EEP Lua Automatic Train Control

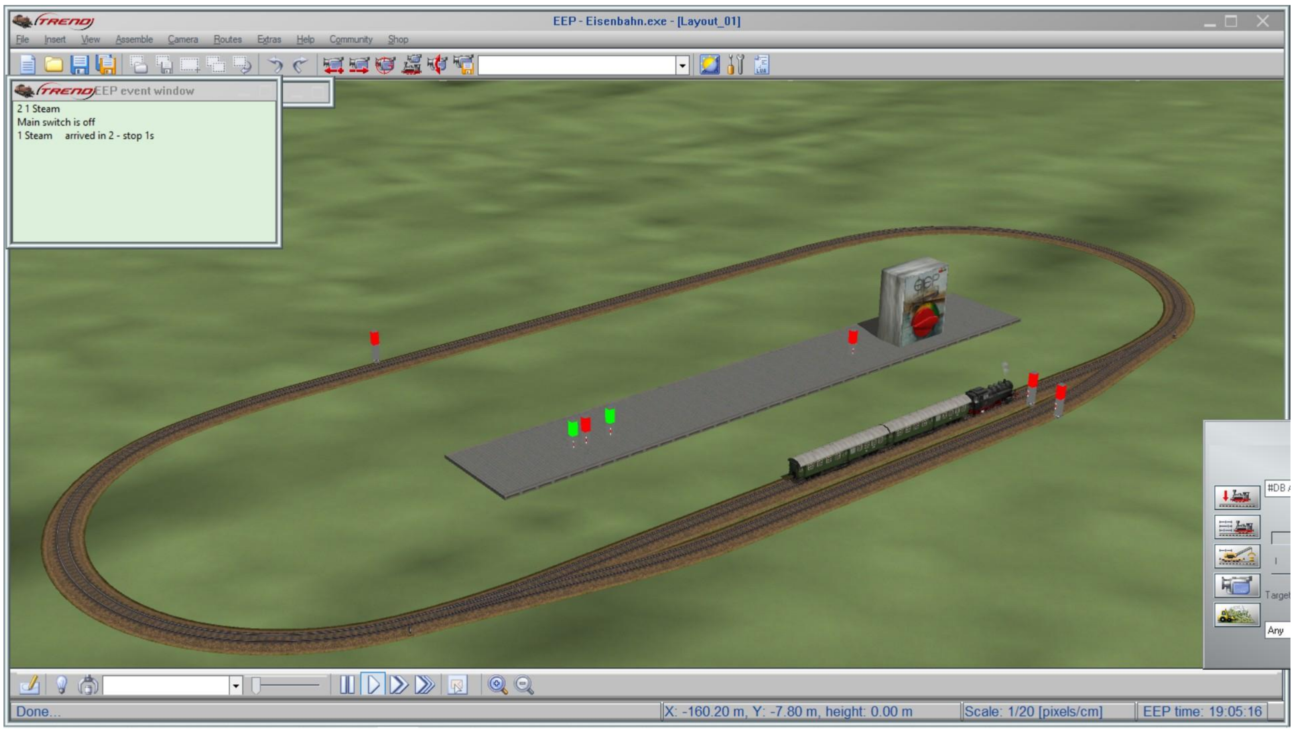
Rudy Boer, February 2022

Purpose

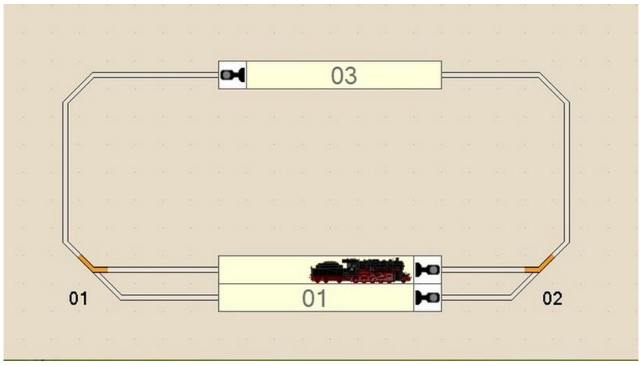
This Lua code generates automatic train traffic on any EEP layout, without the need to write or to change any of the code. All it takes to control a layout is to enter data that defines the layout into a set of Lua tables and variables … data on trains, on signals, and on which turnouts to switch per route.

