System Scripting Variables and Events

Inhaltsverzeichnis

There exist many system variables in LOTUS. A script can read from and write to these variables, depending on the variable.

Some system variables are declared beforehand and do not need to be again declared at another location. Other system variables do have to be declared in the "PUBLIC VARS" section of the script.

Variables that do need to be declared are indicated with 'decl.' in the last column of the table. Variables that do not need to be declared again are indicated with ex (exists).

1 General

Variable name	Description	Type	Unit	Write access	Decl.
`Timegap`	Time past since last calculation step	Single	Seconds		ex
`NightTex`	Map wide variable for the control of night textures etc.	Integer	-		ex
`EnvirBrightness`	Brightness of environment	Single	0..1		ex
`DistrictLight`	State of the linked lighting district. Is no district linkes, the result will be 1.0 everytime.	Single			ex
`TimeOfDay`	Time of day	Single	0 = 0:00, 0.5 = 12:00, 1 = 24:00		decl.
`Date`	Date and Time in days since 30.12.1899, 00:00. Attention! Without Time!	Single	1 = one day		decl.

`Hint`	Hint text. Set this variable to a SpeechData-ID so the corresponding text is shown in the hint screen in LOTUS. You can use this to show the user, when not correctly operating the vehicle, why it is not possible. Convention is that user defined Speech-IDs are made up as follows: "[Vehicle name].Hint.[ShortText]" For example: "GT6N.Hint.ReverserNotSet" You then need to create your own SpeechData where every id (in the left column) is mapped to the corresponding text (in the chosen language) in the right column. By using multiple SpeechData files you can achieve hints in multiple languages.	String	-	X	decl.
`Mouse_X` & `Mouse_Y`	Movement of the mouse pointer since the last simulation step	Single	Pixel		decl
`RealisticElecSupply`	Indicator for the user setting "Realistic electricity supply"	Boolean	-		decl
`InitReadyForMovement`	Indicator for the user setting "Place Set Up Grade" 0 = Cold & Dark (e.g. Reverser at 0, pantograph lowered, main switch off, transformer off) 1 = Set up (e.g. Reverser at I or *, pantograph raised, main switch on, transformer on) 2 = Running (e.g. reverser in Forward)	Integer	0..2		decl
`InitPosInTrain`	Position of the wagon in the train immediately after placing the train (0 = front car).	Integer			decl
`InitCarIsReversed`	Indicates the direction of the wagon in the train after placing the train. False = wagon against the direction of the train, True = wagon along the direction of the train.	Boolean			decl
`InitUserPlaced`	Was the user placed in this car **the first time**? If the car is subsequently left or coupled, etc., this variable will still not change.	Integer	0 = no 1 = front 2 = back		decl
`DeadMansSwitch`	Indicator for the user setting "Simulate Dead Man's switch"	Boolean			decl
`mirrortex_{i}`	If mirrors (or similar) are defined, LOTUS stores the respective texture indices in the variables of this name (i = index of the mirror, starting at 0), so that they can be accessed in the material properties.	Integer	0...		decl

2 Remote-Control (AI-/Multiplayer, Stationary train)

The following variables are used for communication between Multiplayer and AI vehicles and the user vehicle (train). If a vehicle is the user vehicle then "RC_Active" is false. Then the variables can be set. This is also called "Sending mode", If the vehicle is not the user vehicle, e.g. a multiplayer or AI vehicle or a vehicle placed by the user, then "RC_Active" is true. The variables can then only be read; the vehicle is in "Receiving mode".

Variable name	Description	Type	Unit	Write access	Decl.
`RC_Active`	Train is in receiving/remote control mode. This indicates that at the moment this vehicle is not operated by a user. The vehicle is now a vehicle operated by another player (in Multiplayer) or an AI or stationary vehicle. If the variable is false the vehicle is a user vehicle and is in sending mode.	Boolean	-		decl
`RC_DirLeaver`	The sending vehicle can use this variable to indicate what wagon in a train is the "Master" (obviously when the reverser is moved out of the off position) and from which cabin this action was performed. Do note that this does not indicate what direction is set: - If the reverser is operated from the front cabin (or main controls) this variable must be set to 1, even if the reverser is set to reverse. - If the reverser is operated from the rear cabin (or auxiliary reverse driving console) this variable must be set to -1, even if the reverser is set to forward. If this variable is set nowhere in the entire train then: - for certain following variables (RC_Belling, RC_Cockpitlight) the position of the user in the sending vehicle is used. - all wagons receive 0 as variable value. If multiple wagons in a train set this variable then only the first (in the ordering of the train) is used. Other values are ignored.	Integer	1 = set in front cabin 0 = not set -1 = set in rear cabin (set means reverser in forward, reverse or neutral, NOT in off)	[if sending]	decl

