIP ( Bernina )
  TrackSection ( 33080
          SectionSize ( 0.9 1.5825815 )
  )
  TrackSection ( 19950
          SectionSize ( 0.9 12 )
  )
)
TrackShapes ( 40000
_Skip (
Comment here
)
-INFO(Bernina Pass narrow gauge sections / wood tie texture)
_INFO(by Massimo Calvi)
_INFO(straight sections)
  TrackShape ( 30000
          FileName ( track1_6m_wt.s )
          NumPaths ( 1 )
          SectionIdx ( 1 0 0 0 0 33080 )
  )
  TrackShape ( 19858
          FileName ( track12m_wt.s )
          NumPaths ( 1 )
          SectionIdx ( 1 0 0 0 0 19950 )
```

(continues on next page)

(continued from previous page)

```
  )
)
```

In this fictitious example the first TrackSection and TrackShape is present also in the Global tsection.dat, so the effect is that the original TrackSection and TrackShape are modified; the second ones are not present, and so they are added to the lists.

Note: To be able to use these modified items with the actual MSTS RE and AE it is necessary that these modified items are present also in the original tsection.dat file. However, when the work with the RE is terminated and route is distributed, it is sufficient to distribute the above route's specific tsection.dat.

15.9 Overhead wire extensions

15.9.1 Double wire

OR provides an *experimental function that enables the upper wire* for electrified routes. The optional parameter `ortsdoublewireenabled` in the `.trk` file of the route can force the activation or deactivation of the option overriding the user setting in the options panel.

In this example the upper wire is enabled overriding the user setting:

```
OrtsDoubleWireEnabled ( On )
```

while in this one the upper wire is forced to be disabled:

```
OrtsDoubleWireEnabled ( Off )
```

Another parameter (`ortsdoublewireheight`) specifies the height of the upper wire relative to the contact wire; if not specified the default is 1 meter. In this example the upper wire is 130cm above the main wire (as in most Italian routes):

```
include ( "../tures.trk" )
  OrtsTriphaseEnabled ( Off )
  OrtsDoubleWireEnabled ( On )
  OrtsDoubleWireHeight ( 130cm )
```

Of course you can use any *distance unit of measure* supported by OR.

15.9.2 Triphase lines

The modern electric locos are powered by DC or monophase AC, but some years ago there were triphase AC powered locos. A triphase circuit needs three wires (one for each phase, no wire is needed for neutral); in rail systems two wires are overhead and the third is made by the rails.

OR can enable the second overhead wire with the parameter `ortstriphaseenabled` this way:

```
OrtsTriphaseEnabled ( On )
```

If the parameter is missing or its value is `Off` the usual single wire is displayed.

Another parameter (`ortstriphasewidth`) specifies the space between the two wires with a default (if the parameter is not declared) of 1 meter.

15.10 Loading screen

In the `.trk` file of the route the parameter `loadingscreen` can be used as in this example:

```
LoadingScreen ( Load.ace )
```

If in the main directory of the route there is a file with the same name but with extension `.dds` and the DDS texture support is enabled the latter is displayed instead of that with `.ace` extension. If the parameter is omitted then the file `load.ace` is loaded (as in MSTS) or `load.dds` (if present and, again, the dds support is enabled).

The loading screen image can have any resolution and aspect ratio; it will be displayed letter-boxed on the screen keeping the aspect ratio.

Another optional parameter `ortsloadingscreenwide`, can specify the image to show when the user loads the route on a wide (16:9) screen. This parameter is ignored when a traditional 4:3 display is used.

15.11 MSTS-Compatible semaphore indexing

When a signal shape has a semaphore (moving part), and its animation definition within the .s file has only two lines (e.g `slerp_rot` lines), MSTS interprets the `SemaphorePos()` lines within `sigcfg.dat` accordingly to following rule:

```
- SemaphorePos (2) is executed as SemaphorePos (1)
- SemaphorePos (1) is executed as SemaphorePos (0)
- SemaphorePos (0) is executed as SemaphorePos (0).
```

Open Rails follows this rule, in case one of the SemaphorePos lines has 2 as parameter. It does not follow this rule in case only 1 and 0 as parameters are present, because in such a case following the above rule they would be both executed as SemaphorePos (0) and therefore the semaphore would be static.

It is however strongly recommended to always have three animation lines within the .s file, where usually the third line repeats the parameters of the first line (except for the animation step).

15.12 Automatic door open/close on AI trains

The feature is explained *here*.

To override the selection made in the *Experimental Options Window*, a command line must be inserted in a small integration .trk file, that must be located in an Openrails subfolder within the route's folder, and must have the same name as the base folder. Here below an example of such file:

```
include ( "../Platformtest.trk" )
                    ORTSOpenDoorsInAITrains ( 1 )
```

The first line must be empty.

`ORTSOpenDoorsInAITrains ( 1 )` forces door open/close for this route even if the option within the Experimental Options Window is not checked.

`ORTSOpenDoorsInAITrains ( 0 )` disables door open/close for this route even if the option within the Experimental Options Window is checked.

15.13 Removing forest trees from tracks and roads

OR and MSTS determine differently the position of trees in forests. This may result in trees appearing on tracks or roads. To avoid trees on tracks following OR-specific parameter can be added to the .trk file of the route:

```
ORTSUserPreferenceForestClearDistance ( 2 )
```

where the parameter represents a minimum distance in metres from the track for placement of forests. Alternatively, the original .trk file can be left unmodified, and a new .trk file inserted into an `OpenRails` folder in the root folder of the route. This is explained *here*.

To avoid also forest trees on roads following line:

```
ORTSUserPreferenceRemoveForestTreesFromRoads ( 1 )
```

must be added below line:

```
ORTSUserPreferenceForestClearDistance ( 2 )
```

either in the route's root .trk file or in the "Include" .trk file.

It is not possible to remove trees only from roads and not from tracks.

15.14 Multiple level crossing sounds

This feature allows to have level crossing sounds different from the default one for a specific level crossing on a route or for a specific level crossing shape. To get a level crossing sound different from the default one for a specific level crossing sound on a route a line like following one has to be inserted in the .w file LevelCrObj block:

```
ORTSSoundFileName ( "differentcrossingsound.sms" )
```

where "differentcrossingsound.sms" must be replaced with the desired .sms file name.

> **Caution:** If the route is edited with the MSTS route editor modifying the .w files containing such line, the above line will be deleted.

To avoid this problem, two other possibilities are available to insert the additional line. One is described *here*. The other one is to use the OR specific TSRE route editor, that natively manages this feature. Also in the latter case if the route is later edited with the MSTS route editor, the above line will be deleted.

To get a level crossing sound different from the default one for a specific level crossing shape a line like following one must be inserted in the .sd file of the crossing shape:

```
ESD_ORTSSoundFileName ( "differentcrossingsound.sms" )
```

If both lines are present, the first overrides the second. For the first case it is suggested to place the sound file in the sound folder of the route, although it will also be searched in the general Train Simulator Sound folder. For the second case there is no suggestion. The file will again be searched in both folders.

15.15 Defining Curve Superelevation

This feature allows curves within the route to be assigned a value for superelevation. It is inserted either in the route's root .trk file or in the "Include" .trk file.

It should be noted that currently this information only applies to physics calculations for superelevation. The visual movement of the train due to superelevation is set within the option menu.

The values are assigned by inserting the following parameter:

```
ORTSTrackSuperElevation ( x y .......... )
```

where x and y are a series of paired parameters specifying the curve radius in metres (x value), and the amount of superelevation in metres (y value). The statement will take as many paired values as desired. Each paired set of values must have an x and y value present. If it is desired to 'hold' a certain value of SuperElevation for a number of different radii curves, then the same y value needs to be used for succeeding values of curve radius. Where the y value changes between curve radii, then Open Rails will extrapolate the y value between the two points.

15.16 Overhead (catenary) wire

Open Rails uses texture overheadwire.ace to display the overhead wire. Such texture must be present in the route's `TEXTURES` folder. If the texture is not found there, Open Rails looks for it in the `GLOBAL\TEXTURES` folder. If the texture isn't there either, Open Rails selects texture `GLOBAL\TEXTURES\diselsmoke.ace`. It is however strongly suggested to use a specific texture to display the overhead wire. A possible texture to be used can be downloaded here `Documentation\SampleFiles\Manual\overheadwire.zip`.

15.17 Fading signal lamps

In Open Rails, signal lamps fade on and off for a visually pleasing transition effect. The fade time defaults to one-fifth of a second. It can be customized in the `SignalType` block of the `sigcfg.dat` file using the `ORTSOnOffTimeS` property:

```
SignalTypes( ...
    SignalType ( "AM14Light"
        ...
        ORTSOnOffTimeS ( 0.2 )
    )
)
```

The value is the fade time in seconds. Use `0` to disable the effect completely.

15.18 Animated clocks



Animated clocks that show the simulation time can be added or retro-fitted to a route. The clocks can have a second-hand that ticks each second, or one that moves smoothly or none at all. Typically clocks could be station clocks, church tower clocks or clocks at other public buildings. They are placed as normal static shapes in a route, similar to other shapes such as houses or trees.

Note: Loco cabs already have provision for both analogue and digital clocks.

15.18.1 Overview

You will need:

1. Shape and Texture Files

   A shape file which defines each shape of clock, its hands and their animation and the texture files used by the shape.

2. Reference File

   For each shape of clock in the route, a reference to the shape file in the reference file.

3. World File

   The location of each clock in the world must be given in the world file.

15.18.2 Details

1. Shape and Texture Files

   Create a clock just like any other shape. The hands of the clock must be sub-objects within the shape. They must have specific names and an animation.

   Open Rails looks for the following names of clock hands in the shape file and animates them according to the simulation time.

   The names for the clock hands must start with:

   - “ORTS_HHand_Clock” for the hour hand
   - “ORTS_MHand_Clock” for the minute hand
   - “ORTS_SHand_Clock” for the second hand
   - “ORTS_CHand_Clock” for the centi-second hand

   This last is used to provide a smooth movement in hundredths of a second whereas the second hand ticks forward once a second. It is suggested to use either the second hand or the centisecond hand or neither.



If a clock is to have several hands of the same type, simply append a number to the names of the hands, like this:



The animation requires 4 key frames at the 12, 3, 6 and 9 positions and calculates the intermediate positions using linear interpolation.



For example:

```
anim_node ORTS_HHand_Clock01 (
        controllers ( 1
                tcb_rot ( 5
                        slerp_rot ( 0  0     0 0 -1 )
                        slerp_rot ( 1 -0.707 0 0 -0.707 )
                        slerp_rot ( 2 -1     0 0  0 )
                        slerp_rot ( 3 -0.707 0 0  0.707 )
                        slerp_rot ( 4  0     0 0  1 )
                      )
              )
     )
```

Finally, move the clock shape and its textures into the corresponding folders SHAPES and TEXTURES of your route, such as ROUTES\<route_name>\SHAPES\clocks.s

2. Reference File

   Add a reference to the shape file into the reference file ROUTES\<route_name>\<route_name>.ref Make sure that this reference begins with the “Static” keyword.:

