

# HOW TO USE

# Train Simulator

# Engine Scripting

1.	Overview .....	1
2.	Engine and Wagon scripts .....	2
3.	Calling Convention.....	3
4.	Performance.....	3
5.	Function Reference .....	4

## 1. Overview

Many entities in the Train Simulator world are enhanced using LUA scripting. This allows dynamic functionality to be generated for engines and signals. This document is an API reference for the functions available in Dovetail Train Simulator.

## 2. Engine and Wagon scripts

Any engine or wagon can have a LUA script specified in its blueprint. This script normally has a number of the following functions.

**Initialise ()** *Called before the game begins*

**Update (frameTime)**

*Called once per frame if the script request updates (see below)*

*frameTime: Time passed in seconds since the previous frame*

**OnControlValueChange (name, index, value)**

*Called when a control is being altered by the player or engine sim.*

*name: Name of the control*

*index: Unused - always 0*

*value: The value the control wants to be set to*

**OnConsistMessage (message, arg, direction)**

*Called by the **SendConsistMessage** fn. A way of sending information along a consist.*

*message: The ID of the message*

*arg: String argument passed*

*direction: 0 = from behind, 1 = from in front*

**OnCameraEnter (cabEndWithCamera, carriageCam)**

*Called when the camera enters the cabview or carriage view.*

*cabEndWithCamera: 0 = none, 1 = front, 2 = back.*

*carriageCam: 0 if cab cam, 1 if carriage cam*

**OnCameraLeave ()** *Called when the camera leaves to an external cam*

**OnSave ()** *Called when the game is saved*

**OnResume ()** *Called when the game is resumed*

## 3. Calling Convention

To call a code function in LUA, the convention is to use the **Call** function.

```
Call( "SetControlValue", name, index, value );
```

To call a function on a child object:

```
Call( "Main Smoke Stack:SetEmitterColour", r, g, b);
```

The name is first and function is separated by a single ":"

To access the first matching child a wildcard "\*" can be used.

## 4. Performance

Engine scripts have some performance overhead and can (if used in-judicially) have a negative impact on frame rate. To avoid this, here are some guidelines:

Each call into the API has a small overhead (roughly 0.04ms). Don't repeatedly call the same function to get a value from the API. Instead, store the value locally for the duration of the update.

All locos will get an update call even if: They are not the player; They are not active; or They are not near the camera. For visual elements such as smoke linked to tractive effort, the update should check the locality of the engine using **getNearPosition** or **GetIsPlayer** and **GetIsDeadEngine** to check if this is an AI engine.

Similarly for any ancillary controls only the player can set, do not check these in the update loop for AI locos.

Messages sent using **SendConsistMessage** are especially expensive, as it requires chains of calls. This should be avoided where possible and done only for the player consist. Where this is used to send values to simulate jumper cables, you should only send the values when they change, not every frame.

Only use the ":" notation to access features of child entities, in particular, using "\*:" notation leads to a comprehensive search of all children with an associated cost.

e.g. `Call("*:GetControlValue", "SpeedometerMPH", 0)` the "\*:" is not required

Make all variables "local" where possible. These are then created on the stack rather than the heap.

e.g. `local isPlayer = Call("GetIsPlayer")`

Any variable without the "local" prefix becomes a global, and this incurs a small performance penalty. On the plus side, using local variables makes scripts easier to debug.

Here is an example to test the visible distance of a loco:

```
function isNearCamera ()  
    x, y, z = Call ( "getNearPosition" );  
    return (x * x + y * y + z * z) < (2000 * 2000); -- within 2km of camera  
end -- function isNearCamera ()
```

## 5. Function Reference

This section contains details of available functions, separated into their respective modules.

### 5.1. ScriptComponent

These functions are available only to scripted entities.

