

# EEP-Tutorial-Anlagen



(Ausgabe: 02.09.2025)

## Vorwort

|                                 |                                                      |
|---------------------------------|------------------------------------------------------|
| <b>Tutorial_33_LUA_1:</b>       | <b>Signale und Weichen, Funktionsaufruf per KP</b>   |
| <b>Tutorial_34_LUA_2</b>        | <b>Callback(Rückruf)-Funktionen 1</b>                |
| <b>Tutorial_35_LUA_3</b>        | <b>Callback(Rückruf)-Funktionen 2</b>                |
| <b>Tutorial_36_LUA_4</b>        | <b>Tabellen und Zufallsgenerator</b>                 |
| <b>Tutorial_37_LUA_5</b>        | <b>Zeittakt, Geschwindigkeit, Kuppeln</b>            |
| <b>Tutorial_38_LUA_6</b>        | <b>Achsenslots</b>                                   |
| <b>Tutorial_39_LUA_7</b>        | <b>Rollmaterialachsen</b>                            |
| <b>Tutorial_40_LUA_8</b>        | <b>Data-Slots</b>                                    |
| <b>Tutorial_44_LUA_1</b>        | <b>Rauch, Licht und Feuer in Immobilien</b>          |
| <b>Tutorial_45_LUA_2</b>        | <b>Achsen-Animation in Immobilien und GOs</b>        |
| <b>Tutorial_46_LUA_3</b>        | <b>Immobilien positionieren und Achsen setzen</b>    |
| <b>Tutorial_48_LUA</b>          | <b>Rauch, Licht und Warnton eines Zuges</b>          |
| <b>Tutorial_49_LUA</b>          | <b>Achsen und Haken eines Kranzuges</b>              |
| <b>Tutorial_50_LUA</b>          | <b>Zug-Routen ermitteln und zuweisen</b>             |
| <b>Tutorial_51_LUA</b>          | <b>Zugteil abkuppeln, Zugkupplungen schalten</b>     |
| <b>Tutorial_53_LUA</b>          | <b>Gleis besetzt, Kameras aufrufen, Anlage laden</b> |
| <b>Tutorial_54_LUA</b>          | <b>Anlage laden 2</b>                                |
| <b>Tutorial_55_Gleisbesetzt</b> |                                                      |

Weitere Informationen zu Lua in EEP:

- [Grundlagen zu Lua in EEP](#)
- [EEP-spezifische Lua-Variablen und -Funktionen](#)

Achtung: Diese beiden Links funktionieren nur wenn alle 3 Dateien in einem Ordner liegen.

Die Links innerhalb der Dokumentation führen ins Internet. Bei Nutzung können also Kosten durch den Provider entstehen.

## Vorwort

Die in EEP mitgelieferten Tutorial-Anlagen sind unspektakulär und nur mit wenigen Lua-Funktionen ausgestattet. Gerade deshalb eignen sie sich besonders gut zum Studium der Skriptsprache Lua sowie für eigene Experimente. Es ist daher empfehlenswert jede einzelne der Anlagen genau zu studieren.

Meist fährt nur eine einzelne Lok im Kreis und macht dabei kaum mehr als eine Weiche oder ein Signal zu schalten. Das Spannende an den Tutorial Anlagen sind nicht die Dinge, welche auf den Anlagen passieren, sondern die Zeilen im Lua-Skript, welche das bewirken.

In vielen Fällen werden Sie denken, dass das doch *alles auch ganz prima ohne Lua ginge*. Aber verkennen Sie bitte nicht den Zweck eines Tutorials. Es soll nicht demonstrieren, dass mit Lua alles bequemer geht, sondern Ihnen verstehen helfen, wie Lua funktioniert. Und wenn Sie erst ein Gespür für diese Skriptsprache entwickelt haben, dann werden Sie staunen, was mit damit alles möglich ist. Mit Lua können Sie Ihren Anlagen *Intelligenz* einhauchen und in wenigen Zeilen Steuerungen entwerfen, die früher Schaltwege gigantischen Ausmaßes erforderten. *Dazu* haben Sie mit einem Skript alles, was die Anlage steuert, an zentraler Stelle zusammengefasst. Außerdem können sie das Skript auch nach vielen Jahren noch *lesen* und das nicht nur Sie sondern gegebenenfalls auch andere EEP-Nutzer.