```
Static (
        Filename    ( "ChurchClock.s" )
        Class       ( "Clocks" )
        Align       ( None )
        Description ( "ChurchClock" )
)
```

3. World File

   Use a route editor to locate the clocks in the world file.

   Note: Do not insert the shapes as animated ones. Otherwise, if MSTS is used to view the route then the hands of the clock will rotate wildly. In Open Rails they will match the simulation time anyway.

CHAPTER 16

Developing OR Content

Open Rails already has some own development tools and is defining and developing other ones. A path editor is available within TrackViewer under the *Tools* button in the main menu window. An editor for timetable mode is also available under the *Tools* button. Route editor and consist editor are in an advanced stage of development and may already be tested. You can read about and download the consist editor here . You can read about and download the TSRE5 route editor at this link

IT is of course already possible to develop OR content (rolling stock, routes, 3D objects, activities) using the tools used to develop MSTS content, thanks to the high compatibility that OR has with MSTS. Below, some of the advantages of OR-specific content are described.

16.1 Rolling Stock

- OR is able to display shapes with many more polygons than MSTS. Shapes with more than 100.000 polys have been developed and displayed without problems.
- Thanks to the additional physics description parameters, a much more realistic behavior of the rolling stock is achieved.
- 3D cabs add realism.
- OR graphics renders the results of the rolling stock developers at higher resolution.
- Rolling stock running on super-elevated track improves gaming experience.

16.2 Routes

- Routes are displayed in higher resolution.
- Extended viewing distance yields much more realism.
- *Double overhead wire* increases the realism of electrified routes.
- Built-in *triphase overhead electric line*.
- Extended signaling features provide more realistic signal behavior.
- *Widescreen* and hi-res loading screen.

16.3 Activities

- *Timetable mode* is a new activity type available only in Open Rails that allows for development of timetable based gaming sessions.
- By using the dispatcher monitor window, the dispatcher HUD, and the ability to switch the camera to any AI train, the player can more closely monitor and control the execution of conventional activities.
- *Extended AI shunting* greatly increases the interactions between trains.
- New *OR-specific additions* to activity (.act) files enhance activities.

16.4 Parameters and Tokens

The parameters used in content files have been mentioned throughout this manual for:

Content Type	File Extension
locomotive	eng
wagon or non-powered vehicle	wag
activity	act
cab view	cvf
consist	con
train service	srv
train traffic	trf
signal configuration	sigcfg.dat
signal scripts	sigscr.dat
sound management	sms
train timetable	timetable-or

The complete list is very extensive and is documented in an online spreadsheet at tinyurl.com/or-parameters-excel.

Since this is a spreadsheet with many rows, you can restrict your view to relevant rows using the filters at the top of each column.

16.5 Testing and Debugging Tools

As listed *here*, a rich and powerful set of analysis tools eases the testing and debugging of content under development.

16.6 Open Rails Best Practices

16.6.1 Polys vs. Draw Calls – What's Important

Poly counts are still important in Open Rails software, but with newer video cards they're much less important than in the early days of MSTS. What does remain important to both environments are Draw Calls.

A Draw Call occurs when the CPU sends a block of data to the Video Card. Each model in view, plus terrain, will evoke one or more Draw Calls per frame (i.e., a frame rate of 60/second means all of the draw calls needed to display a scene are repeated 60 times a second). Given the large number of models displayed in any scene and a reasonable frame rate, the total number of Draw Calls per second creates a very large demand on the CPU. Open Rails software will adjust the frame rate according to the number of

required Draw Calls. For example, if your CPU can handle 60,000 Draw Calls per second and the scene in view requires 1000 Draw Calls, your frame rate per second will be 60. For the same CPU, if the scene requires 2000 Draw Calls, your frame rate per second will be 30. Newer design / faster CPU's can do more Draw Calls per second than older design / slower CPU's.

Generally speaking, each Draw Call sends one or more polygon meshes for each occurrence of a texture file for a model (and usually more when there are multiple material types). What this means in practice is if you have a model that uses two texture files and there are three instances of that model in view there will be six draw calls – once for each of the models (3 in view) times once for each texture file (2 files used), results in six Draw Calls. As an aid to performance Open Rails will examine a scene and will issue Draw Calls for only the models that are visible. As you rotate the camera, other models will come into view and some that were in view will leave the scene, resulting in a variable number of Draw Calls, all of which will affect the frame rate.

Model builders are advised that the best performance will result by not mixing different material types in a texture file as well as using the fewest number of texture files as is practical.

16.7 Support

Support can be requested on the OR forum on http://www.elvastower.com/forums.

The OR development team, within the limits of its possibilities, is willing to support contents developers.

CHAPTER 17

Version 1.3 Known Issues

17.1 Empty Effects Section in .eng File

If an .eng file is used that has an `Effects()` section that contains no data, the engine will not be loaded by ORTS. In this case it is suggested to fully delete the `Effects()` section.

17.2 Curly brackets in file sigscr.dat

Open Rails does not correctly handle, and also generates a misleading error message in file OpenRailsLog.txt file, when there is a curly bracket at the end of a conditional statement within file sigscr.dat, e.g.:

```
if ( next_hp ==# 0 && next_gue !=# 2 ) {
```

Therefore the file must be edited as follows to be correctly interpreted by Open Rails:

```
if ( next_hp ==# 0 && next_gue !=# 2 )
{
```

17.3 Spurious emergency braking in Timetable mode

If in Timetable mode a speedplate with higher speedlimit follows a signal with reduced speedlimit, the allowed speed in the Trackmonitor rises to the speed shown on the speedplate. This occurs accordingly to specs of Timetable mode (and differently from activity mode).

However the overspeedmonitor considers the reduced signal speed, coherently with activity mode. Therefore in this case if, in timetable mode, a train is accelerated above the signal speed, the overspeedmonitor may trigger an emergency braking.

CHAPTER 18

In Case Of Malfunction

18.1 Introduction

When you have an issue with Open Rails (ORTS), no matter what it is, the OR development team is always thankful for reports of possible bugs. Of course, it is up to the developers to decide if something is a real bug, but in any case your reporting of it is an important step in helping the development team to improve Open Rails.

18.2 Overview of Bug Types

The development team uses two ways of keeping track of bugs:

1. So called “Maybe-Bugs” are reported in a simple forum post: see next paragraph for links. This is done in order to give developers a chance to filter out problems caused by circumstances the development team cannot control such as corrupted content.
2. Decided Bugs are issues a developer has looked at and has found to be a real issue in the program code of Open Rails. They are reported at our Bug Tracker at https://bugs.Launchpad.net/or/ (registration is required).

18.3 Maybe-Bugs

If you find an issue with Open Rails you should first file a Maybe-Bug report at any of the following forums monitored by the Open Rails development team:

- Elvas Tower, “Maybe it’s a bug” section of the Open Rails sub-forum. This is the forum that is most frequently checked by the OR development team;
- TrainSim.com, “Open Rails discussion” section of the Open Rails sub-forum
- …more forums may be added in the future

A Maybe-Bug report consists of a simple post in a new topic in the forum. The title of the topic should be of the form “Open Rails V#### Bug: +++++”, where #### is the version number of the Open Rails release you are having problems with, and +++++ is a quick description of the problem you are having. This format aids the developers in getting a quick idea of the issue being reported.

The first post in this newly started topic should give further information on your problem: Start out with exactly what problem you are getting, describing it in narrative and supplementing this description with screenshots, error messages produced by Open Rails, and so on.

Next give a clear indication of the content you were using (that is: Route, Activity, Path, Consist, Locomotive and Rolling Stock; whatever is applicable), whether it is freeware or payware, what the exact name of the downloaded package was and where it can be obtained. Of course, posting a download link to a trustworthy site or directly attaching files to the post also is OK.

Continue with an exact description of what you were doing when the problem arose (this may already be included in the first paragraph, if the problem is train-operation-related). Again, screenshots etc. can be helpful to better describe the situation.

Lastly, take a look at your desktop for a text file entitled `OpenRailsLog.txt`. Upload and attach this file to the end of your post. This is very important as the log file contains all relevant program data the user has no chance to ever see, and thus it is one of the most important sources of information for the developer trying to solve your problem.

Once your post has been submitted, keep adding further information only in additional posts, in order to avoid the risk of people not noticing your edits. Also, please be patient with developers responding to your report. Most forums are checked only once a day, so it may take some time for a developer to see your report.

Important: The more information a developer gets from the first post, the quicker he will be able to locate, identify and eventually resolve a bug. On the other hand, reports of the form, “I have problem XYZ with recently installed Open Rails. Can you help me?” are of little use, as all required information must be asked for first.

Important: Please do not rush to report a Decided Bug on the Bug Tracker before a developer has declared your problem a real bug!

The above description is available in a condensed “checklist” form below.

18.4 Decided bugs

Many bug reports never even make it to the status of a Decided Bug, being a content or user error. Some Maybe-Bugs, however, will eventually be declared Decided Bugs. Such secured bugs should be reported at our Bug Tracker, when the developer taking the report asks you to.

The Open Rails Bug Tracker is found at https://bugs.Launchpad.net/or/, following the “Report a bug” link in the upper half to the right of the screen. You will need to register at Launchpad in order to be able to report a bug.

Once that is done, follow the steps the software takes you through: In “Summary” copy and paste the quick description of the bug you also entered as a forum thread name for the Maybe-Bug report.

Next, look through the list of topics Launchpad thinks your bug may be related to – maybe your issue has already been reported?

If you cannot relate to any of the suggested bugs, click the “No, I need a new bug report” button and continue.

In the “Further Information” field, enter the same info you also gave in the Maybe-Bug report (copy and paste). Screenshots may need to be added as attachments, and you will also need to re-upload the `OpenRailsLog.txt` file. Do not forget to include all info you added in additional posts to the original Maybe-Bug report, and also add a link to the latter at the bottom of the “Further Information” field.

Once your bug has been submitted, keep adding further information only in additional posts, in order to avoid the risk of developers missing the additional info.

The above description is available in a condensed “checklist” form below.

Important: Do not say “All information is included in the linked thread” as skimming through a thread for the crucial bit of information is a really annoying task. Instead, please provide a concise, but complete summary of the Maybe-Bug thread in the “Further Information” field.

Important: Please do not rush to report a Decided Bug on our Bug Tracker before a developer has declared your Maybe-Bug a real bug!

18.5 Additional Notes

Please do not post feature requests as a Maybe-Bug to the Bug Tracker on Launchpad!

Please do not report the same bug multiple times, just because the first report did not get attention within a short time. Sorting out the resulting confusion can slow things down even more.

Please do not report Bugs directly to the Bug Tracker when you are not 100% sure it’s a real, significant bug, or have not been asked to do so.

Don’t be offended by bug statuses - they often sound harsher than they really mean, like “Invalid”.

Don’t expect a speedy response in general – issues will get looked at as and when people have the time.

Be prepared to expand upon the initial report – it is remarkably easy to forget some crucial detail that others need to find and fix your bug, so expect to be asked further questions before work can begin.

Try to avoid comments that add no technical or relevant detail – if you want to record that the bug affects you, Launchpad has a dedicated button at the top: “Does this bug affect you?”.

If you wish to follow the progress of someone else’s bug report and get e-mail notifications, you can subscribe to bug mail from the sidebar.

18.6 Summary: Bug Report Checklists

“Maybe-Bug”

- New topic in appropriate sub-forum
- Topic Title: “Open Rails V<version> Bug: <description>”
- Description of problem, supplemented by screenshots etc.
- Content used (Route, Activity, Path, Consist, Locomotive & Rolling Stock; choose applicable); Freeware / Payware?; Package name & download location / download link
- Narrative of actions shortly before & at time of problem, supplemented by screenshots etc.
- Attach log file (Desktop: `OpenRailsLog.txt`)
- Add further info only in additional posts
- Be patient

Decided Bug

- Report to Bug Tracker only if asked to do so
- https://bugs.Launchpad.net/or/ (Registration required) -> “Report a bug”
- “Summary”: Description from the topic title of the Maybe-Bug report
- Look for similar, already reported bugs
- Condense whole Maybe-Bug thread into “Further information” field
- Add link to original Maybe-Bug report
- Re-upload and attach OpenRailsLog.txt & explanatory screenshots etc.

- Add further info only in additional posts
- Be patient

18.7 Bug Status in Launchpad

- **New** – this is where all bugs start. At this point, the bug has not been looked at by the right people to check whether it is complete or if more details are needed.
- **Incomplete** – a member of the Open Rails teams has decided that the bug needs more information before it can be fixed. The person who created the bug report does not have to be the one to provide the extra details. A bug remaining incomplete for 60 consecutive days is automatically removed.