Example of a Layout Definition

The picture below shows an example of a simple model train layout in EEP.

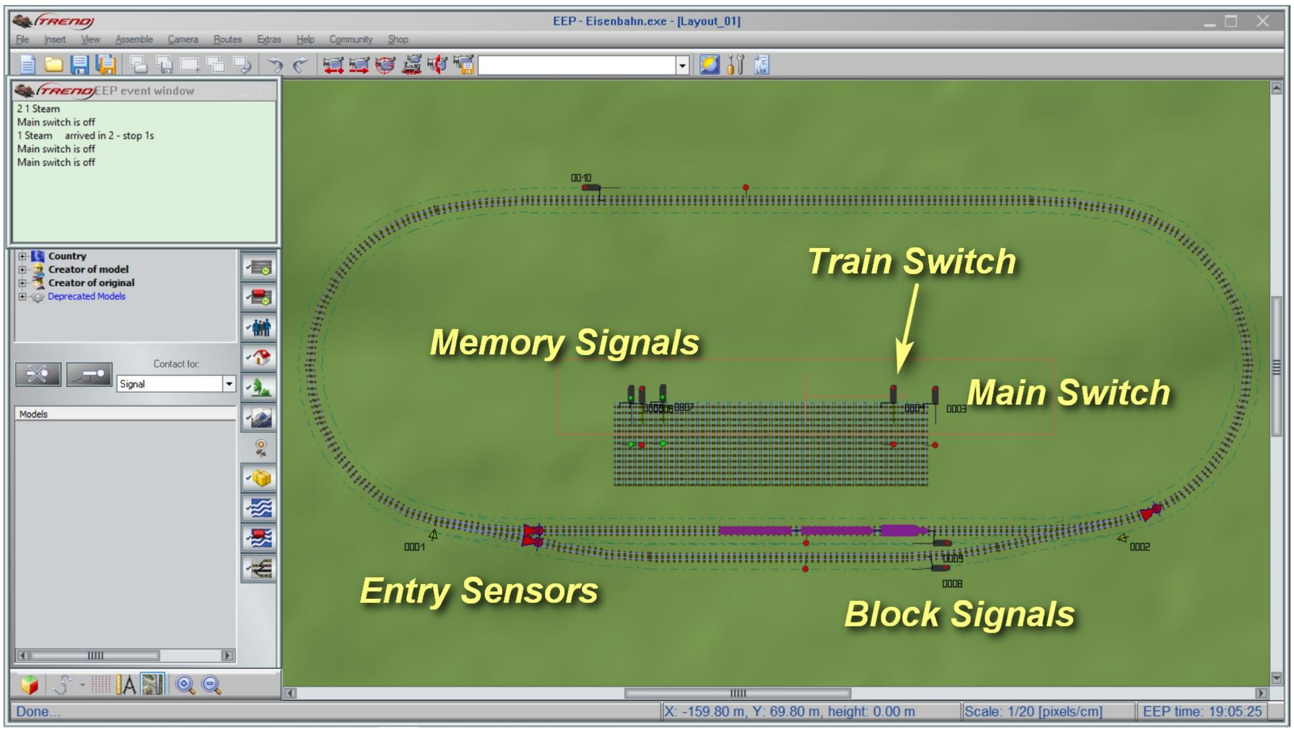

EEP - Eisenbahn.exe - [Layout_01]
File Insert View Assemble Camera Routes Extras Help Community Shop
EEP event window
Done...
X: -160.20 m, Y: -7.80 m, height: 0.00 m
Scale: 1/20 [pixels/cm]
EEP time: 19:05:16