Scheuen Sie sich nicht Veränderungen vorzunehmen und die Konsequenzen zu beobachten, welche diese Veränderungen haben. Eigene Experimente sind der spannendste und effektivste Weg um etwas zu lernen. Und solange sie die Tutorial-Anlagen nicht überschreiben, können Sie keinen Schaden anrichten.

Die Tutorial-Anlagen sind deswegen einfach gehalten, damit die zugehörigen Skripte klein und nachvollziehbar sind. Veränderungen am Skript haben überschaubare Folgen. Haben Sie keine Angst Fehler zu machen, wenn Sie die Skripte verändern. Im Gegenteil - provozieren Sie Fehler. Aus Fehlern lernt man am meisten.

Die Erläuterungen zu den Tutorial-Anlagen bauen aufeinander auf. Was beispielsweise im Kapitel zur Anlage "Tutorial\_33\_Lua\_1" erläutert wurde, wird im Kapitel zur Anlage "Tutorial\_34\_Lua\_2" als bekannt vorausgesetzt.

### Tutorial\_33\_LUA\_1: Signale und Weichen, Funktionsaufruf per KP

Dieses Tutorial führt Sie in die Grundlagen der Anlagensteuerung per Lua ein. Ein Zug fährt abwechselnd auf dem inneren und dem äußeren Oval und schaltet dabei Weichen und Schranken mit Hilfe der Lua Funktionen:

- [EEPVer](#)
- [EEPMain\(\)](#)
- [EEPSetSwitch\(\)](#)
- [EEPSetSignal\(\)](#)

Die Erläuterungen zu dieser Tutorial Anlage werden wesentlich ausführlicher sein als die zu allen folgenden Tutorials, weil sie viele grundlegende Erklärungen enthalten. Daher sollten Sie dieses erste Tutorial sehr gründlich zu studieren. Es vermittelt Ihnen Grundlagenkenntnisse für die Steuerung einer Anlage mittels Lua. Außerdem enthält es besonders viele Anregungen für eigene Versuche, die Ihnen ein Gespür für den Umgang mit Lua geben sollen.