- **Opinion** – the bug has been identified as an opinion, meaning that it isn't clear whether there is actually a bug or how things should be behaving.
- **Invalid** – a member of the team believes that the report is not actually a bug report. This may be because Open Rails is working as designed and expected or it could just be spam. The bug may be put back to the new state if further information or clarity is provided in comments.
- **Won't Fix** – a member of the team has decided that this bug will not be fixed at this time. If the bug report is a "feature request", then they have decided that the feature isn't desired right now. This status does not mean something will never happen but usually a better reason for fixing the bug or adding the feature will be needed first.
- **Confirmed** – a member of the team has been able to experience the bug as well, by following the instructions in the bug report.
- **Triaged** – a member of the team has assigned the importance level to the bug or has assigned it to a specific milestone. Bugs generally need to get to this state before the developers will want to look at them in detail.
- **In Progress** – one or more members of the team are currently planning to or actually working on the bug report. They will be identified by the assignee field.
- **Fix Committed** – the fix for the bug report or feature request has been completed and checked in to the source control system, Subversion. Once there, the fix will usually appear in the next experimental release.
- **Fix Released** – The code containing the bug fix has been released in an official release.

18.8 Disclaimer

Having posted a bug report in a forum or in Launchpad does not generate any obligation or liability or commitment for the OR development team to examine and fix the bug. The OR development team decides whether it will examine and fix the bug on a completely voluntary and autonomous basis.

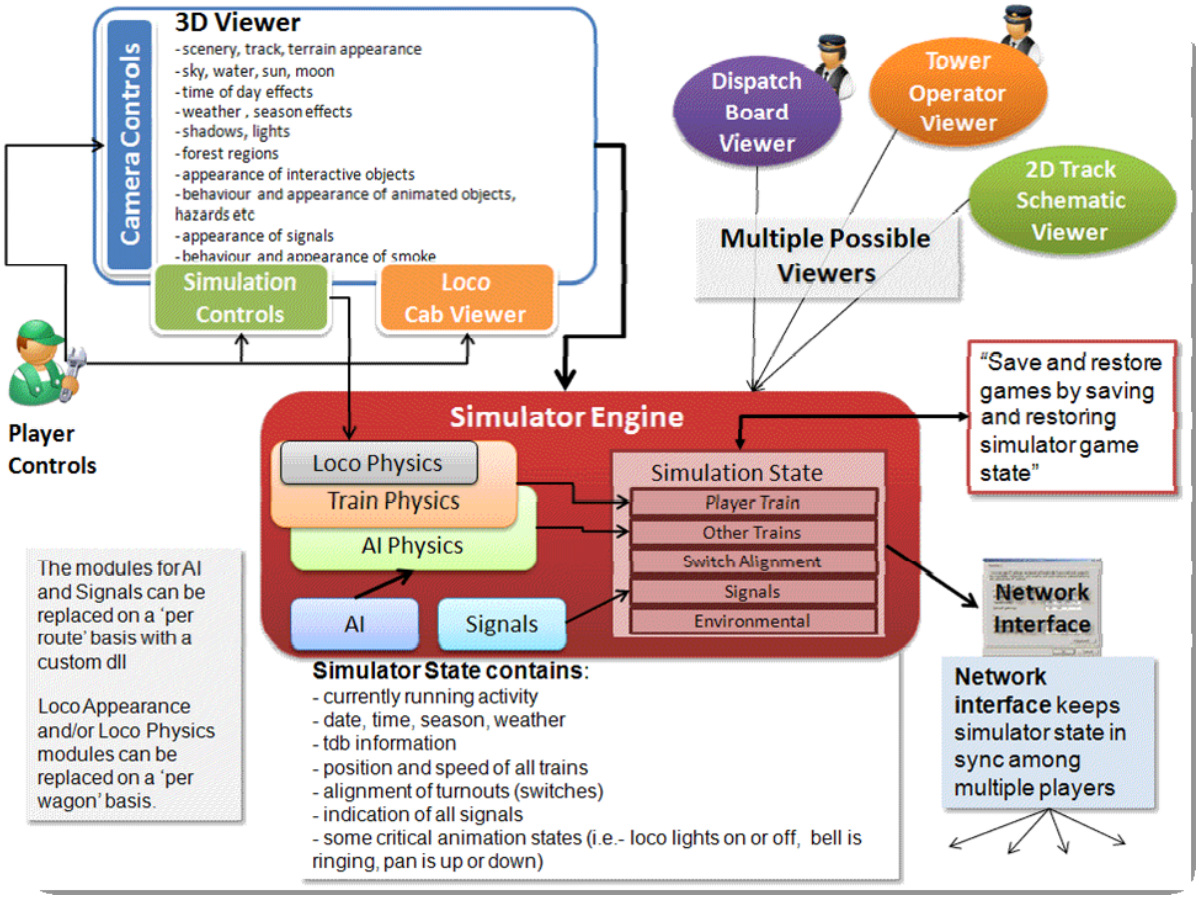
CHAPTER 19

Open Rails Software Platform

19.1 Architecture

To better understand how the Open Rails game operates, performs, and functions, the architecture diagram below lays out how the software code is organized. The architecture of the Open Rails software allows for modular extension and development, while providing standardized methods to customize the simulation experience.

Note: Please note that this diagram includes many capabilities and functions that are yet to be implemented.



19.2 Open Rails Game Engine

The Open Rails software is built on the MonoGame platform. MonoGame is an open source implementation of the Microsoft XNA 4 Framework and provides:

- Game framework
- 2D and 3D rendering
- Sound effect and music playback
- Keyboard, mouse, touch, and controller inputs
- Content building and optimization
- Math library optimized for games

19.3 Frames per Second (FPS) Performance

FPS rate is as default not linked to the sync rate of the monitor. However, with *this option* FPS rate may be set at the value of the monitor refresh rate.

19.4 Game Clock and Internal Clock

Like other simulation software, Open Rails software uses two internal *clocks*; a game clock and an internal clock. The game clock is required to synchronize the movement of trains, signal status, and present the correct game environment. The internal clock is used synchronize the software process for optimal efficiency and correct display of the game environment.

The Open Rails team is dedicated to ensuring the game clock properly manages time in the simulation, so that a train will cover the proper distance in the correct time. The development team considers this vital aspect for an accurate simulation by ensuring activities run consistently across community members' computer systems.

19.5 Resource Utilization

Because Open Rails software is designed for the MonoGame framework, it natively exploits today's graphics cards' ability to offload much of the display rendering workload from the computer's CPU.

19.6 Multi-Threaded Coding

The Open Rails software is designed from the ground up to support up to 4 CPUs, either as virtual or physical units. Instead of a single thread looping and updating all the elements of the simulation, the software uses four threads for the main functions of the software.

- Thread 1 – Main Render Loop (RenderProcess)
- Thread 2 – Physics and Animation (UpdaterProcess)
- Thread 3 – Shape and Texture Loading/Unloading (LoaderProcess)
- Thread 4 – Sound

There are other threads used by the multiplayer code as each opened communication is handled by a thread.

The RenderProcess runs in the main game thread. During its initialization, it starts two subsidiary threads, one of which runs the UpdaterProcess and the other the LoaderProcess. It is important that the UpdaterProcess stays a frame ahead of RenderProcess, preparing any updates to camera, sky, terrain, trains, etc. required before the scene can be properly rendered. If there are not sufficient compute resources for the UpdaterProcess to prepare the next frame for the RenderProcess, the software reduces the frame rate until it can *catch up*.

Initial testing indicates that *stutters* are significantly reduced because the process (LoaderProcess) associated with loading shapes and textures when crossing tile boundaries do not compete with the main rendering loop (RenderProcess) for the same CPU cycles. Thread safety issues are handled primarily through data partitioning rather than locks or semaphores to maximise performance.

Ongoing testing by the Open Rails team and the community will determine what and where the practical limits of the software lie. As the development team receives feedback from the community, improvements and better optimization of the software will contribute to better overall performance – potentially allowing high polygon models with densely populated routes at acceptable frame rates.

19.7 Web Server

The game uses a built-in web-server to deliver standard and custom web-pages to any browser. This can be running on the same PC as Open Rails or another PC or other device which is connected to your local network.

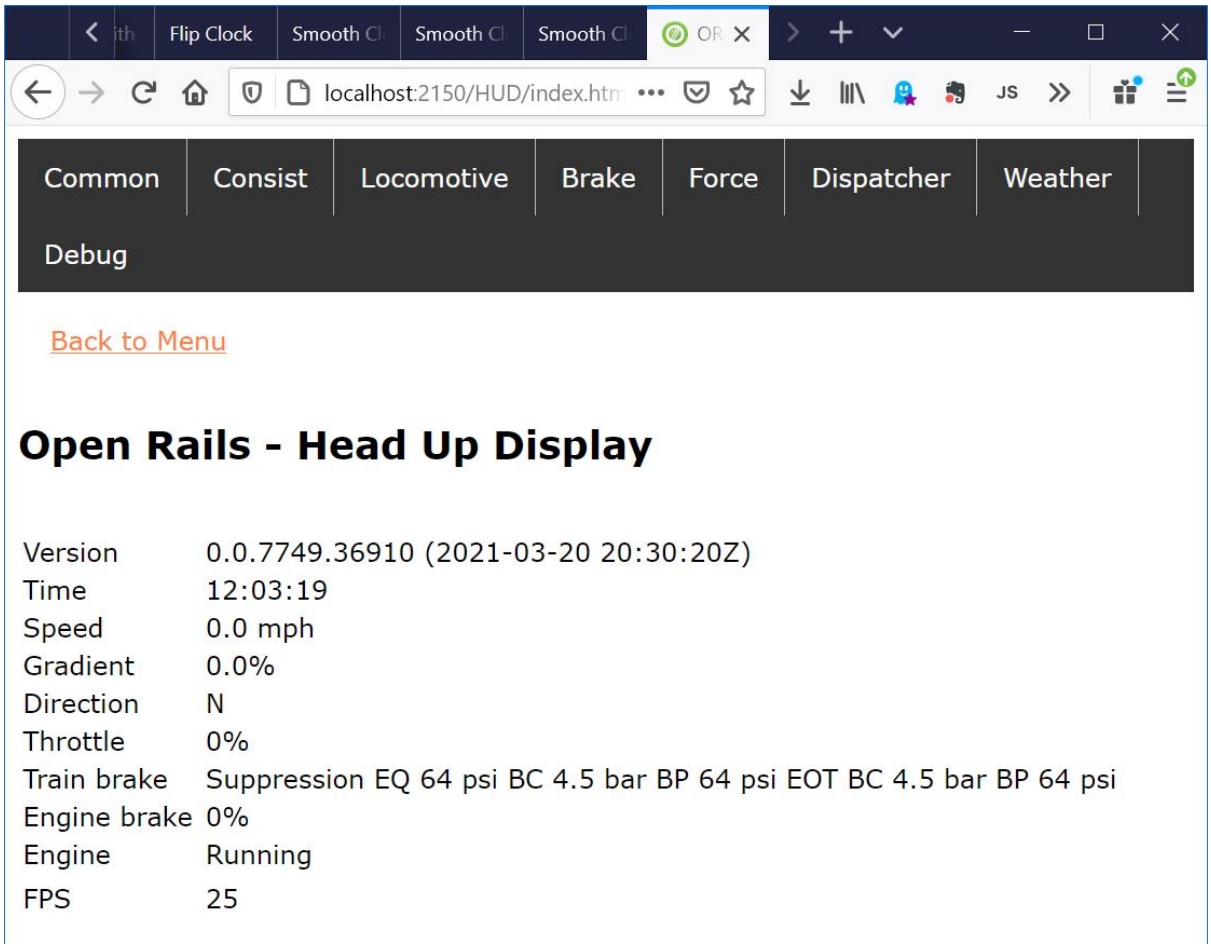
The simplest way to access these samples is to start the game and then launch a browser on the same PC. Then enter "localhost:2150" into your browser address bar. (2150 is the default port number set in *Menu > Options > General*)

19.7.1 Sample Web Pages

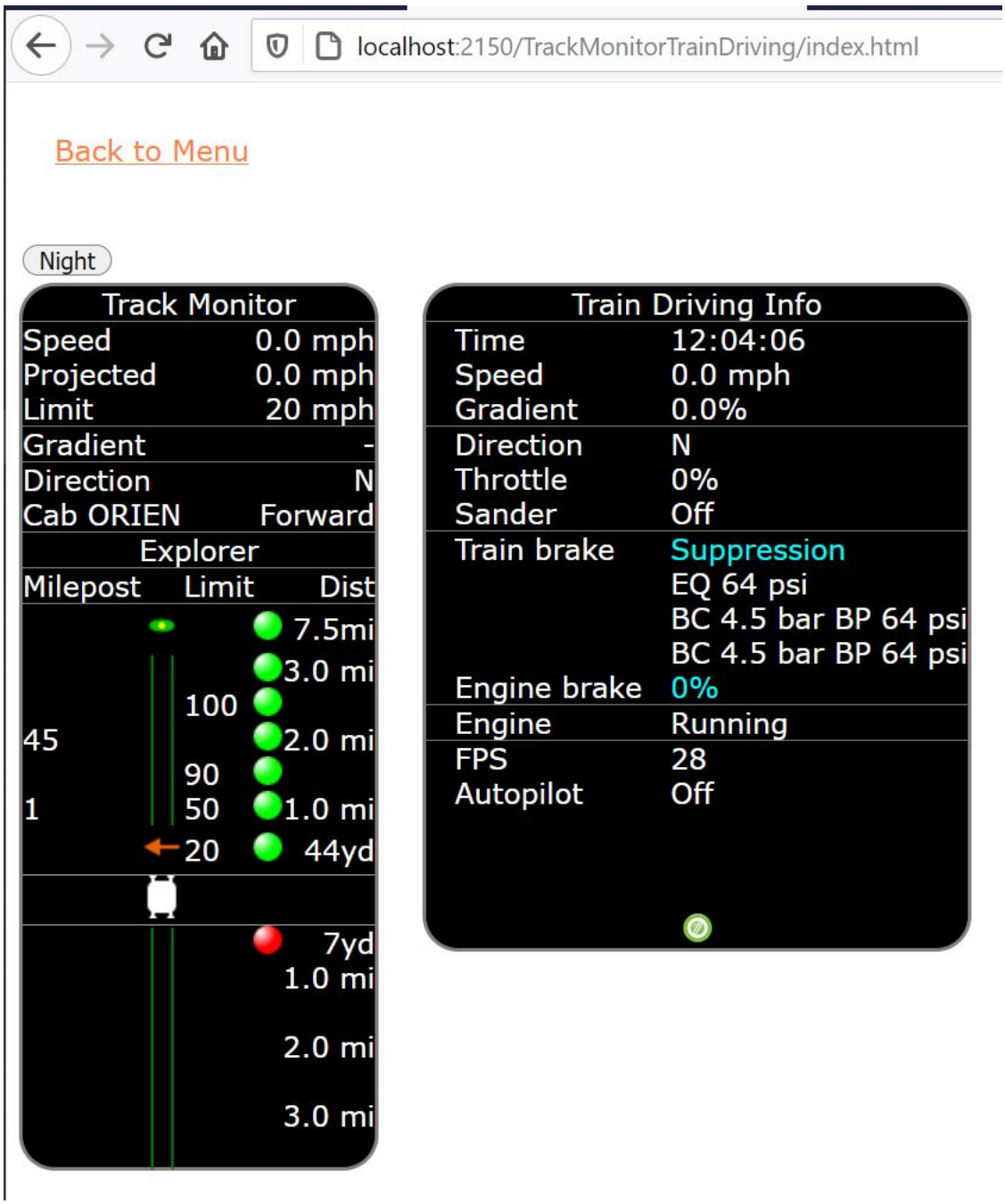
A number of web-pages are included in the Open Rails installation as examples of what can be done with the APIs.

Some of these sample pages repeat data from the in-game panels to provide a more convenient display.

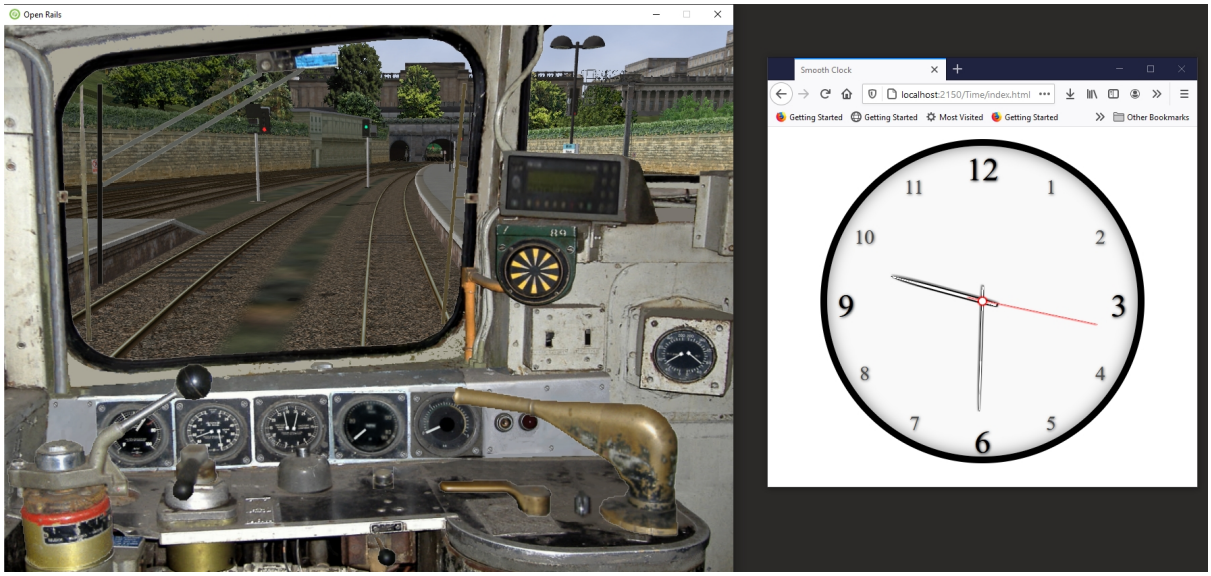
- The HUD web page repeats the F5 overlay.



- The Track Monitor page repeats the F4 panel and is also available with a dark background for night-time use.
- The Train Driving page offers a panel which is not yet available in the official version of Open Rails.
- Another page offers both of these and the panels can be dragged around for the best arrangement.



- The time page shows the simulation time as a digital clock and links to 3 versions of an analogue clock.



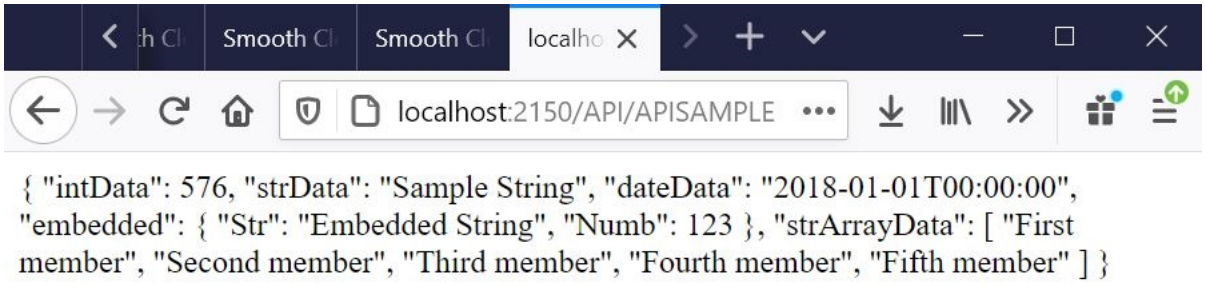
The sample pages can be found in the `Content\Web` subfolder of the OR program folder and the web server defaults to `Content\Web\index.html`.

If you choose to develop your own pages, please consider sharing them with the Open Rails community.

19.7.2 Application Programming Interfaces (APIs)

The web server features a simple API to obtain data from the simulator. Responses are OR data structures serialized in JSON format.

You can see the JSON data just by browsing. E.g.: for APISample, browse to `http://localhost:2150/API/APISAMPLE`



Note: The API portion of this address is case-sensitive.

Note: To avoid overloading the simulator, please keep API calls to once or twice a second.

Method	API call	Description	Response type
GET	/API/HUD/<n>	Retrieves the information rendered on the <F5> HUD, row by row, page by page, where <n> is the HUD page number 0 to 7.	Orts.Viewer3D.WebServices .WebServer.ORTSApiController .HudApiArray
GET	/API/ TRAINMONITOR or /API/TRAININFO	Retrieves information rendered on the Track Monitor, such as speed, acceleration, grade, and upcoming hazards.	Orts.Simulation.Physics .Train.TrainInfo
GET	/API/TIME	Retrieves just the simulation time in seconds since midnight.	Orts.Viewer3D.WebServices .WebServer.ORTSApiController .ApiTime
GET	/API/ CABCONTROLS	Retrieves an array of the cab controls for the player localhost TypeName, MinValue, MaxValue, RangeFraction.	Orts.Viewer3D.WebServices .WebServer.ORTSApiController .ApiCabControls
GET	/API/APISAMPLE	A test object that demonstrates the JSON serialization of various data types.	Orts.Viewer3D.WebServices .WebServer.ORTSApiController .ApiSampleData

CHAPTER 20

Plans and Roadmap

Here are some highlights that the community can expect from the Open Rails team after v1.0. A more complete roadmap can be found at https://launchpad.net/or/+milestones

20.1 User Interface

A new Graphical User Interface (GUI) within the game.

20.2 Operations

In addition to the new Timetable concept described in this document, some further improvements are planned:

- Extended ability to customize signals to accommodate regional, geographic, or operational differences
- Ability to use mixed signal environments – from dark territory to fully automatic in-cab train control within the same route
- Specifying random variations for AI trains in consist and delays.
- Specifying separate speed profiles for passenger or freight trains.
- A schedule for AI trains which can depend on other trains (e.g. wait a limited time).

20.3 Open Rails Route Editor

The Open Rails Route Editor (called TSRE5) is well under way, and it is expected that in reasonable time it will replace the MS Route Editor. However, *no timetable is available for this work*. The route editor already can use GIS data. it will be possible to lay both track pieces and procedural track. TSRE5 is able to read route files created with the MS Route Editor, however it extends the MSTS file structure allowing for new functions. Routes that will use these extensions will in general not run under MSTS.

CHAPTER 21

Acknowledgements

Open Rails is the result of true teamwork performed by a group of passionate people. We owe a massive thanks to all of them and therefore wish to mention them below and excuse ourselves if someone has been forgotten:

Adam Kane	Derek Morton	Jim Ward	Remus Iancu
Adam Miles	Doug Kightley	John Sandford	Richard Plokhaar
Alex Bloom	Douglas Jones	Joseph Hoevet	Rick Grout
Andre Ming	Edward Keenan	Joseph Realmuto	Rick Hargraves
Anthony Brailsford	Eric Pannese	Larry Steiner	Riemer Grootjans
Barrie Scott	Eric Swenson	Laurie Heath	Rob Lane
Barry Munro	Eugen Rippstein	Lutz Doellermann	Robert Hodgson
Bill Currey	Fabian Joris	Marc Nelson	Robert Murphy
Bill Prieger	Greg Davies	Markus Gelbmann	Robert Roeterdink
Bob Boudoin	György Sárosi	Matêj Pácha	Roberto Ceccarelli
Bruno Sanches	Haifeng Li	Matt Munro	Samuel Kelly
Carlo Santucci	Hank Sundermeyer	Matt Peddlesden	Scott Miller
Chris Jakeman	James Ross	Mauricio Muñoz	Sid Penstone
Chris Van Wagoner	Jan Vytlačil	Paul Bourke	Tim Muir
Craig Benner	Jean-Louis Chauvin	Paul Gausden	Walter Niehoff
Dan Reynolds	Jeff Bush	Paul Wright	Wes Card
Daniel Leach	Jeffrey Kraus-Yao	Peter Gulyas	
David B. Clarke	Jijun Tang	Peter Newell	
Dennis Towlson	Jim Jendro	Phil Voxland	

And a special thanks to:

- Dave Nelson for providing us a meeting place at Elvas Tower
- Pete Peddlesden for hosting our website and repository
- And, of course, Wayne Campbell for inspiring this improbable journey

CHAPTER 22

Appendices

22.1 Units of Measure

Open Rails supports the same default units of measure as MSTS which are mostly, but not exclusively, metric.

When creating models just for Open Rails, we recommend you do not use defaults but specify units for all values that represent physical quantities.

As shown below, Open Rails provides a wider choice of units than MSTS.

Measure	Default unit	Applies to	OR accepts	MSTS accepts	Comment
Mass	kg		kg	kg	
			t	t	metric tonne (1000 kg)
			lb	lb	
			t-uk		Imperial ton (2240 lb)
			t-us		US ton (2000 lb)
Distance			mm		
			cm	cm	
	m		m	m	
			km		
			in	in	
			in/2	in/2	half-inch – historic unit for tyre diameters
			ft		

continues on next page

Table 1 – continued from previous page

Measure	Default unit	Applies to	OR accepts	MSTS accepts	Comment
			mile		
Area			m^2		
			*(m^2)	*(m^2)	
	ft^2		ft^2		
		*(ft^2)	*(ft^2)		
Volume	l	diesel fuel	l		liter
			m^3		
			*(m^3)		
			in^3		
			*(in^3)		
	ft^3	other	*(ft^3)	*(ft^3)	e.g. BoilerVolume
			g-uk		Imperial gallons
			g-us		US gallons
			gal		US gallons
			gals	gals	US gallons
Time	s		s		
			m		
			h		
Current	amp		amp		
			A		
Voltage	volt		V		
			kV		
Mass Flow			g/h		
			kg/h		
	lb/h		lb/h	lb/h	
Speed	m/s	other	m/s	m/s	meter per second
			km/h		
			kph	kph	kilometer per hour
			kmh	kmh	misspelling accepted by MSTS

continues on next page

Table 1 – continued from previous page

Measure	Default unit	Applies to	OR accepts	MSTS accepts	Comment
		kmph			
	mph	dynamic brake	mph	mph	miles per hour
Frequency	Hz		Hz		Hertz
			rps		revolutions per second
			rpm		
Force	N		N	N	Newton
			kN	kN	
			lbf		Pounds force
			lb		
Power	W		W		Watt
			kW		
			hp		horsepower
Stiffness	N/m		N/m	N/m	Newton per meter
Resistance	N/m/s		N/m/s	N/m/s	Newton per meter per second
			Ns/m		Newton seconds per meter
Angular Resistance	N/rad/s		N/rad/s		
Pressure	psi	air pressure	psi		pounds per square inch
			bar		atmospheres
			kPa		KiloPascal
	inHg	vacuum	inHg		inches of mercury
Pressure Rate of Change	psi/s		psi/s		
			bar/s		
			kpa/s		
			inHg/s		
Energy Density	kJ/kg		kJ/kg		kiloJoule per kilogram
			J/g		
			btu/lb		Board of Trade Units per pound
Temperature Difference	degC		degC		
			degF		

continues on next page

Table 1 – continued from previous page

Measure	Default unit	Applies to	OR accepts	MSTS accepts	Comment
Angle	radians		–		
			deg		
Angular Speed	rad/s		–	rad/s	
Other			–	lb/hp/h	e.g. CoalBurnage

22.2 Folders used by Open Rails

The following folders are also written to by Open Rails.

(In this table, we assume a user called Joe.)

Table 2: Folders Table

Purpose	**Folder** (all beginning C:\Users\Joe\) + **Sample File**
Logs	Desktop\OpenRailsLog.txt
Data dumps	Desktop\OpenRailsDump.csv
Screenshots	Pictures\Open Rails\Open Rails 2021-09-21 07-26-58.png
Saves	AppData\Roaming\Open Rails\shunt_1 2021-07-18 19.46.35.save
Save images	AppData\Roaming\Open Rails\shunt_1 2021-07-18 19.46.35.png
Replays	AppData\Roaming\Open Rails\shunt_1 2021-07-18 19.46.35.replay
Evaluations	AppData\Roaming\Open Rails\shunt_1 2021-07-18 19.46.35.dbfeval
Loading progress bar	AppData\Roaming\Open Rails\Load Cache\3cd9… …0ce2.dat

22.3 Signal Functions

This is an overview of the functions available in OR for use in signal scripts, known as SIGSCRIPT functions.

22.3.1 Original MSTS Functions

The following are basic MSTS functions:

BLOCK_STATE
ROUTE_SET
NEXT_SIG_LR
NEXT_SIG_MR
THIS_SIG_LR
THIS_SIG_MR
OPP_SIG_LR
OPP_SIG_MR
DIST_MULTI_SIG_MR
SIG_FEATURE
DEF_DRAW_STATE

22.3.2 Extended MSTS Functions

The following are extensions of basic MSTS functions.

NEXT_NSIG_LR(SIGFN_TYPE, N)
Extension of NEXT_SIG_LR
Returns state of Nth signal of type SIGFN_TYPE. Note that state SIGASP_STOP is returned if any intermediate signal of type SIGFN_TYPE is set to that state.

DIST_MULTI_SIG_MR_OF_LR(SIGFN_TYPE, SIGFN_ENDTYPE)
Extension of DIST_MULTI_SIG_MR
The original DIST_MULTI_SIG_MR excluded any heads for which the link (route_set) was not valid. However, when signals are routed through route-definition signals rather than through links, this exclusion fails and therefore the function does not return the correct state. This extended function checks all required heads on each signal, and uses the least restricted aspect on this signal as state for this signal. It returns the most restrictive state of the states determined in this manner for each intermediate signal until a signal of type SIGFN_ENDTYPE is found.

22.3.3 SIGNAL IDENT Functions

When a function is called which requires information from a next signal, a search is performed along the train's route to locate the required signal. If multiple information is required from that signal, and therefore multiple functions are called requiring that next signal, such a search is performed for each function call.

This process can be made much more efficient by using the signal ident of the required signal. Each signal in a route has a unique ident. A set of functions is available to obtain the signal ident of the required signal. Also available are functions which are equivalent to normal signal functions, but use the signal ident and do not perform a search for the required signal. Obviously, using these functions it must be checked that the retrieved signal ident is valid (i.e. a valid signal is found), and the integrity of the variable holding this ident must be ensured (the value must never be altered).

The following functions are available to obtain the required signal ident. The functions return the signal ident for the signal as found. If no valid signal is found, the value of -1 is returned.

NEXT_SIG_ID(SIGFN_TYPE)
Returns Signal Ident of next signal of type SIGFN_TYPE.

NEXT_NSIG_ID(SIGFN_TYPE, N)
Returns Signal Ident of Nth signal of type SIGFN_TYPE.

OPP_SIG_ID(SIGFN_TYPE)
Returns Signal Ident next signal of type SIGFN_TYPE in opposite direction.

The following functions are equivalent to basic functions but use Signal Ident to identify the required signal.

ID_SIG_ENABLED(SigID)
Returns 1 if the identified signal is actively enabled (i.e. a train has cleared a route leading to that signal)

ID_SIG_LR(SigId, SIGFN_TYPE)
Returns the least restricted aspect of the heads with type SIGFN_TYPE of the identified signal.

Note there are other functions which also use the signal ident as detailed below.

22.3.4 Signal SubObject functions