#### *BeginUpdate()*

**Function:** Request script to get update call once per frame

**Arguments:** N/A

**Returns:** N/A

#### *EndUpdate()*

**Function:** Request script to end update call once per frame

**Arguments:** N/A

**Returns:** N/A

#### *GetSimulationTime()*

**Function:** Get the simulation time in seconds

**Arguments:** N/A

**Returns:** the simulation time in seconds

#### *IsExpertMode()*

**Function:** Is the game in expert mode controls

**Arguments:** N/A

**Returns:** TRUE (1) if the controls are in expert mode

### 5.2. PosOri

These functions are related to the position and orientation of a component.

#### *getNearPosition()*

**Function:** Get the position in the current world frame of the object (local coordinates are local to a moving origin centred on the cameras current tile)

**Arguments:** N/A

**Returns:** the position x, y, z in metres relative to the origin

### **5.3. RailVehicleComponent**

These functions apply the the base class of all rail vehicles.

#### ***GetIsPlayer()***

**Function:** Is the rail vehicle player controlled

**Arguments:** N/A

**Returns:** TRUE (1) if the train is player controlled

#### ***GetSpeed()***

**Function:** Get the rail vehicles speed

**Arguments:** N/A

**Returns:** the speed in m/s

#### ***GetAcceleration()***

**Function:** Get the rail vehicles acceleration

**Arguments:** N/A

**Returns:** the acceleration in m/s<sup>2</sup>

#### ***GetTotalMass()***

**Function:** Get the total mass of the rail vehicle including cargo

**Arguments:** N/A

**Returns:** the mass in Kg

#### ***GetConsistTotalMass()***

**Function:** Get the total mass of the entire consist including cargo

**Arguments:** N/A

**Returns:** the mass in Kg

#### ***GetConsistLength()***

**Function:** Get the consist length in metres

**Arguments:** N/A

**Returns:** length in metres

#### ***GetGradient()***

**Function:** Get the gradient at the front of the consist

**Arguments:** N/A

**Returns:** the gradient as a percentage

#### ***GetRVNumber()***

**Function:** Get the Rail Vehicle number for the engine

**Arguments:** N/A

**Returns:** The rail vehicle number

### ***SetRVNumber(rvNo)***

**Function:** Set the Rail Vehicle number (used for changing the destination boards)

**Arguments:** rvNo - the new rail vehicle no

**Returns:** N/A

### ***GetCurvature()***

**Function:** Get the curvature (radius of curve) at the front of the consist

**Arguments:** N/A

**Returns:** the radius of the curve  $m^{-1}$

### ***SendConsistMessage(message, arg, dir)***

**Function:** Send a message to the next or previous rail vehicle in the consist. Calls the script function: *OnConsistMessage(message, arg, dir)* in the next or previous rail vehicle

**Arguments:** *message* - the ID of a message to send (IDs 0 - 100 are reserved, please use ids > 100)

*arg* - a textual argument

*dir* - the direction to send the message:

0 - to rail vehicle in front

1 - to rail vehicle behind

**Returns:** 1 if there was a next/previous rail vehicle

### ***GetCurvatureAhead(displacement)***

**Function:** Get the curvature relative to the front of the vehicle.

**Arguments:** *displacement* - if positive, gets curvature this number of meters ahead of the front of the vehicle. If negative, gets curvature this number of meters behind the rear of the vehicle.

**Returns:** the radius of the curve  $m^{-1}$ , positive if curving to the right, negative if curving to the left, relative to the way the vehicle is facing.

### ***SetBrakeFailureValue(name, value)***

**Function:** Set a failure value on the train brake system for this vehicle

**Arguments:** *name* - the name of the failure type one of:

"BRAKE\_FADE" - The proportion of brake power lost due to fade in the braking pads due to heat

"BRAKE\_LOCK" - The proportion of max force the brake is stuck at due to the pad locking on the wheel

*value* - the value of the failure dependent on failure type

**Returns:** N/A

## **GetNextRestrictiveSignal([direction = 0], [minDistance = 0], [maxDistance = 10000])**

**Function:** Get the next restrictive signal's distance and state

**Arguments:** (optional) *direction* - 0 = forwards, 1 = backwards (def: 0)

(optional) *minDistance* - how far ahead to start searching (def: 0m)

(optional) *maxDistance* - how far ahead to stop searching (def: 10,000m)

**Returns:** param1 - result: -1 = nothing found, 0 = end of track, > 0 signal found

param2 - basic signal state: -1 = invalid, 1 = yellow, 2 = red

param3 - distance (m) to signal

param4 - 2d map's "pro" signal state for more detailed aspect info.