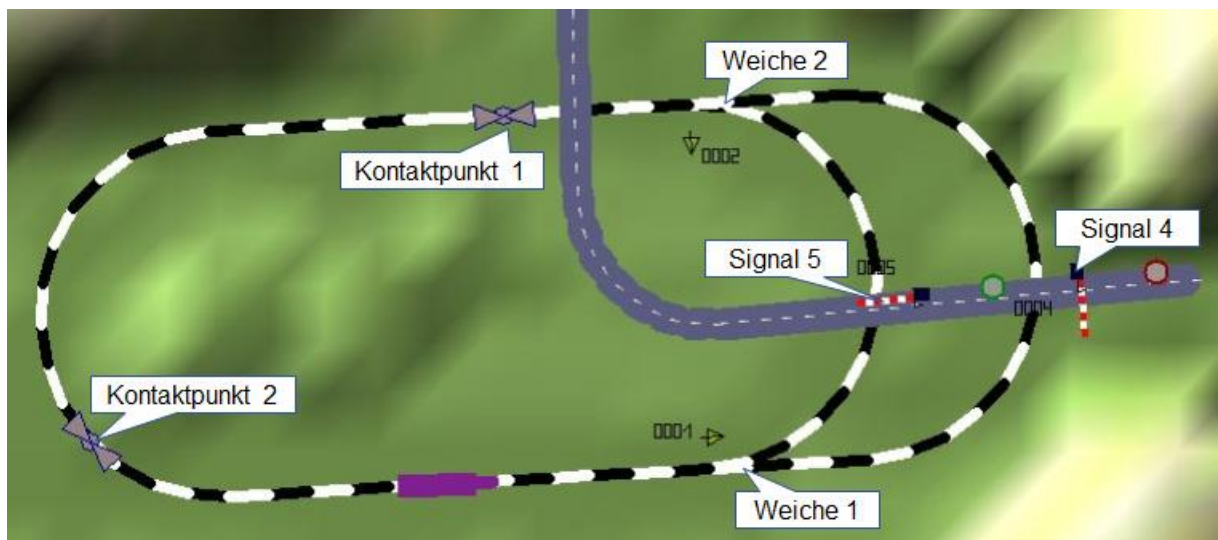


Bild 33\_1

Auf der 2D Ansicht zu dieser Anlage (Bild 33\_1) sind zwei Schranken, zwei Weichen und zwei Kontaktpunkte für Fahrzeuge zu erkennen. Nimmt man die Anlage in Betrieb, dann fährt eine einzelne Lok ständig im Kreis. Dabei wechselt sie in jeder Runde den Weg und schließt die zugehörige Schranke. Die Schranken und Weichen werden dabei durch die beiden Kontaktpunkte gesteuert, jedoch nicht in der gewohnten Weise, sondern dadurch, dass diese Kontaktpunkte Funktionen aufrufen, welche im Lua Skript definiert sind.

Öffnen Sie bitte den Lua-Skript-Editor (Bild 33\_2) und betrachten Sie die ersten vier Zeilen des Skripts. Sie enthalten Kommentare, die vom in EEP integrierten Lua-Interpreter ignoriert werden und nur dem Benutzer als Merkhilfen dienen. Kommentarzeilen beginnen immer mit 2 Minus-Zeichen.



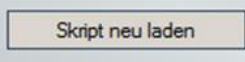
Bild 33\_2

```
-- Lua tutorial
-- This tutorial demonstrates how to change switches and signals
-- depending on value of variable
-- Functions SETROUTE1() and OpenAllSignals() calls contact points.
```

**Auf Deutsch:**

-- Dieses Tutorial zeigt, wie man Weichen und Signale ändern kann  
-- abhängig vom Wert einer Variablen  
-- Die Funktionen SETROUTE1() und OpenAllSignals() rufen Kontaktpunkte auf.

Sie können die Kommentarzeilen für einen ersten, eigenen Versuch nutzen. Löschen Sie in einer Zeile eins der zwei Minuszeichen am Zeilenanfang. Laden Sie anschließend das geänderte Skript durch einen Mausklick auf die in Bild 33\_3

**Bild 33\_3**

gezeigte Schaltfläche neu. Es öffnet sich ein Fenster und meldet Ihnen, dass Ihr Skript einen Fehler enthält. Der Skripteditor bleibt geöffnet, weil der Interpreter das fehlerhafte Skript abgelehnt hat. Korrigieren Sie bitte den Fehler, laden Sie das korrigierte Skript erneut und wechseln Sie in den 3D Modus, um die Vorgänge auf der Anlage zu beobachten.

Sie können den Skript Editor auch öffnen, während Sie im 3D Modus sind. Ratsam ist das nicht, weil die Anlage dann weiterläuft, aber Lua pausiert. Probieren Sie es trotzdem aus. Öffnen Sie den Skript Editor und beobachten dann weiterhin den Zug. Der dreht wie zuvor seine Kreise, aber er wechselt die Route nicht mehr und bedient die Schranken nicht.

Es ist daher mehr als empfehlenswert **immer in den 2D Editor zu wechseln**, wenn Sie am Skript Veränderungen vornehmen möchten, damit die Anlage für die Dauer Ihrer Arbeiten am Skript angehalten wird.