`RC_Throttle`	Used for communicating the throttle setting in absolute value, that is, without considering the direction. Of the individual values in the wagons in the sending train the maximum is found and communicated for the entire train. This value is not used in the driving physics (the receiving train will anyway move in the same way as the sending vehicle). It is used for a correct simulation of especially sounds, but also animations and lights that vary based on the power.	Single	0..1	[if sending]	decl
`RC_Brake`	Used for communicating the brake setting. Use in the same way as RC_Throttle. Normalized to 0..1.	Single	0..1	[if sending]	decl
`RC_IndR`, `RC_IndL`	Indicator right/left. This variable shows whether the right and/or left indicator/turn signal is active. If both variables are true then the hazard indicators are on. Special about these variables is that single vehicles send and receive this variable taking into account their direction. As an example: In a train there are 2 vehicles running back to back. If the right indicators (in running direction) should turn on then in the forwards facing train the RC_IndR variable is set to true. In the backwards facing train RC_IndL is set to true. In sending mode it is sufficient for one vehicle to set the respective variable to true, the entire train is then interpreted as indicating. In receiving mode the variables of all vehicles are set.	Boolean	-	[if sending]	decl
`RC_FrontLight_Front` `RC_FrontLight_Back`	This variable is only transferred for the first and last part of a train and there only for the uncoupled end. In receiving mode the variable stays at 0 for the coupled ends.	Integer	0 = off 1 = daytime running lights 2 = dipped beam 3 = main beam	[if sending]	decl

`RC_OtherLights_Front` `RC_OtherLights_Back`	These variables can be used to set additional light at the front and back ends, like rear light, braking lights and fog lights. These variables are only transferred for the first and last part of a train and there only for the uncoupled end. In receiving mode the variables stay at 0 for the coupled ends.	Integer	Bit-Flags: +1 = Tail lights +2 = Braking lights +4 = Reversing lights +8 = Fog lights +16 = Rear fog lights +32 = free +64 = free +128 = free	[if sending]	decl
`RC_Belling`	State of the bell. In sending mode it is irrelevant what vehicle is sending the bell state. In receiving mode only the vehicle in which the reverser was set (or where the sending user is located) will get the new bell state. All others only receive 0.	Integer		[if sending]	decl
`RC_Sanding`	State of the sander. When at least one vehicle in a train is sanding then this state is transferred to the entire receiving train.	Boolean		[if sending]	decl
`RC_DoorsOpen_Left`, `RC_DoorsOpen_Right`	These variables are used to communicate the state of the doors. Each door is represented bitwise in an Integer for the left and right doors. For more about this see the Bit-Flags article. It is recommended to convey the "Goal" of the door movements. Not following this leads to the following issue: if you use as open state the formula "DoorOpen >0.5" then the closing for the multiplayer colleagues will only start when the doors for the player are already halfway closed. Especially with warnings before the closing of the doors this can lead to considerable delays. **Important: To let this variable work correctly you must indicate the number of doors in the "Object settings" in the Content-Tool!** This number refers to the number of doors per side, where the highest number is chosen: Value = Maximum(Doors on left side, Doors on right side).	Integer	Bit-Flags	[if sending]	decl

RC_CockpitLight	The user vehicle (or the vehicle with set reverser) in a sending train can indicate the setting of the lighting in the driving cabin.	Boolean		[if sending]	decl
RC_CabinLight	This variable uses Bit-Flags to allow for multiple (passenger) cabin lights (like upper and lower deck). When sending the state of a specific light source, it only needs to be set in one wagon. Then that light will turn on in the entire train (Bitwise OR). Max 8 bit.	Integer	Bit-Flags	[if sending]	decl
RC_PantographApplied	State of the pantographs of the vehicle.	Integer	Bit-Flags	[if sending]	decl
RC_Wiper	Setting of the wiper. Only read and written in the vehicle where the reverser is set (or where the user is located).		0: Off 1: Interval 2: Normal 3: Quick 4: Cleaning liquid 5 and bove: User defined		decl
RC_PIS_Line	Serves to transmit the described line number (0 to 65535), the special character code (0 to 4095) and offers still place for a freely usable value (0 to 15, see below table). Here a special coding takes place, which combines both values when sending and separates them again when receiving: Send: RC_PIS_Line := ({specchar} and $FFF) + ({line} and $FFFF) shl 12 + ({free value} and $F) shl 28; Receive: {specchar} := RC_PIS_Line and $FFF; {line} := (RC_PIS_Line shr 12) and $FFFF; {free value} := RC_PIS_Line shr 28; **Attention:** In timetables the special character is not stored; this leads to the fact that in AI vehicles the value {specchar} is not contained in this variable (= 0). In this case, the special character must be fetched using the PIS_GetRouteSpecialCharCode function via the route information.	Integer		[if sending]	decl