2 1 Steam
Main switch is off
1 Steam arrived in 2 - stop 1s

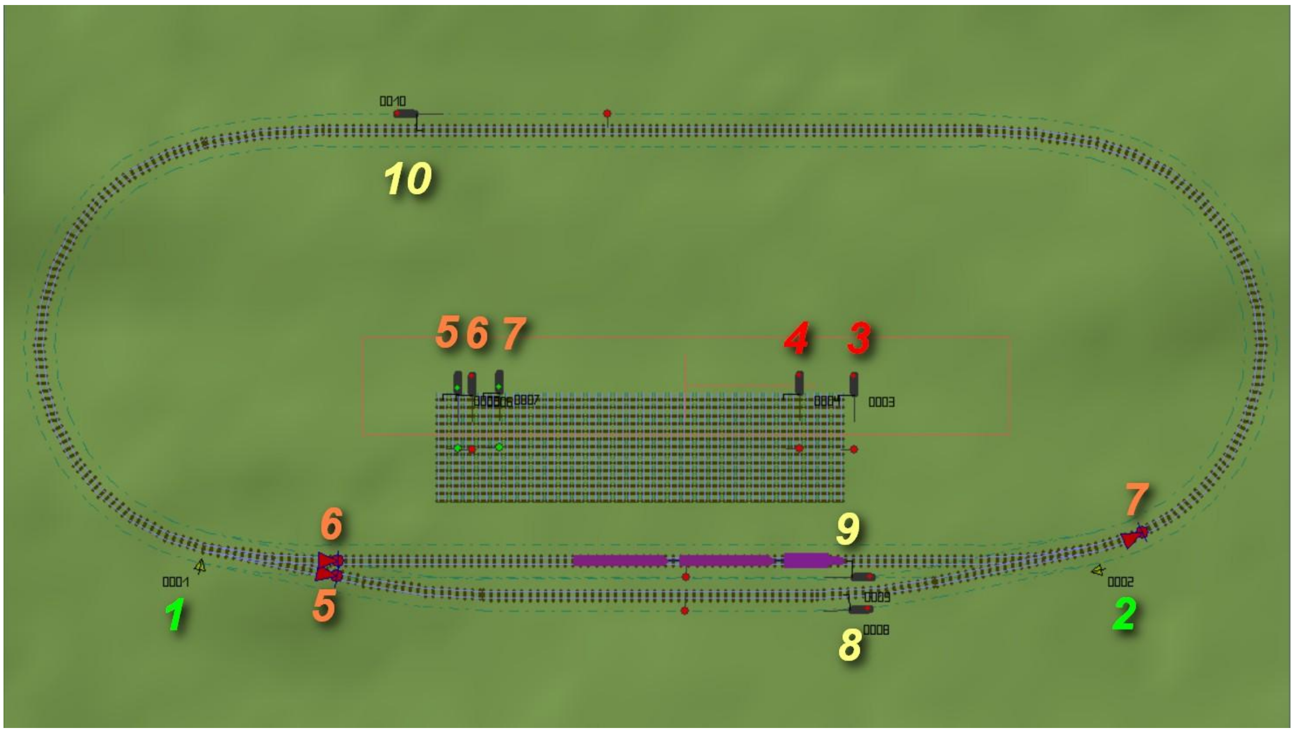


With automatic traffic trains drive from block to block. The first step is to decide which blocks we want to define. There's only one rule here: turnouts can not be part of a block, they are part of the route between two blocks. For this layout it seems logical to have three blocks, as shown here.