In der nächsten Zeile im Skript steht

```
clearlog()
```

Die runden Klammern hinter dem Wort kennzeichnen `clearlog()` als eine **Funktion**. Sie ist in EEP definiert und kann deshalb sofort ausgeführt werden. Die Funktion `clearlog()` löscht den Inhalt des Ereignisfensters.

Wenn Sie mögen, dann können Sie einen Kommentar an die Zeile anfügen, der Sie an den Zweck dieser Funktion erinnert:

```
clearlog() -- löscht den Inhalt des Ereignisfensters
```

Wie Sie am Beispiel sehen können, sind in Kommentaren auch Umlaute (sowie Sonderzeichen) erlaubt. Denn alles, was hinter zwei Minuszeichen steht, wird vom Interpreter nicht weiter beachtet.

In der folgenden Zeile steht:

```
route = 0
```

Diese Zeile erstellt eine **Variable** namens `route` und weist ihr den Wert 0 zu.

Die Variable wird etwas später im Skript benötigt. Wenn Sie den Namen hier vorne ändern, aber an den übrigen Stellen im Skript nicht, dann wird das Skript nicht mehr ordnungsgemäß arbeiten. Aber der Interpreter kann diesen Fehler nicht sofort bei der Übertragung des Skripts erkennen. Der wird erst dann auffallen, wenn Sie die Anlage in Betrieb nehmen und Lua an den Punkt im Skript kommt, wo die Variable benötigt wird.

Die nächste Zeile im Skript lautet:

```
print("Hey let's start, EEP Version is: ", EEPVer)
```

Die runden Klammern hinter dem Wort `print` zeigen, dass es sich wieder um eine Funktion handelt. Und am Beispiel `print()` kann man erkennen, welchen Zweck die Klammern am Ende einer Funktion haben. Sie dienen dazu, einer Funktion beim Aufruf etwas mitzugeben. In diesem Fall den Text, den `print()` ausdrucken soll.

Um die Ausgaben von `print()` sehen zu können, muss das Ereignisfenster (Bild 33.5) in den Programmeinstellungen geöffnet werden (Bild 33\_6). Da ein Skript erst dann ausgeführt wird, wenn die Anlage läuft, erscheint der Text, den `print()` ausgibt, ebenfalls erst dann im Ausgabefenster, wenn man das Skript geladen hat und in den 3D Modus wechselt.

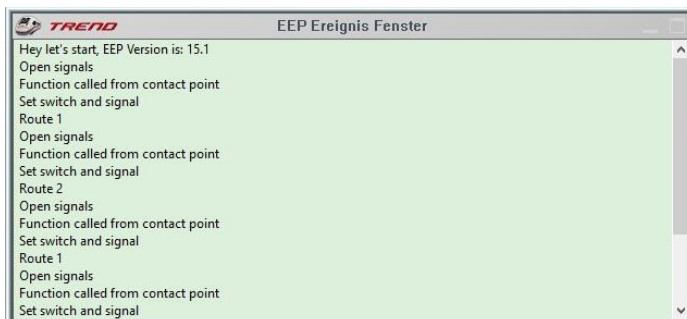


Bild 33\_4

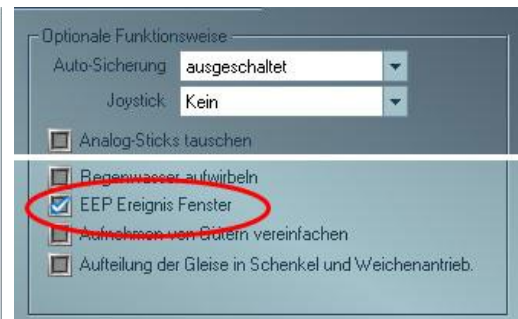


Bild 33\_5

Wenn Sie den Ausgabetext im Ereignisfenster genau betrachten, dann werden Sie bemerken, dass die Anführungszeichen nicht mit ausgegeben wurden. Zur Syntax von Lua gehört nämlich, dass Anführungszeichen als Steuerzeichen benutzt werden. Sie kennzeichnen das, was zwischen Ihnen steht, als Text damit der Interpreter es von Schlüsselwörtern sowie Variablen- und Funktionsnamen unterscheiden kann.