RC_PIS_Terminus	Serves to transmit the internal index of the displayed terminus (corresponds to "TERMINUS_LISTINDEX" of the standard modules of the base content vehicles, -1 if no valid route) and offers still place for a freely usable value (0 to 65535, see below table). If no free value is to be used, only the destination index must be transferred 1:1. Otherwise a special coding is done, which combines both values when sending and separates them again when receiving: Send: RC_PIS_Terminus := ({Zielindex} and $FFFF) + ({freier Wert} and $FFFF) shl 16; Receive: {terminusindex} := RC_PIS_Terminus - (RC_PIS_Terminus and $FFFF0000); {freevalue} := RC_PIS_Terminus shr 16; **Attention:** In the case of AI vehicles, it is usually the case that the number 65535 (or $FFFF) is transferred as the terminus index! In this case the AI would like that the terminus is to be taken by FIS file from line and route! Since the destination is dependent on the current stop, the RC variable RC_PIS_StopSeq is to be used, even if this is not yet written by the AI.	Integer	[if sending]	decl

RC_PIS_Route	Serves to transmit the internal index of the selected route (corresponds to "ROUTE_LISTINDEX" of the standard modules of the BaseContent vehicles, -1 if no valid route) and offers still place for a freely usable value (0 to 65535, see below table). If no free value is to be used, only the route index must be transferred 1:1. Otherwise a special coding is done, which combines both values when sending and separates them again when receiving: Send: RC_PIS_Route := ({routeindex} and $FFFF) + ({freevalue} and $FFFF) shl 16; Receive: {routeindex} := RC_PIS_Route - (RC_PIS_Route and $FFFF0000); {freevalue} := RC_PIS_Route shr 16;	Integer	[if sending]	decl
RC_PIS_StopSeq	Serves to transmit the index of the current stop within the selected route (corresponds to "STOP_SEQ" of the standard modules of the BaseContent vehicles, -1 if no valid route or stop) and still offers space for a freely usable value (0 to 65535, see below table). If no free value is to be used, only the stop index must be transferred 1:1. Otherwise a special coding is done, which combines both values when sending and separates them again when receiving: Send: RC_PIS_StopSeq:= ({stopseq} and $FFFF) + ({freevalue} and $FFFF) shl 16; Receive: {stopseq} := RC_PIS_StopSeq- (RC_PIS_StopSeqand $FFFF0000); {freevalue} := RC_PIS_StopSeq shr 16;	Integer	[if sending]	decl

2.1 About the RC_PIS variables

Like all other RC variables, they have a dual role, which is particularly important here: On the one hand, they serve to transmit the PIS state between the multiplayer vehicles, i.e. the user vehicle sends its PIS state to the server and the server distributes this information to the vehicles of the other players; on the other hand, they serve to communicate between the AI controller, the visual vehicle and - depending on the variable -

also the passengers. For this reason, there is the division into a fixed value and a free value described in more detail above. The free value is used to transmit optional state values that go beyond the scope of those in the fixed value. Nevertheless, it must not be forgotten that the AI does not supply these optional state values! I.e. an AI vehicle receives only the fixed values, the free values are always 0. Nevertheless, also in this case must be described sensibly! Conversely, the passengers expect valid values among the fixed values! They cannot process the optional values. However, at the moment - and probably also in the future - the passengers are limited to the variable `RC_PIS_Terminus`.