Every block gets a block-signal that can stop the train. Every block also gets a memory-signal. These are placed on a virtual control panel. They show if a block is free or occupied, while they also serve as a memory because signal states are saved when EEP is exited. Every block gets a track sensor at its entry that switches its memory signal to red. A main switch and an on/off switch per train (in this example there's just one train) are also added on the control panel.



```
2 1 Steam
Main switch is off
1 Steam    arrived in 2 - stop 1s
Main switch is off
Main switch is off
```



The above picture shows the turnout- and signal numbers, as they were generated by EEP when they were placed. These numbers are used in the following tables and variables to define the layout:

```
train[1] = {name="Steam", onoff=4, route=0, block=2}

--     block  1, 2, 3
allowed[1]= {15, 1, 1} -- train 1 has a 15s stop time in block 1
twowayblk = { 0, 0, 0} -- all blocks are one way traffic
blocksig  = { 8, 9,10} -- numbers of the block signals
memsig    = { 5, 6, 7} -- numbers of the memory signals

route[ 1] = { 1, 3,turn={2,1}} -- from block, to block, turnouts to switch
route[ 2] = { 2, 3,turn={2,2}}
route[ 3] = { 3, 1,turn={1,1}}
route[ 4] = { 3, 2,turn={1,2}}

MAINSW    = 3 -- number of the main switch
MAINON    = 1 -- 'on' state of the main switch
MAINOFF   = 2 -- 'off' state of the main switch
BLKSIGRED = 1 -- 'red' state of the block signals
BLKSIGGRN = 2 -- 'green' state of the block signals
MEMSIGRED = 1 -- 'red' state of the memory signals
MEMSIGGRN = 2 -- 'green' state of the memory signals
```

These tables, residing at the top of the Lua code, are all that needs to be edited. The remainder of the Lua code stays the same, no matter what layout we want to control.

How Lua Generates Automatic Train Traffic

When a train arrives in a block it runs over the entry sensor, which sets the block's memory signal to red. Lua code cycles 5 times per second. Every cycle Lua checks for green-to-red transitions of the memory signals. If such a transition is detected it triggers the following sequence of events.

First Lua reads the train number from the *blockreserved[b]* table, which holds the number of the train that reserved the block, which was written there when the train started its route.

Then Lua puts the train number in the *request[b]* table to remember that this block now holds a newly arrived train which requests a new route. Lua also fills the *stoptimer[b]* table with the stop time for this train in this block. This stop time is read from the *allowed[t][b]* table, which doubles as the stop time definition table. Stop timers are counted down every cycle until they reach zero.

Lua now reads the route number of the arriving train from the train table *train[t].route*, which was written there when it started its route.

Knowing the train's route, Lua reads its departure block from the *route[r]* table, which holds the departure block, the destination block and the 'main' or 'branch' state of the turnouts in between, if any. Lua sets the memory signal of the departure block to green.

Every cycle Lua checks for red-to-green transitions of the memory signals. Thus, the next cycle this red-to-green transition will be detected, based on which the departure block will be set to ‘free’ in the *blockreserved[b]* table and also the route’s turnouts are set to ‘free’ in the *turnreserved[to]* table.

The 'check for arrivals, flag new route requests and release old blocks and turnouts' phase of the cycle is now finished. What follows is the determination of available new routes, randomly pick a route, and start it.

For all the above generated route requests, Lua will determine which new routes are available, by checking:

- if the main switch and the individual train on/off switch are on
- if the stop timer has run out
- which routes have the train's current block as their departure block
- which destination blocks are allowed for this train and if these blocks are free
- which turnouts are needed for new routes and if they are free

Available routes (the ones where all the above conditions are met) are placed in the *available[r]* table, which is cleared and rebuilt every cycle. From this table, one route is randomly selected. Now Lua:

- places this route in the train table *train[t].route*
- reserves the destination block by writing the train number in *blockreserved[b]*
- reserves the turnouts by writing the train number in *toreserved[b]* for all turnouts on this route and switches the turnouts in the state needed for this route
- finally sets the block signal to green, which allows the train to move on.

The whole process starts from the top again when the train is detected in its destination block.