Sie können den Text in den Klammern nach Belieben ändern und zum Beispiel durch einen deutschen Text ersetzen:

```
print("Los geht's - die EEP Version ist: ", EEPVer)
```

Beachten Sie bitte, dass nur der erste Teil in Anführungszeichen steht. Dann folgt ein Komma und danach ein Wort, welches nicht in Anführungszeichen gesetzt ist. Das Komma ist ebenfalls ein Steuerzeichen. Es teilt dem Interpreter mit, dass der Funktion `print()` nach dem ersten Element noch ein zweites mitgegeben wird. Das zweite Element ist eine Systemvariable aus dem Satz an Variablen und Funktionen, welche

EEP mitgegeben wurden, damit es mit Lua Daten austauschen kann. `EEPVer` enthält die Versionsnummer der laufenden EEP-Version.

Die nächsten drei Zeilen enthalten die Definition einer Funktion

```
function EEPMain()  
    return 1  
end
```

Bei `EEPMain()` handelt es sich um eine spezielle Funktion, welche EEP immer wieder aufruft während eine Anlage läuft. Sie entspricht in etwa einem ständig kreisenden Schaltauto. Alles, was innerhalb der Funktion steht, entspricht vereinfacht ausgedrückt den Kontaktpunkten in einem Schaltkreis.

In der Anlage "Tutorial\_33\_LUA\_1" hat die Funktion `EEPMain()` keine Aufgaben. Dennoch muss sie definiert sein, denn EEP ruft diese Funktion unaufhörlich pro Sekunde fünfmal auf. Und die Funktion muss dabei jedes Mal eine Zahl zurückgeben. Tut sie das nicht, dann nimmt EEP an, dass es sich um eine Anlage ohne Lua Skript handelt und macht anschließend ohne Lua weiter.

Die Funktion `EEPMain()` bietet sich für eine ganze Reihe von Experimenten an.

Probieren Sie doch mal was passiert, wenn sie vor `return 1` eine Zeile einfügen und dort die `print()` Funktion nutzen:

```
function EEPMain()  
    print("Hallo!")  
    return 1  
end
```

Wenn Sie die Anlage starten und das Ereignisfenster geöffnet haben, dann werden Sie sehen, dass Lua den Text `Hallo!` dort hineinschreibt. Und zwar immer wieder. Wie Bart Simpson in der Titelsequenz von "The Simpsons". Denn die Funktion `EEPMain()` wird von EEP selbständig immer wieder aufgerufen. Fünfmal je Sekunde. Also alle 200 Millisekunden. Sie ist ein verdammt schnelles Schaltauto.

Zeit für ein zweites Experiment!

Schreiben Sie vor die `print()` Funktion zwei Minuszeichen:

```
function EEPMain()  
    -- print("Hallo!")  
    return 1  
end
```

Wenn sie die Anlage erneut starten, dann bleibt der Text `Hallo!` diesmal aus. Denn mit den beiden Minuszeichen haben sie alles, was dahinter steht, zum Kommentar

erklärt und der Interpreter ignoriert es. Sie haben soeben einen unter Programmierern sehr beliebten Trick gelernt, um Zeilen in einem Skript vorübergehend zu deaktivieren.

Die folgenden Zeilen im Skript definieren eine weitere Funktion. Ihr Name ist SETROUTE1

```
function SETROUTE1()
```

Dass es sich hier um die Definition einer Funktion handelt erkennt man am vorangestellten Schlüsselwort `function`. Es zeigt dem Interpreter, dass er die Funktion zu diesem Zeitpunkt nicht ausführen, sondern für spätere Zwecke speichern soll.