Vehicles

2.2 General

Variable name	Description	Type	Unit	Write access	Decl.
`v_ground`	Speed in the direction along the vehicle	Single	Meters per second		decl
`a_ground`	Acceleration in the direction along the vehicle	Single	Meters per square second		decl
`panto_voltage_{a}`	Catenary voltage available at the pantograph with index `{a}`. **Trolley**: trolley is connected with a catenary wire and this wire is powered. **Pantograph:** The lowest catenary wire (the height if which is passed through) is powered. Whether the pantograph itself is raised must be checked by the script.	boolean	-		decl
`panto_{a}`	Height of the lowest catenary wire reachable by the (single or double) pantograph with index `{a}`. The result is represented in the coordinate system of the vehicle object plus the movement state of the associated animation.	Single	m		decl
`trolley_angle_{a}_hori`, `trolley_angle_{a}_vert`	Displacement of the trolley rod with index `{a}` horizontally compared to the middle position (_hori) or vertically compared to the horizon (_vert) of the vehicle object plus the movement state of the associated animation. You can only write as long at the trolley rod is connected to the wire. You can use this to implement protection mechanisms against the derailment of the trolley rod.	Single	Degrees(°)	[X]	decl

`trolley_free_{a}`	Trolley with index `{a}` is free: it is not connected to a trolley wire nor is it being moved by a user.	Boolean	-		decl
`trolley_online_{a}`	Trolley with index `{a}` is connected to the trolley wire.	Boolean	-		decl
`TextureRaindropSet_{x}`	Texture index of the rain drop window with index `{x}` (0 based).	Integer	-		ex
`coupled_{a}`	Coupled at the front (a = 0) or rear (a = 1). Is true when another vehicle is coupled and the coupling is secure.	Boolean	-		ex
`couplingState_{a}`	Coupling state: 0: Coupler deactivated, when attempting to couple nothing happens, "only buffers" 1: Coupler ready, when driving up coupling will take place - Automatic coupler 2: Coupled Front: a = 0, Rear: a = 1.	Integer	-	[X]	ex
`couplingOffsetY_{a}`	Temporarily move the coupler in the Y direction. E.g. on the GT6N when the couplers are being retracted. Front: a = 0, rear: a = 1.	Single	-	[X]	decl

2.3 Rail

Variable name	Description	Type	Unit	Write access	Decl.
`M_Axle_N_{b}_{a}`	Traction force of the wheel at the point of contact with the rail. This also includes forces from the electronic brake. `{b}` = Index of the bogie, `{a}` = Index of the axle, all 0 based.	Single	Newton	X	decl
`MBrake_Axle_N_{b}_{a}`	Braking force of the wheel at the point of contact with the rail. Only positive values; these work against the driving direction and are able to stop and hold a train on a gradient. `{b}` = Index of the bogie, `{a}` = Index of the axle, all 0 based.	Single	Newton	X	decl

`sanding_{b}_{a}`	Sanding. When set to true this axles is being sanded. `{b}` = Index of the bogie, `{a}` = Index of the axle, all 0 based.	Boolean	-	X	decl
`F_RailBrake_Bogie_N_{b}`	Contact force of a rail brake on bogie number `{b}` (0 based). The effect depends on the current coefficient of friction of the rails. Always works against the direction of travel and able to stop and hold the train on a gradient.	Single	Newton	X	decl
`v_Axle_mps_{b}_{a}`	Speed of the wheel at point of contact with the rails. If the wheel is not blocking or spinning this is equal to the speed of the vehicle. `{b}` = Index of the bogie, `{a}` = Index of the axle, 0 based.	Single	Meters per second		ex
`alpha_Axle_deg_{b}_{a}`	Angle or rotation of the axle. `{b}` = Index of the bogie, `{a}` = Index of the axle, 0 based.	Single	Degrees (°)		ex
`spring_Axle_m_{b}_{a}`	Actual deflection of the axle suspension (primary suspension) from the zero position. The zero position if the position of the spring when the wagon stands quietly on a horizontal track without load. `{b}` = Index of the bogie, `{a}` = Index of the axle, 0 based.	Single	Meter		ex
`loadforce_Axle_N_{b}_{a}`	Actual contact force of an axle on the rails. `{b}` = Index of the bogie, `{a}` = Index of the axle, 0 based.	Single	Newton		decl
`invradius_abs_max`	Maximal curvature of the track section underneath the axles of this vehicle. The curvature is the reciprocal of the radius: 200m radius = 0.005 curvature, 100m radius = 0.01 curvature	Single	1 / m		decl
`invradius_{b}_{a}`	Current curvature of the track section under an axle. `{b}` = Index of the bogie, `{a}` = Index of the axle, 0 based.	Single	1 / m		decl