In the original MSTS signal definition, a number of specific Signal SubObjects could be used as flags (USER_1 … USER_4). Other items (NUMBER_PLATE and GRADIENT) could also be used as flag but were linked to physical items on the signal. The number of flags available in this way was very restricted. In OR, an additional functions has been created which can check for any Signal SubObject if this SubObject is included for this particular signal or not. This function can be used on any type of Signal SubObject. By setting SubObjects of type 'DECOR', additional flags can be defined for any type of signal. SubObjects defined in this manner need not be physically defined in the shape file. The information is available at signal level, so all heads on a signal can use this information. The function uses the SubObject number to identify the required SubObject, the name of the SubObject is irrelevant. The maximum of total SubObjects for any shape is 32 (no. 0 … 31), this includes the actual signal heads.

HASHEAD(N)
Returns 'true' (1) if SubObject with number N is available on this signal.

22.3.5 Approach Control Functions

Approach Control is a method used in some signalling systems which holds a signal at danger until the approaching train is at a specific distance from the signal, or has reduced its speed to below a certain limit. This functionality is used in situations where a significant reduction in speed is required for the approaching train, and keeping the signal at danger ensures the train has indeed reduced its speed to near or below the required limit.

The following set of Approach Control Functions is available in OR.

The required distance and speed can be set as constants (dimensions are m and m/s, these dimensions are fixed and do not depend on any route setting).

It is also possible to define the required distance or speed in the signal type definition in sigcfg.dat. The values defined in this way can be retrieved using the pre-defined variables **Approach_Control_Req_Position** and **Approach_Control_Req_Speed**.

APPROACH_CONTROL_POSITION(APPROACH_CONTROL_POSITION)
The signal will be held at danger until the train has reached the distance ahead of the signal as set. The signal will also be held at danger if it is not the first signal ahead of the train, even if the train is within the required distance.

APPROACH_CONTROL_POSITION_FORCED(APPROACH_CONTROL_POSITION)
The signal will be held at danger until the train has reached the distance ahead of the signal as set. The signal will also clear even if it is not the first signal ahead of the train.

APPROACH_CONTROL_SPEED(APPROACH_CONTROL_POSITION, APPROACH_CONTROL_SPEED)
The signal will be held at danger until the train has reached the distance ahead of the signal as set, and the speed has been reduced to below the required limit. The speed limit may be set to 0 in which case the train has to come to a stand in front of the signal before the signal will be cleared. The signal will

also be held at danger if it is not the first signal ahead of the train, even if the train is within the required distance.

APPROACH_CONTROL_NEXT_STOP(APPROACH_CONTROL_POSITION, APPROACH_CONTROL_SPEED)
Sometimes, a signal may have approach control but the signal may be held at danger if the next signal is not cleared. Normally, if a signal is held for approach control, it will not propagate the signal request, meaning that the next signal will never clear. This could lead to a signal lockup, with the first signal held for approach control and therefore the next signal cannot clear. This function is specifically intended for that situation. It will allow propagation of the clear request even if the signal is held at danger for approach control, thus allowing the next signal to clear. The working of this function is similar to APPROACH_CONTROL_SPEED.

APPROACH_CONTROL_LOCK_CLAIM()
If a signal ahead of a train is held at danger, the train may claim sections beyond that signal in order to ensure a clear path from that signal as soon as possible. If this function is called in a script sequence which also sets an active approach control, no claims will be made while the signal is held for approach control.

22.3.6 CallOn Functions

CallOn functions allow trains to proceed unto a track section already occupied by another train. CallOn functions should not be confused with 'permissive' signals as often used in North American signal systems.

A 'permissive' signal will always allow a train to proceed on occupied track, following a previous train. Such signals are generally only used in situations where a signal covers a 'free line' section only, i.e. a section of track without switches or crossings etc.

The CallOn facility, on the other hand, will only allow the train to proceed in certain specific situations and is primarily used in station and yard areas.

The CallOn functions are specifically intended for use is timetable mode, and are linked directly to a number of timetable commands. Trains will be allowed to proceed based on these commands.

CallOn functions in timetable mode

The following conditions will allow a train to proceed.

- The route beyond the signal leads into a platform, and the **$callon** parameter is set for the related station stop.
- The train has an **$attach**, **$pickup** or **$transfer** command set for a station stop or in the #dispose field, and the train in the section beyond the signal is static or is the train as referenced in the command (as applicable). If the command is set for a station stop, the route beyond the signal must lead into a platform allocated to that station. If the command is set in the #dispose field, there are no further conditions.
- The train action is part of a **$stable** command in the #dispose field.
- The route beyond the signal is a Pool Storage path, and the train is booked to be stored in that pool.

CallOn may also be allowed if the route does not lead into a platform depending on the function call.

CallOn functions in activity mode

CallOn is never allowed if the route beyond the signal leads into a platform. CallOn may be allowed in other locations depending on the actual function call.

Available functions

TRAINHASCALLON()
TRAINHASCALLON_RESTRICTED()
These functions are similar, except that TRAINHASCALLON will always allow CallOn if the route does not lead into a platform, and therefore acts like a 'permissive' signal in that situation. The function TRAINHASCALLON_RESTRICTED will only allow CallOn when one of the criteria is met as detailed above.

22.3.7 SignalNumClearAhead Functions

The SignalNumClearAhead (SNCA) value sets the number of signals ahead which a signal will need to clear in order to be able to show the required least restrictive aspect. The value is set as a constant for each specific signal type in the sigcfg.dat file. However, it may be that certain signal options require that value to be changed. For instance, a signal type which optionally can display an advance approach aspect, needs a higher value for SNCA in case this advance approach is required. This may even depend on the route as set from that signal. In OR, functions are available to adjust the value of SNCA as required, which prevents the need to always set the possible highest value which could lead to a signal to clear a route too far ahead. Note that these functions always use the default value of SNCA as defined in sigscr.dat as starting value. Repeated calls of these functions will not lead to invalid or absurd values for SNCA.

INCREASE_SIGNALNUMCLEARAHEAD(n)
Increase the value of SNCA by n, starting from the default value.

DECREASE_SIGNALNUMCLEARAHEAD(n)
Decrease the value of SNCA by n, starting from the default value.

SET_SIGNALNUMCLEARAHEAD(n)
Set the value of SNCA to n.

RESET_SIGNALNUMCLEARAHEAD()
Reset the value of SNCA to the default value.

22.3.8 Local signal variables

Originally, the only means of interfacing between signals, or between signal heads within a signal, is through the signal aspect states. This sets a severe restriction of the amount of information that can be passed between signals or signal heads.

In OR, local signal variables have been introduced. These variables are specific for a signal. The variables are persistent, that is they do retain their value from one update to the next. Because the variables are assigned per signal, they are available to all signalheads which are part of that signal. The variables can also be accessed by other signals.

Each signalhead which is part of a signal can access the variables for both reading and writing.

Each signalhead from other signals can access the variables for reading only.

Each variable is identified by an integer number. The variables can contain integer values only.

STORE_LVAR(IDENT, VALUE)
Sets the variable as identified by IDENT to VALUE. The function has no return value.

THIS_SIG_LVAR(IDENT)
Returns the value of the variable identified by IDENT of this signal.

NEXT_SIG_LVAR(SIGFN_TYPE, IDENT)
Returns the value of the variable identified by IDENT of the first signal ahead of type SIGFN_TYPE.
If no such signal is found, the function returns value 0.

ID_SIG_LVAR(SIGID, IDENT)
Returns the value of the variable identified by IDENT of the signal identified by the signal ident SIGID.

22.3.9 Functions for Normal Head Subtype

Although there can be different types of signal, and OR allows the definition and additional of any number of type, only signals of type NORMAL will affect the trains.

Certain signal systems, however, have different types of signals, e.g. main and shunt signals, which require different behaviour or different response. In order to be able to distinguish between such signals, OR has introduced a Subtype which can be set for a NORMAL signal.