Tables that define the layout, configured by the user

train = {}

```
train[1] = {name="Passenger", route=0, onoff=14, block=2}
train[2] = {name="Cargo", route=0, onoff=15, block=5}
```

- **name=”Passenger”**. Train names are shown on the Lua monitor screen upon arrival in a block and at the start of a new route. They have no other purpose than that.
- **route = 0**. The route the train is on. Initially these need to be set to 0.
- **onoff=14**. The signal number that is used as this train’s on/off switch.
- **block=2**. The block number where this train currently is located on the layout.

When the Lua code is run for the very first time it does not know where the trains are. See the chapter "How to place trains on the track and initialize Lua" below on how to place trains.

allowed = {}

```
/*       block 1, 2, 3, 4, 5, 6, 7 */
allowed[1] = { 1, 1, 1, 1, 1, 0, 0}
allowed[2] = {23, 1,35, 0, 0, 1, 1}
```

This table specifies if a train is allowed in a block and if it has a stop time:

- 0: this train is not allowed in this block
- 1: this train is allowed in this block, no stop time
- >1: this train is allowed in this block and it has a stop time. The time the train requires to drive from the block entry sensor to the block signal has to be included. Example: if the measured drive time from sensor to signal of this train in this block is 15 seconds and a stop time of 8 seconds is desired, the number to fill in is 15+8=23.

In the above example:

- train 1 is not allowed in blocks 6 and 7. No stop times are specified.
- train 2 is not allowed in blocks 4 and 5. It has a stop time of approximately 8s in block 1 and 20s in block 3 (assuming the drive time from sensor to signal in both blocks is 15s).

blocksig = {14,15,16,…}

Every block has a signal where trains will stop if it’s red. The signal is controlled by Lua. It is switched to green when the train is allowed to start a new route to an adjacent block. It is switched to red again upon arrival of the train in the next block.

If a block is used in two directions, this is treated as being two blocks, in opposite directions, each with their own block signal and memory signal. This is specified in the *twowayblk[b]* table.

IMPORTANT: In EEP not all signals have similar states. With some signals 1 means ‘closed’, 2 means ‘open’, while with others it’s vice versa. To deal with this, only use block signals that have the same red/green states for the entire layout. The correct states of the block- and memory signals in use are defined with variables, see the ‘Variables …’ chapter below.

memsig = {39,20,21,…}

Besides a track signal, every block also has a memory signal. This is not placed on the train rails but on a separate (hidden) track. They are used for visual feedback of occupied blocks, as part of a control panel. If preferred they can also be completely hidden.

They are switched to red by a sensor that is triggered when a train enters the block. Lua detects this green-to-red transition and now knows the train has arrived in this block. Lua switches the *memsig[b]* of the departure block to green (free).

EEP saves the state of signals when exiting the program. When starting up later, the *memsig[b]* table is filled by reading out these memory signals.

IMPORTANT: only use memory signal types that have similar states for red/green.

twowayblk = { 0, 0, 4, 3, 0, 0}
In this example blocks 3 and 4 are two way block 'twins'. If a train reserves block 3 for its new route, Lua reads in *twowayblk[3]* that it also has to reserve block 4. Vice versa if a train reserves block 4, Lua reads in *twowayblk[4]* that it also has to reserve block 3.

route = {}
```
route[1]={2,3}
route[2]={3,8,7,1,12,2} -- 1:main, 2:branch
```

Routes are specified from one block to the next. The table holds the departure block, the destination block and possibly one or more turnouts that need to be switched. In the above example:
- for route 1, 2 is the departure block, 3 is the destination block, and there are no turnouts to be switched.
- for route 2, 3 is the departure block, 8 is the destination block, turnout 7 needs to be switched to ’main’ and turnout 12 to ‘branch’.

Variables configured by the user

PLACE_TRAINS = 0 or PLACE_TRAINS = 1
See the chapter "How to place trains on the track and initialize Lua" below on how to place trains.