railquality_{b}_{a}	Track alignment and quality under the axle: ({b} = index of the bogie, {a} = index of the axle, each based on 0) 0 = Even profile 1 = uneven profile 2 = with frogs, even 3 = with frogs, uneven 4 = Flat groove 5 = Very uniform profile (e.g. high-speed line) 6 = Dirty with even profile 7 = Dirty with uneven profile	Integer	-	decl
surfacetype_{b}_{a}	Surface type underneath the track ({b} = Index of the bogie, {a} = Index of the axle, 0 based). 0 = Normal (Gravel) 1 = Street 2 = Grass	Integer	-	decl
V_ThirdRailCollector_{b}_{L/R}	It is checked whether there is a third rail next to the bogie with the index {b} on the side L and/or R and what voltage is present there.	Single	-1.0: next to the bogie is no third rail -0.5: next to the bogie is partial a third rail 0: Third rail next to bogie, no voltage 1: Third rail next to bogie, normal voltage	decl

Note: when replacing the indices in the variable names you also remove the accolades! Example: loadforce_Axle_N_{b}_{a} on the first bogie, second axle is loadforce_Axle_N_0_1.

2.4 Street

Variablenname	Beschreibung	Typ	Einheit	Schreibzugriff	Dekl.
Steering	Steering angle, related to the maximum steering angle	Single	-1 (full L) ... +1 (full R)	X	decl

Variablenname	Beschreibung	Typ	Einheit	Schreibzugriff	Dekl.
M_Axle_N_{a}	Traction force of the wheels of an axle at the contact point with the road. This includes the forces of the engine and/or gearbox brake. {a} = Index of the axle, 0 based.	Single	Newton	X	decl
MBrake_Wheel_N_{a}_{s}	Braking force of a wheel of an axle at the contact surface with the road. Only positive values; these always act against the direction of travel and have the ability to hold the vehicle on a gradient. {a} =Index of the axle, 0 based, {s} = side, 0 = L, 1 = R	Single	Newton	X	decl
v_Wheel_mps_{a}_{s}	Speed of the wheel at its contact with the road. If the wheel is neither locked nor spinning, this speed is the vehicle speed. {a} = Index of the axle, 0 based, {s} = side, 0 = L, 1 = R	Single	Meter per second		ex
alpha_Wheel_deg_{a}_{s}	Rotation angle of the wheel. {a} = Index of the axle, 0 based, 0-basiert, {s} = side, 0 = L, 1 = R	Single	Degrees (°)		ex
spring_Wheel_m_{a}_{s}	Current deflection of the wheel suspension from the zero position. The zero position is the position in which the spring is when the car is fully raised so that the wheel no longer touches the ground. {a} = Index of the axle, 0 based, {s} = side, 0 = L, 1 = R	Single	Meter		ex
steering_Wheel_m_{a}_{s}	Angle of rotation of the wheel due to the steering angle around the vertical axis. {a} = Index of the axle, 0 based, {s} = side, 0 = L, 1 = R	Single	Degrees (°), >0 = R		ex

Note: when replacing the indices in the variable names you also remove the accolades! Example: `MBrake_Wheel_N_{b}_{a}` on the first axle, right side is `MBrake_Wheel_N_0_1`.

3 Scenery Objects

Variable name	Description	Type	Unit	Write access	Decl.

`trafficlight_phase`	Current phase of the traffic light system and direction: 0..2 = three different red phases 3..5 = three different yellow phases for red to green 6..8 = three different green phases 9..11 = three different yellow phases for green to red. 12 = traffic light disabled for the primary (priority) direction 13 = traffic light disabled for the secondary (yield priority) direction 14..16 = three different phases for other purposes	Integer	-	decl

4 System Events

The underlying system for the System Events is understandable as follow: There are certain internal LOTUS events that should be triggered from within the vehicle. These triggers should not only be triggered by a keyboard shortcut but also by using mouse click hotspots in the vehicles. At the moment System Events are not listed when you want to assign a mouse event to a vehicle mesh. Instead System Events have to be typed in manually.

Event-Name	Function
`TrolleyRerail_{a}`	Used to attaching (again) the trolley rod to the overhead wires. For this the marked click hotspot is clicked, the mouse button is held down and moved to turn and raise/lower the trolley rod. At the wanted position the mouse button is released.

5 System Send Events

A script can also trigger certain events in LOTUS. For this there is `procedure SendEvent(self: integer, id: string, value: integer)`. The first parameter must always be the Public Var "Self" (declared automatically by LOTUS). The following IDs can be used:

Event-ID	Function
`TrolleyDerail`	Derails the trolley rod "on purpose". For value fill out the index of the current collector that should be derailed.