Diese Funktion wird durch den Kontaktpunkt 2 (Bild 33\_6) aufgerufen, welcher in der linken Kurve sitzt. Öffnen Sie bitte das Eigenschaftsfenster des Kontaktpunktes. Sie sehen in der oberen Hälfte eine neue Zeile *Lua Funktion* und im Feld dahinter den Namen der Funktion, die bei Auslösen des Kontaktpunktes aufgerufen werden soll. Beachten Sie bitte, dass im Feld nur der Funktionsname steht, aber nicht die runden Klammern! Würden Sie an dieser Stelle die Klammern mit eingeben, dann käme es beim Aufruf der Funktion zu Fehlverhalten. Denn die runden Klammern gehören zwar zu jeder Funktion, sind aber nicht Teil des Namens. Im Feld hinter Lua Funktion darf nur der Name der Funktion stehen.

Wenn der Name im Kontaktpunkt nicht exakt mit dem Namen übereinstimmt, den die Funktion bei ihrer Definition im Skript erhalten hat, dann kann EEP die Funktion nicht finden und gibt eine entsprechende Fehlermeldung aus. Groß- und Kleinschrift werden unterschieden und müssen deshalb im Kontaktpunkt und in der Funktionsdefinition identisch sein!

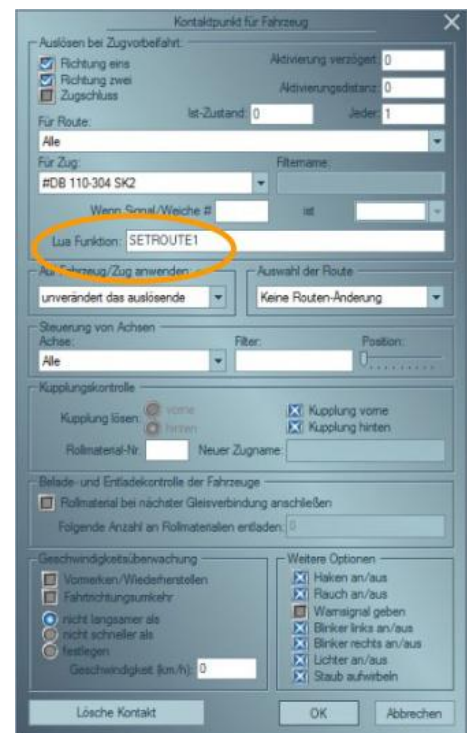


Bild 33\_6

Probieren sie es aus, wenn Sie mögen. Ersetzen Sie im Kontaktpunkt SETROUTE1 durch SetRoute1. Sie werden beim Schließen des Eigenschaftsmenüs folgende Fehlermeldung erhalten:

Der Grund für den Fehler ist in solch einem Fall leicht zu übersehen. Denn beim Lesen unterscheiden wir Groß- und Kleinschrift nicht so genau, wie es der Lua-Interpreter tut. Noch gemeiner ist ein Leerzeichen vor dem Funktionsnamen im Kontaktpunkt. Es ist nur sehr schwer zu erkennen und wird auch bei Schließen des Eigenschaftsmenüs nicht bemerkt. Der

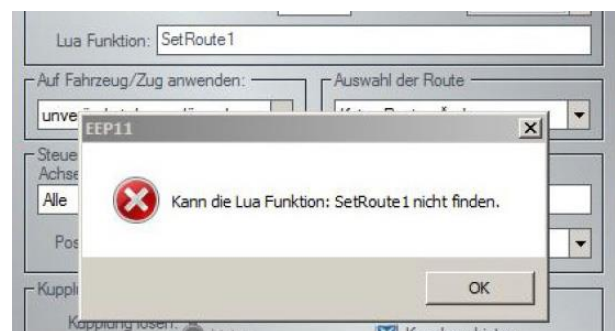


Bild 33\_7

Fehler tritt erst in dem Moment auf, in dem der Zug den Kontaktpunkt überfährt. Daher erscheint die Fehlermeldung diesmal im Ereignisfenster (Bild 33\_8).