-1 = invalid, 1 = yellow, 2 = double yellow, 3 = red, 10 = flashing yellow

11 = double flashing yellow

### **Example:**

```
local result, state, distance, proState = Call("GetNextRestrictiveSignal")
if result <= 0 then
    Print("No restrictive signals ahead!")
else
    Print("Restrictive signal state: " .. state .. ", Distance:" .. distance .. ", Pro
          State:" .. proState)
end
```

**Remarks:** If a signal is within 1cm of minDistance, it is ignored. This is to compensate for floating point rounding errors and to help prevent an infinite loop being stuck at the same signal in a while loop.

If calling GetNextRestrictiveSignal iteratively, based on the last call's distance, you should include a check to make sure the last and new distances are not equal. If they are equal, add 1cm to the next minDistance. This will also prevent an infinite loop.

## **GetNextSpeedLimit([direction = 0], [minDistance = 0], [maxDistance = 10000])**

**Function:** Get the next speed limit's distance and restriction (both speed signs and signal link speed limits)

**Arguments:** (optional) *direction* - 0 = forwards, 1 = backwards (def: 0)

(optional) *minDistance* - how far ahead to start searching (def: 0m)

(optional) *maxDistance* - how far ahead to stop searching (def: 10,000m)

**Returns:** param1 - result: -1 = nothing found, 0 = end of track, 1 = track speed limit (no signage), 2 = track speed limit sign, 3 = signal speed limit

param2 - restriction (m/s)

param3 - distance (m) to speed limit

### **Example:**

```
local limitType, limit, distance = Call("GetNextSpeedLimit")
if limitType == -1 then
    Print("No speed limit ahead!")
elseif limitType == 0 then
    Print("End of track: " .. distance)
else
    Print("Speed limit: " .. limit .. ", Distance:" .. distance .. ", Type: " .. limitType)
end
```

**Remarks:** If a speed limit is within 1cm of minDistance, it is ignored. This is to compensate for floating point rounding errors and to help prevent an infinite loop being stuck at the same speed limit in a while loop.

If calling GetNextSpeedLimit iteratively, based on the last call's distance, you should include a check to make sure the last and new distances are not equal. If they are equal, add 1cm to the next minDistance. This will also prevent an infinite loop.

### **GetCurrentSpeedLimit([separateComponents = 0])**

**Function:** Get the current speed limit for the consist.

**Arguments:** (optional) *separateComponents* - 0 = return current limit, 1 = return separate track and signal limit

**Returns:** if *separateComponents* is 0 or the parameter is omitted, then a single value for the current limit is returned. Otherwise, 2 values are returned for track and signal limits respectively.

**Example:**

– For a single combined limit

```
local currentLimit = Call("GetCurrentSpeedLimit")
Print("Current limit is " .. currentLimit)
```

– For separate limits:

```
local trackLimit, signalLimit = Call("GetCurrentSpeedLimit", 1)
Print("Track limit is " .. trackLimit .. ", Signal limit is " .. signalLimit)
```

– The minimum of *trackLimit* and *signalLimit* will be equal to *currentLimit*

### **GetConsistType**

**Function:** Get the type of consist.

**Arguments:** N/A

**Returns:** eTrainTypeSpecial = 0,  
eTrainTypeLightEngine = 1,  
eTrainTypeExpressPassenger = 2,  
eTrainTypeStoppingPassenger = 3,  
eTrainTypeHighSpeedFreight = 4,  
eTrainTypeExpressFreight = 5,  
eTrainTypeStandardFreight = 6,  
eTrainTypeLowSpeedFreight = 7,  
eTrainTypeOtherFreight = 8,  
eTrainTypeEmptyStock = 9,  
eTrainTypeInternational = 10,

### **GetIsNearCamera**

**Function:** Return true if the camera is near this vehicle ( < 4km )

**Arguments:** N/A

**Returns:** True if near

### **GetIsInTunnel**

**Function:** Return true if the rail vehicle is in a tunnel

**Arguments:** N/A

**Returns:** True if in a tunnel

## 5.4. RenderComponent

These functions relate to the RenderComponent which encompasses the model, nodes and animations.