The subtype can be defined for any signal in the sigcfg.dat file, in the same way as the signal type. A number of functions is available to query a signal to identify its subtype.

THIS_SIG_HASNORMALSUBTYPE(SIGSUBTYPE)
Returns value 1 (true) if this signal has any head of type NORMAL with the required subtype.

NEXT_SIG_HASNORMALSUBTYPE(SIGSUBTYPE)
Returns value 1 (true) if next signal with any head of type NORMAL has any head with the required subtype.

ID_SIG_HASNORMALSUBTYPE(SIGIDENT, SIGSUBTYPE)
Returns value 1 (true) if signal identified by SIFIDENT has any head of type NORMAL with the required subtype.

22.3.10 Functions to verify full or partial route clearing

As mentioned, some signal systems differentiate between main and shunt signals (e.g. in Germany, UK). This may affect the clearing of a signal in locations where both such types occur on the same route. If a train requires the full route from a main signal to the next main signal, in locations where there are shunt signals inbetween, the first main signal may not clear until the full route to the next main signal is available, and will then clear to a main aspect. If, however, the train only requires a partial route (e.g. for shunting), the signal may clear as soon as (part of) this route is available, and will generally then clear only to a restricted or auxiliary aspect (shunt aspect).

The original MSTS signal functions could not support such a situation, as the signal would always clear as soon as the first part of the route became available, because it was not possible to distinguish between the different types of signal.

Due to the introduction of the Normal Subtype as detailed above, such a setup is not possible. A number of functions have been introduced to support this.

Use of these functions is, however, fairly complicated, and only a brief description of these functions is provided in this document.

TRAIN_REQUIRES_NEXT_SIGNAL(SIGIDENT, REQPOSITION)
Returns value 1 (true) if train requires the full route to the signal as identified by SIGIDENT.
If REQPOSITION is set to 0, the route is checked up to and including the last section ahead of the relevant signal.
If REQPOSITION is set to 1, the route is checked up to and including the first section immediately behind the relevant signal.

FIND_REQ_NORMAL_SIGNAL(SIGSUBTYPE)
Returns the Signal Ident of the first NORMAL signal which has a head with the required SIGSUBTYPE, or -1 if such a signal cannot be found.

ROUTE_CLEARED_TO_SIGNAL(SIGIDENT)
Returns value 1 (true) if the route as required is clear and available.

ROUTE_CLEARED_TO_SIGNAL_CALLON(SIGIDENT)
As ROUTE_CLEARED_TO_SIGNAL, but will also return value 1 (true) if the route is available because the train is allowed to call-on.

22.3.11 Miscellaneous functions

A number of miscellaneous functions which are not part of any of the groups detailed above.

ALLOW_CLEAR_TO_PARTIAL_ROUTE(SETTING)
If the route of a train passing a signal stops short of the next signal (no further NORMAL signal is found on that route), the relevant signal will only clear if the train is approaching that signal, i.e. it is the first signal in the train's path.
This setting can be overruled by this function.
If SETTING is set to 1, the signal will clear if required and the route is available, even if no further NORMAL signal is found.
If SETTING is set to 0, the normal working is restored.

THIS_SIG_NOUPDATE()
After the signal has been processed once, it will not be updated anymore. This is useful for fixed signals, e.g. at end of track like bufferstop lights, but also for fixed signals like route control or route information signals. Calling this function in the script for such signals excludes these signals from the normal updates which will save processing time.
Note that the signals are always processed once, so the script will be executed once to set the signal to the required fixed state.

SWITCHSTAND(ASPECT_STATE_0, ASPECT_STATE _1)
Special functions for signals used as switchstand. A direct link is set between the switch and the signal, such that the signal is immediately updated as and when the state of the switch is changed.
The signal will be set to ASPECT_STATE_0 when the switch is set to route 0, and to ASPET_STATE_1 when the switch is set to route 1. Linking the signal to the switch routes is not necessary.
Using this function for switchstands eliminates the delay which normally can occur between the change of the switch state and the state of the signal, due to the independent processing of the signal.
Note that the signal can be excluded from the normal update process as it will be updated through the direct link with the switch.

22.3.12 Timing Functions

These two functions allow time-triggered actions on signals, e.g. a fixed time-triggered delay on clearing etc..

Activate_Timing_Trigger() : activates a timing trigger.

Check_Timing_Trigger(n) : checks the timing trigger, and returns true if it was set more than n seconds ago.

22.4 OR-specific additions to SIGCFG files

Detailed below are OR-specific additions which can be set in the SIGCFG file to set specific characteristics or enhance the functionality of the signal types.

22.4.1 General definitions

The following are general definitions which must be set before the definitions of the signal types, immediately following the lighttextures and lightstab definitions.

22.4.2 ORTSSignalFunctions

Additional signal types can be defined in OR, over and above the standard MSTS signal types. The additional types must be predefined in the sigcfg.dat file using the ORTSSignalFunctions definition.

The defined ORTS signal types can be set in the signal type definition and used in signal script functions in the same way as the default MSTS types.

Note that SPEED is a fixed signal type which is available in OR without explicit definition (see below for details on SPEED type signals). Also note that any type definition starting with “OR_” is not valid, these names are reserved for future default types in OR.

Syntax:

ORTSSignalFunctions (n
    ORTSSignalFunctionType (“signaltype”)
    …
)

The value **n** indicates the total number of definitions.
The value **signaltype** is the name of the additional type.

22.4.3 ORTSNormalSubtypes

As detailed above, subtypes can be defined for NORMAL type signals which allows to distinguish different use of NORMAL type signals.

The Normal Subtype must be predefined in the sigcfg.dat file using the ORTSNormalSubtypes definition.

The Subtype can be set in the type definition for NORMAL type signals using the ORTSNormalSubtype statement, see below.

The subtype can be used in specific signal script functions as detailed above.

Syntax:

ORTSNormalSubtypes (n
    ORTSNormalSubtype (” subtype ”)
        …
)

The value **n** indicates the total number of definitions.
The value **subtype** is the name of the subtype.

22.4.4 Signal Type definitions

The following section details OR specific additions to the signal type definition.

Glow settings

Signal Glow is a feature in OR to improve the visibility of signals at larger distances. The required glow setting can de set per signal type in the signal definition. The value is a real number, and sets the intensity of the glow. Value 0.0 defines that there is no glow effect.

Default program values for glow are:

Day value = 3.0;
Night value = 5.0;

Notes :

- For signal types which have “Semaphore” flag set, the Day value = 0.0.
- For signals of type INFO and SHUNTING, both Day and Night value are set to 0.0 (no glow).

Syntax:

ORTSDayGlow (d)
ORTSNightGlow (n)

The values d and n are the day and night glow values, as real numbers.

22.4.5 Light switch

There were many signalling systems where semaphore signals did not show lights during daytime. This effect can be simulated using the ORTSDayLight setting.

Syntax:

ORTSDayLight(l)

The value l is a logical value, if set to false, the signal will not show lights during daytime.

22.4.6 Script Function

Normally, each signal type must have a linked signal script, with the same name as defined for the signal type. However, often there are a series of signal types which may differ in definition, e.g. due to differences in the position of the lights, but which have the same logic scripts.

In OR, a signal type can have a definition which references a particular script which this signal type must use. Different signal types which have the same logic can therefore all use the same script. This script may be defined using the name of one of these signal types, or it may have a generic name not linked to any existing signal type.

Syntax:

ORTSScript(name)

The value **name** is the name of the signal script as defined in the sigscr.dat file.

22.4.7 Normal Subtype

As detailed above, a signal type of type NORMAL may have an additional subtype definition.

Syntax:

ORTSNormalSubtype(subtype)

The value **subtype** is the subtype name and must match one of the names defined in ORTSNormalSubtypes.

22.4.8 Approach Control Settings

The required values for approach control functions for a particular signal type can be defined in the signal type definition. These values can be referenced in the signal script as defined for the approach control functions.

Syntax:

```
ApproachControlSettings (
    PositionDefinition ( position )
    SpeedDefinition ( speed )
)
```

Possible position definitions

Positionkm
Positionmiles
Positionm
Positionyd

Possible speed definitions

Speedkph
Speedmph

The value position is the required position value in dimension as set by the relevant parameter. The value speed is the required speed value in dimension as set by the relevant parameter. Inclusion of speed definition is optional and need not be set if only approach control position functions are used.

22.4.9 Signal aspect parameters

The following parameters can be included in signal aspect definitions.

or_speedreset

Can be used in combination with a speed setting. Its function, combined with the speed setting, is as follows.

In activity mode :

- If set, the speed as set applies until overruled by a speedpost or next signal setting a higher speed value;
- If not set, speed as set applies until the next signal and will not be overruled by a speedpost.

In timetable mode :

- Speed as set always applies until overruled by a speedpost or next signal setting a higher speed value, this flag has no effect in timetable mode.

or_nospeedreduction

For signal aspects “STOP_AND_PROCEED” and “RESTRICTING”, trains will reduce speed to a low value on approach of the signal. If this flag is set, trains are allowed to pass the signal at normal linespeed.

22.4.10 SPEED Signal

A new standard signaltype, “SPEED”, has been added to OR.

Signals defined as type “SPEED” are processed as speedposts and not as signals. The required speed limit can be set using the speed setting of the signal aspect definition.

The advantages of using “SPEED” signals over speedposts are :

- “SPEED” signals can be scripted, and can therefore be conditional, e.g. a speed restrictions in only set on approach to a junction if a restricted route is set through that junction.
- “SPEED” signals can set a state according to their setting, and this state can be seen by a preceeding signal. This can be used to set up variable speed warning signs.

A “SPEED” signalhead can be part of a signal which also contains other heads, but for clarity of operation this is not advisable.

Index

Symbols

_FUEL_COAL_, 165

A

Activation_Level, 226
AirBrakesAirCompressorPowerRating, 36
AMMETER, 304
ANIMATED_PARTS, 165
ANIMATED_PARTS1, 165
ANIMATED_PARTS2, 165
AWSMonitor, 157

B

BoilerVolume, 121
BrakeCutsPowerAtBrakeCylinderPressure, 157
BrakeCylinderPressureForMaxBrakeBrakeForce, 137
BrakeEquipmentType, 139
BrakePipeVolume, 135
BrakeSystemType, 139

C

CAB_RADIO, 291
CABVIEW3D, 304
CarSpawnerItem, 315
CarSpawnerList, 315
CentreOfGravity, 121
ChangeDownRPMpS, 101
ChangeUpRPMpS, 101
Class, 165
CLOCK, 304
Cooling, 101
CylinderDiameter, 121
CylinderStroke, 121

D

Delay, 107
Description, 165
Diesel, 101
DieselConsumptionTab, 101
DieselEngineType, 105
DieselPowerTab, 101
DieselTorqueTab, 101
DIGITAL_CLOCK, 302
direction, 58
DisplayMessage, 226
DoesBrakeCutPower, 157
Dummy, 129
DYNAMIC, 165
dynamic brake, 58
DynamicBrakesDelayTimeBeforeEngaging, 40

E

Elevator, 313
EmergencyStopMonitor, 157
EmptyCentreOfGravity_Y, 167
EmptyMaxBrakeForce, 167
EmptyMaxHandbrakeForce, 167
EmptyORTSDavis_A, 167
EmptyORTSDavis_B, 167
EmptyORTSDavis_C, 167
EmptyORTSDavisDragConstant, 167
EmptyORTSWagonFrontalArea, 167
Engine, 101
engine brake, 58
EngineBrakesControllerApplyStart, 137
EngineBrakesControllerReleaseStart, 137
EngineBrakesControllerRunningStart, 137
ESD_Alternative_Texture, 304
ESD_ORTSBellAnimationFPS, 183
ESD_ORTSSoundFileName, 320
EventCategoryLocation, 226, 228
EventCategoryTime, 226
Events, 228
EventTypeLocation, 226
ExhaustColor, 101
ExhaustDynamics, 101
ExhaustDynamicsDown, 101
ExhaustTransientColor, 101
EXTERNALWIPERS, 304

F

FileName, 165
final_fogDistance, 227
final_overcastFactor, 227
final_precipitationIntensity, 227
final_precipitationLiquidity, 227

Flip, 166
fog_transitionTime, 227
fps, 58
FreightAnimContinuous, 164, 167
FreightAnimStatic, 166, 169
FreightCoal, 165
FreightFuel, 165
FreightGeneral, 165
FreightGrain, 165
FreightGravel, 165
FreightLivestock, 165
FreightMilk, 165
FreightSand, 165
FreightWeight, 166, 169
FreightWeightWhenFull, 165, 167, 169
FUEL_GAUGE, 305
FuelCoal, 165
FuelDiesel, 165
FuelSand, 165
FuelWater, 165
FuelWood, 165
FullAtStart, 165, 167
FullCentreOfGravity_Y, 167, 169
FullMaxBrakeForce, 167, 169
FullMaxHandbrakeForce, 167, 169
FullORTSDavis_A, 167, 169
FullORTSDavis_B, 167, 169
FullORTSDavis_C, 167, 169
FullORTSDavisDragConstant, 167, 169
FullORTSWagonFrontalArea, 167, 169

G

GearBoxBackLoadForce, 105
GearBoxCoastingForce, 105
GearBoxEngineBraking, 105
gradient, 58
Graphic, 191, 192

I

IdleExhaust, 101
IdleRPM, 101
IdleTemperature, 101
IgnoreXRotation, 316
IntakePoint, 165, 167, 168
IsGondola, 164, 167
IsTenderRequired, 121

L

LEFTDOOR, 304
ListName, 315
LoadingScreen, 319
Location, 226
LPCylinderDiameter, 121
LPCylinderStroke, 121
LPNumCylinders, 121

M

MaxApplicationRate, 40, 137, 139
MaxBoilerPressure, 121
MaxBrakeForce, 137, 139
MaxContinuousForce, 100
MaxExhaust, 101
MaxForce, 100, 105, 107
MaxHeight, 165, 167, 168
MaximalPower, 101
MaxOilPressure, 101
MaxPower, 100, 105, 107
MaxReleaseRate, 40, 137, 139
MaxRPM, 101
MaxSteamHeatingPressure, 119, 121
MaxTenderCoalMass, 121
MaxTenderWaterMass, 121
MaxVelocity, 100, 105
MinHeight, 165, 167, 168
MinOilPressure, 101
MIRRORS, 304
MouseControl, 192
MSTSFreightAnimEnabled, 164, 166, 167, 169
MultiStateDisplay, 191

N

Name, 226
NextActivityObjectUID, 225
NIGHT, 304
NumCylinder, 121
NumFrames, 192
NumPaths, 312, 317

O

Offset, 169
OptTemperature, 101
ORTS3DCab, 304
ORTS3DCabFile, 304
ORTS3DCabHeadPos, 304
ORTS_2DEXTERNALWIPERS, 297
ORTS_BAILOFF, 291
ORTS_BATTERY_SWITCH_COMMAND_BUTTON_CLOSE, 288
ORTS_BATTERY_SWITCH_COMMAND_SWITCH, 288
ORTS_BATTERY_SWITCH_ON, 288
ORTS_BLOWDOWN_VALVE, 118
ORTS_CABLIGHT, 291
ORTS_CURRENT_CAB_IN_USE, 288
ORTS_ELECTRIC_TRAIN_SUPPLY_COMMAND_SWITCH, 289
ORTS_ELECTRIC_TRAIN_SUPPLY_ON, 289
ORTS_HELPERS_DIESEL_ENGINES, 289
ORTS_LEFTDOOR, 299
ORTS_MASTER_KEY, 288
ORTS_MIRRORS, 299
ORTS_OTHER_CAB_IN_USE, 288
ORTS_OVERCHARGE, 291
ORTS_PANTOGRAPH3, 107
ORTS_PANTOGRAPH4, 107
ORTS_PLAYER_DIESEL_ENGINE, 289
ORTS_PLAYER_DIESEL_ENGINE_STARTER, 290
ORTS_PLAYER_DIESEL_ENGINE_STATE, 290

ORTS_PLAYER_DIESEL_ENGINE_STOPPER, 290
ORTS_QUICKRELEASE, 291
ORTS_RIGHTDOOR, 299
ORTS_SERVICE_RETENTION_BUTTON, 289
ORTS_SERVICE_RETENTION_CANCELLATION_BUTTON, 289
ORTS_SIGNED_TRACTION_BRAKING, 292
ORTS_SIGNED_TRACTION_TOTAL_BRAKING, 292
ORTSActivitySound, 226
ORTSActSoundFile, 226, 229
ORTSAdhesion, 97
ORTSAICrossingHornPattern, 225
ORTSAIHornAtCrossings, 225
ORTSAlternatePassengerViewPoint, 183
ORTSAlternatePassengerViewPoints, 183
ORTSAngle, 302
ORTSAuxilaryResCapacity, 137
ORTSAuxPowerOnDelay, 107, 108, 189
ORTSAuxTenderWaterMass, 125
ORTSBattery, 155
ORTSBearingType, 121
ORTSBoilerEfficiency, 121
ORTSBoilerEvaporationRate, 121
ORTSBoilerSurfaceArea, 117
ORTSBrakeCylinderSize, 137
ORTSBrakeEmergencyTimeFactor, 135, 137
ORTSBrakePipeChargingRate, 135
ORTSBrakePipeQuickChargingRate, 135
ORTSBrakePipeTimeFactor, 135, 137
ORTSBrakeServiceTimeFactor, 135
ORTSBrakeShoeFriction, 131, 137
ORTSBurnRate, 121
ORTSCircuitBreaker, 108, 184, 188
ORTSCircuitBreakerClosingDelay, 108
ORTSContinue, 224, 226–228
ORTSContinuousForceTimeFactor, 140
ORTSCurtius_Kniffler, 97
ORTSCurveSMSNumber, 285
ORTSCurveSwitchSMSNumber, 285
ORTSCylinderBackPressure, 121
ORTSCylinderEfficiencyRate, 121
ORTSCylinderExhaustOpen, 121
ORTSCylinderInitialPressure, 121
ORTSCylinderPortOpening, 121
ORTSDavis_A, 95, 121
ORTSDavis_B, 95, 121
ORTSDavis_C, 95, 121
ORTSDavisDragConstant, 152, 169
ORTSDefaultTurntableSMS, 285
ORTSDelayToRestart, 228
ORTSDieselEngineMaxPower, 100
ORTSDieselEngines, 101
ORTSDirectAdmissionValve, 137
ORTSDoubleTunnelArea, 151
ORTSDoubleTunnelPerimeter, 151
OrtsDoubleWireEnabled, 318
OrtsDoubleWireHeight, 318
ORTSDriveWheelWeight, 121
OrtsDynamicBlendingForceMatch, 40
OrtsDynamicBlendingOverride, 40
ORTSElectricTrainSupply, 156
ORTSEmergencyValveActuationRate, 135
ORTSEngineBrakeApplicationRate, 135
ORTSEngineBrakeController, 184, 188
ORTSEngineBrakeReleaseRate, 135
ORTSEvaporationArea, 121
ORTSExternalSoundPassedThroughPercent, 286
ORTSFastVacuumExhauster, 137
ORTSFog, 227
ORTSfont, 302
ORTSFractionBoilerInsulated, 117
ORTSFreightAnims, 163, 164, 166, 167, 169
ORTSFuelCalorific, 121
ORTSGearedTractiveEffortFactor, 121
ORTSGrateArea, 121
ORTSHeatCoefficientInsulation, 117
ORTSHeatingBoilerFuelTankCapacity, 120
ORTSHeatingBoilerFuelUsage, 120
ORTSHeatingBoilerWaterTankCapacity, 120
ORTSHeatingBoilerWaterUsage, 120
ORTSHeatingCompartmentPipeAreaFactor, 119
ORTSHeatingCompartmentTemperatureSet, 119, 189
ORTSHeatingConnectingHoseInnerDiameter, 119
ORTSHeatingConnectingHoseOuterDiameter, 119
ORTSHeatingTrainPipeInnerDiameter, 119
ORTSHeatingTrainPipeOuterDiameter, 119
ORTSHeatingWindowDeratingFactor, 119
ORTSInertia, 97
ORTSLargeEjector, 137
ORTSListName, 316
ortsloadingscreenwide, 319
ORTSMainResChargingRate, 135
ORTSMainResPipeAuxResCharging, 135
ORTSMasterKey, 155
ORTSMatchingWPDelay, 228
ORTSMaxTractiveForceCurves, 105, 140, 154
ORTSMergeSpeed, 96
ORTSNumberBrakeCylinders, 137
ORTSOnOffTimeS, 321
ORTSOpenDoorsInAITrains, 319
ORTSOvercast, 227
ORTSPantographs, 107
ORTSPowerOnDelay, 107, 108, 189
ORTSPowerSupply, 184, 189
ORTSPowerSupplyAirConditioningPower, 189
ORTSPowerSupplyAirConditioningYield, 189
ORTSPowerSupplyContinuousPower, 189
ORTSPowerSupplyHeatingPower, 189
ORTSPrecipitationIntensity, 227
ORTSPrecipitationLiquidity, 227
ORTSRestartWaitingTrain, 228
ORTSRigidWheelBase, 96, 121, 143
ORTSSingleTunnelArea, 151
ORTSSingleTunnelPerimeter, 151
ORTSSlipControlSystem, 99

ORTSSlipWarningThreshold, 97
ORTSSmallEjector, 137
ORTSSoundFileName, 320
ORTSSoundLocation, 226
ORTSSpeedOfMaxContinuousForce, 100
ORTSStandstillFriction, 96
ORTSSteamBoilerType, 121
ORTSSteamFiremanMaxPossibleFiringRate, 121
ORTSSteamGearRatio, 121
ORTSSteamHeat, 119
ORTSSteamLocomotiveType, 121
ORTSSteamMaxGearPistonRate, 121
ORTSSuperheatArea, 121
ORTSSwitchSMSNumber, 285
ORTSTrackGauge, 96, 121, 143, 148
ORTSTrackSuperElevation, 321
ORTSTractionCharacteristics, 105
ORTSTractionCutOffRelay, 106, 184, 188
ORTSTractionCutOffRelayClosingDelay, 106
ORTSTractionMotorType, 108
ORTSTrailLocomotiveResistanceFactor, 152
ORTSTrainBrakeController, 184, 188
ORTSTrainBrakesControllerMaxOverchargePressure, 129
ORTSTrainBrakesControllerOverchargeEliminationRate, 129
ORTSTrainBrakesControllerSlowApplicationRate, 129
ORTSTrainControlSystem, 184, 190
ORTSTrainControlSystemParameters, 190
ORTSTrainControlSystemSound, 190
ORTSTriggeringTrain, 225
OrtsTriphaseEnabled, 318
ORTSUnbalancedSuperElevation, 121, 148
ORTSUnloadingSpeed, 100
ORTSUserPreferenceForestClearDistance, 308, 320
ORTSUserPreferenceRemoveForestTreesFromRoads, 320
ORTSVacuumBrakesExhausterRestartVacuum, 137
ORTSVacuumBrakesMainResChargingRate, 137
ORTSVacuumBrakesMainResMaxVacuum, 137
ORTSVacuumBrakesMainResVolume, 137
ORTSWagonFrontalArea, 152, 169
ORTSWagonSpecialType, 120
ORTSWaitingTrainToRestart, 228
ORTSWeatherChange, 227, 229
ORTSWheelSlipCausesThrottleDown, 99
Outcomes, 226, 228
overcast_transitionTime, 227
OverspeedMonitor, 157

P

Pantograph, 107
PassengerCabinHeadPos, 183
Pickup, 165
PickupType, 165
Position, 191
precipitationIntensity_transitionTime, 227
precipitationLiquidity_transitionTime, 227

R

RateOfChangeDownRPMpSS, 101
RateOfChangeUpRPMpSS, 101
RIGHTDOOR, 304
RotationLimit, 183, 304

S

SectionIdx, 312, 317
SectionSize, 317
SemaphorePos, 319
Shape, 165–169
SpecialMail, 165
speed, 58
SPEEDOMETER, 300, 304
StartDirection, 183, 304
StartingConfirmRPM, 101
StartingRPM, 101
StartTime, 77
State, 191
States, 191
SteamFiremanIsMechanicalStoker, 121
Style, 191, 192
SubType, 166, 169
SwitchVal, 191

T

TempTimeConstant, 101
throttle, 58
ThrottleRPMTab, 101, 105
tile, 335
time, 58
Tr_Activity, 228
Tr_Activity_File, 228
TrackSections, 312, 317
TrackShape, 312, 317
TrackShapes, 312, 317
TRACTION_BRAKING, 292
train brake, 58
TRAIN_BRAKE, 304
TrainBrakesControllerApplyStart, 129, 137
TrainBrakesControllerContinuousServiceStart, 129
TrainBrakesControllerEmergencyStart, 129, 137
TrainBrakesControllerEPApplyStart, 129
TrainBrakesControllerEPFullServiceStart, 129
TrainBrakesControllerEPHoldStart, 129
TrainBrakesControllerEPOnlyStart, 129
TrainBrakesControllerFullQuickReleaseStart, 129, 137
TrainBrakesControllerFullServiceStart, 129
TrainBrakesControllerGraduatedSelfLapLimitedHoldingStart, 129
TrainBrakesControllerGraduatedSelfLapLimitedStart, 129

TrainBrakesControllerHoldLappedStart, 129, 137
TrainBrakesControllerHoldStart, 129
TrainBrakesControllerManualBrakingStart, 139
TrainBrakesControllerMinimalReductionStart, 129
TrainBrakesControllerOverchargeStart, 129
TrainBrakesControllerReleaseStart, 129, 137
TrainBrakesControllerRunningStart, 129, 137
TrainBrakesControllerSelfLapStart, 129
TrainBrakesControllerSlowServiceStart, 129
TrainBrakesControllerSuppressionStart, 129
TrainBrakesControllerVaccumApplyContinuousService, 129
TrainBrakesControllerVaccumContinuousServiceStart, 129
TrainBrakesControllerVacuumContinuousServiceStart, 137
TrainPipeLeakRate, 137
Transfertable, 311
TriggerOnStop, 226
Turntable, 309, 311
TwoState, 192
Type, 191, 192

U

UNLOADINGPARTS, 165
UNLOADINGPARTS1, 165
UNLOADINGPARTS2, 165
UnloadingStartDelay, 164, 167

V

VacuumBrakesHasVacuumPump, 137
VacuumBrakesLargeEjectorUsageRate, 137
VacuumBrakesMinBoilerPressureMaxVacuum, 137
VacuumBrakesSmallEjectorUsageRate, 137
version, 58
VigilanceMonitor, 157
Visibility, 166

W

WagonEmptyWeight, 164, 166, 167, 169
WheelRadius, 121