MAINSW = 27
The address of the signal that is used as the main switch. If the main switch is turned on this enables trains to drive. In addition each train has an individual on/off switch. If a train switch or the main switch is turned off, trains will first continue driving on their current route, until they reach their destination block where they are halted by a red signal. From tere they will not start a new route until switched on again.

MAINON = 1, MAINOFF = 2, or vice versa
Values can be either 1 or 2, depending on the type of signal used as the main switch. The user has to check this inside EEP.

BLKSIGRED = 1, BLKSIGGRN = 2, or vice versa
The value used in EEP for a red and green block signal, which can be 1 or 2 depending on the type of block signals used. The user has to check this in EEP. Beware to only use block signals that have the same states on the entire layout.

MEMSIGRED = 1, MEMSIGGRN = 2, or vice versa
The value used in EEP for a red and green memory signal, which can be 1 or 2 depending on the type of memory signals used. The user has to check this inside EEP. Beware to only use memory signals that have the same states on the entire layout.

Tables internally used by Lua

memsigold = {}

This table stores the previous value of *memsig[b],* such that a 0>1 transition can be detected, which happens when a train arrives, or a 1>0 transition for the departed block.

blockreserved = {}

blockreserved[b] holds 0 if a block is free. If a block is reserved, it holds the number of the train that reserves it. These blocks are not available for new routes. When a train arrives at the destination block of its route, the departure block will be set free.

turnreserved = {}

If a train reserves a route, it not only reserves the destination block, it also reserves the turnouts between the blocks to prevent other trains from using them. When a train arrives at the destination block of its route, the turnouts that were reserved for this route are now set free.

request = {}

When a train arrives at the destination block of its route Lua fills *request[b]* with the train number, to remember that this train now requests a new route.

available = {}
This table is cleared every Lua cycle and then built up again. It holds the 'available route' numbers for all the trains that requested new routes. Lua randomly selects one route from the *available[r]* list ... that train will be the only one allowed to start a new route during this Lua cycle. Lua will now reserve the destination block and the turnouts of this route. This could mean that other routes that were available this same cycle may now not be available anymore, which is why the table is cleared and recalculated every Lua cycle.

stoptimer = {}

Stop times, per train per block, are specified in the *allowed[t][b]* table. Upon arrival *stoptimer[b]* is loaded with 5x the defined stop time. 5x, because Lua cycles 5x per second and the stop timers are decremented each cycle. When *stoptimer[b]* reaches 0, the stop time for this train in this block has run out and the train becomes available for a new route.

How to place trains on the track and initialize Lua

When the Lua code is run for the first time, it does not yet know where the trains are … we’ll have to tell it that.

There's a line of code that reads PLACE_TRAINS = 0 or PLACE_TRAINS = 1. Lua checks this variable and if it's 1 Lua will turn the main switch and the individual train-switches off and it will halt selection of new routes.

Trains will not drive automatically in this mode. This gives the opportunity to place one or more new trains or to manually move existing ones. A newly placed train is manually given a speed, which sets it in motion until it reaches a red signal, where it will stop.

When all trains are manually driven to their desired positions, we have to edit the *trains[t]* table to reflect the current block numbers of the trains and to define the on/off switch signal numbers and train names.

After editing the train table, reload the code, still with PLACE_TRAINS = 1. Lua reads the *train[t]* table, reserves the blocks that have a train on them, sets the corresponding memory signals to red and writes the block status (free: 0, reserved: train nr) to disk.

Now change the code to PLACE_TRAINS = 0 and reload the code. This brings Lua into driving mode. What happens now is explained in the 'How it Works' chapter above.

How to stop driving and save the current state

Before exiting EEP it is good practice to switch off the main switch and wait until all trains have come to a stop at a signal, otherwise the status of the layout may not be unambiguous and may not be saved correctly.

There’s no need to save the layout via the menu, the current state is saved automatically when EEP is exited.