### **ActivateNode(*nodeName*, *activate*)**

**Function:** Activate/Deactivate a node in a model

**Arguments:** *nodeName* - name of the node ("all" for all nodes)

*activate* - 1 show, 0 hide

**Returns:** N/A

## 5.5. AnimObjectRender

These functions relate to the AnimObjectRender of the RenderComponent.

### **AddTime(*animName*, *time*)**

**Function:** add time to an animation

**Arguments:** *animName* - name of the animation

*time* - the amount of time in seconds +ve or -ve

**Returns:** the time remainder

### **Reset(*animName*)**

**Function:** Reset an animation

**Arguments:** *animName* - name of the animation

**Returns:** N/A

### **SetTime(*animName*, *time*)**

**Function:** set the time of an animation

**Arguments:** *animName* - name of the animation

*time* - the amount of time in seconds +ve or -ve

**Returns:** the time remainder

## 5.6. SoundComponent

These functions are related to the SoundComponent aspect of rail vehicles.

### **SetParameter(*paramName*, *value*)**

**Function:** Set a parameter on a audio proxy

**Arguments:** *paramName* - name of the parameter

*value* - the value

**Returns:** N/A

## 5.7. ControlContainer

These functions are related to the ControlContainer aspect of rail vehicles.

### ***ControlExists(controlName, index)***

**Function:** Does a control with a name exist

**Arguments:** *controlName* - the name of the control

*index* - the index of the control (usually 0 unless multiple controls with same name)

**Returns:** TRUE if the control exists

### ***GetControlValue(controlName, index)***

**Function:** Get the value of a control

**Arguments:** *controlName* - the name of the control

*index* - the index of the control (usually 0 unless multiple controls with same name)

**Returns:** The value for the control

### ***SetControlValue(controlName, index, value)***

**Function:** Set the value for a control

**Arguments:** *controlName* - the name of the control

*index* - the index of the control (usually 0 unless multiple controls with same name)

**Returns:** N/A

### ***GetControlMinimum(controlName, index)***

**Function:** Get the minimum value for a control

**Arguments:** *controlName* - the name of the control

*index* - the index of the control (usually 0 unless multiple controls with same name)

**Returns:** The controls minimum value

### ***GetControlMaximum(controlName, index)***

**Function:** Get the maximum value for a control

**Arguments:** *controlName* - the name of the control

*index* - the index of the control (usually 0 unless multiple controls with same name)

**Returns:** The controls maximum value

### ***GetWiperValue(pairIndex, aOrB)***

**Function:** Get the normalised value of a wiper animation current frame

**Argument:** *pairIndex* - Which wiper pair to get

*aOrB* - Which wiper of the pair to get the value of

**Returns:** value between 0.0 and 1.0 of the wiper's current position in the animation

### ***SetWiperValue(pairIndex, aOrB, value)***

**Function:** Set the normalised value of a wiper's animation

**Argument:** *pairIndex* - Which wiper pair to get

*aOrB* - Which wiper of the pair to get the value of

*value* - Value to set the wiper to

**Returns:** N/A

### ***GetWiperPairCount()***

**Function:** Get the number of wiper pairs this control container has

**Argument:** N/A

**Returns:** Number of wiper pairs in the control container

### ***IsControlLocked()***

**Function:** Get whether or not a control is locked

**Arguments:** N/A

**Returns:** 0 if unlocked, 1 if locked

### ***LockControl(controlName, index, locked)***

**Function:** lock a control so the user can no longer effect it E.g to simulate a failure

**Arguments:** *controlName* - the name of the control

*index* - the index of the control (usually 0 unless multiple controls with same name)

*locked* - TRUE/FALSE lock or unlock control

**Returns:** N/A

## **5.8. Engine**

These functions are also available to Engines specifically.

### ***GetTractiveEffort()***

**Function:** Get the proportion of tractive effort being used

**Arguments:** N/A

**Returns:** proportion 0 - 100% of tractive effort

### ***GetIsEngineWithKey()***

**Function:** Is this the player controlled primary engine

**Arguments:** N/A

**Returns:** TRUE if this is the engine the player is controlling

### ***GetIsDeadEngine()***

**Function:** Is this engine broken/disabled

**Arguments:** N/A

**Returns:** TRUE if the engine is broken/disabled

### ***SetPowerProportion(index, value)***

**Function:** Set the proportion of normal power a diesel unit should output

**Arguments:** *index* - the index of the power unit (-1 for all power units)

*values* - the proportion of normal power output 0.0 - 1.0

**Returns:** N/A

### ***GetFireboxMass()***

**Function:** Get the proportion of full firebox mass

**Arguments:** N/A

**Returns:** The mass of the firebox as a proportion of max in the range 0.0 - 1.0

## **5.9. EmitterComponent**

These functions are related to particle emitter aspects involved in rail vehicles.

### ***SetEmitterColour(r, g, b, [a])***

**Function:** Set the emitters colour multiplier

**Arguments:** *r, g, b, [a]* - Red, green and blue components, and optionally alpha.

**Returns:** N/A

### ***SetEmitterRate(rate)***

**Function:** Set the emitter rate multiplier

**Arguments:** *rate* - the rate (by default 1)

**Returns:** N/A

### ***SetEmitterActive(active)***

**Function:** Activate an emitter

**Arguments:** *active* - 1 activate, 0 deactivate

**Returns:** N/A

### ***GetEmitterColour()***

**Function:** Get the emitter colour

**Arguments:** N/A

**Returns:** colour in RGBA with components r, g, b, a

### ***GetEmitterRate()***

**Function:** Get the emitter rate multiplier 1.0 is default 0.0 is no emission

**Arguments:** N/A

**Returns:** the emitter rate

### ***GetEmitterActive()***

**Function:** Is the emitter active

**Arguments:** N/A

**Returns:** TRUE if active

### ***RestartEmitter()***

**Function:** Restart the emitter

**Arguments:** N/A

**Returns:** N/A

### ***SetInitialVelocityMultiplier(multiplier)***

**Function:** Multiply the initial velocity by a given value. Default value is 1.0

**Arguments:** *multiplier* - Multiplier to scale XYZ velocity components

**Returns:** N/A

## **5.10. LightComponent**

These functions concern the operation of Spot and Point lights on rail vehicles.

### ***Activate(activate)***

**Function:** Turn the light on or off.

**Arguments:** *activate* - on/off

**Returns:** N/A

### ***SetColour(rgb)***

**Function:** Set the colour of the light

**Arguments:** *r,g,b* - the red, green and blue components of the colour

**Returns:** N/A

### ***GetColour()***

**Function:** Get the colour of the light

**Arguments:** N/A

**Returns:** *r,g,b* - the red, green and blue components of the colour

### ***SetRange(range)***

**Function:** Set the range of the light

**Arguments:** *range* - the range of the light in metres

**Returns:** N/A

### ***GetRange()***

**Function:** Get the range of the light

**Arguments:** N/A

**Returns:** the range of the light in metres

### ***SetUmbraAngle(*umbra*)***

**Function:** Set the umbra of a spot light

**Arguments:** *umbra* - the angle of the outer cone in degrees

**Returns:** N/A

### ***GetUmbraAngle()***

**Function:** Get the Umbra of a spot light

**Arguments:** N/A

**Returns:** the angle of the outer cone in degrees

### ***SetPenumbraAngle(*penumbra*)***

**Function:** Set the Penumbra of a spot light

**Arguments:** *penumbra* - the angle of the inner cone in degrees

**Returns:** N/A

### ***GetPenumbraAngle()***

**Function:** Get the penumbra of a spot light

**Arguments:** N/A

**Returns:** the angle of the inner cone in degrees