Man muss schon sehr genau hinschauen um zu erkennen, dass in der Fehlermeldung vor SETROUTE1 ein Leerzeichen steht. Deshalb auch hier die Empfehlung: Probieren Sie es selbst aus. Provozieren Sie in der Tutorial Anlage absichtlich diesen Fehler. So werden Sie damit vertraut und wenn er Ihnen später einmal aus Versehen widerfährt, dann werden Sie sofort wissen, was da schief gelaufen ist.

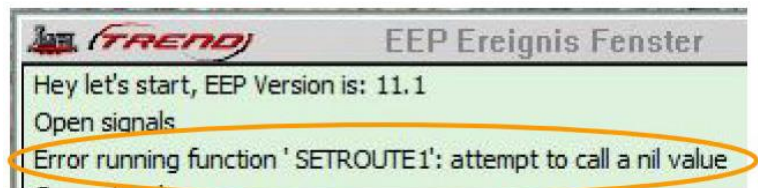


Bild 33\_8

Die Definition der Funktion SETROUTE1 besteht aus vielen unterschiedlichen Anweisungen:

Die Definition der Funktion SETROUTE1 besteht aus vielen unterschiedlichen Anweisungen:

```
function SETROUTE1()  
    print("Function called from contact point")  
    print("Set switch and signal")  
    -- variable changes switch --  
    route = route + 1  
    if ((route % 2) == 1) then  
        print("Route 1")  
        EEPSetSwitch(1, 1)  
        EEPSetSignal(5, 2)  
    else  
        print("Route 2")  
        EEPSetSwitch(1, 2)  
        EEPSetSignal(4, 2)  
    end  
end
```

Zunächst werden zwei Textzeilen ins Ereignisfenster geschrieben. (*"Funktion wird vom Kontaktpunkt aufgerufen"* und *"Schalte Weiche und Signal"*)

Dann folgt ein Kommentar (*"Variable ändert Weichenstellung"*). Die beiden Minuszeichen am Ende des Kommentars dienen nur der Optik und haben keine Funktion. Alles, was nach den ersten zwei Minuszeichen folgt, wird als Kommentar betrachtet. Also auch die beiden Minuszeichen zum Schluss.

In der folgenden Zeile steht dann

```
route = route + 1
```

Etwas Ähnliches hatten Sie schon weiter oben im Skript gesehen. Dort stand `route = 0`

Ein Wort gefolgt von einem `=` Zeichen bedeutet, dass einer Variablen ein Wert zugewiesen wird.

Der Unterschied hier ist, dass dieser Wert zuerst mit `route + 1` ausgerechnet wird. Das ist gewöhnungsbedürftig, weil es unserem alltäglichen Sprachgebrauch widerspricht. Aber in Programmiersprachen hat sich diese Schreibweise bewährt und deshalb gehört sie auch zur Syntax von Lua. Der Ausdruck rechts vom `=` Zeichen wird zuerst berechnet und dann der Variablen links vom `=` Zeichen zugewiesen. Diese Reihenfolge ist äußerst praktisch. Denn sie bedeutet, dass man wie im vorliegenden Fall den alten Wert einer Variablen nehmen, damit rechnen und das Ergebnis wieder in derselben Variablen abspeichern kann.

Im konkreten Beispiel wird zur Variablen `route` jedes Mal 1 hinzugezählt, wenn der Kontaktpunkt 2 überfahren wird. Also ist `route` eine Art Strichliste für die Runden, welche die Lok dreht.

In der nächsten Zeile zeigt sich der Zweck dieser Strichliste:

```
if ((route % 2) == 1) then
```

Die Worte `if` und `then` sind Schlüsselworte. Das erste ist das englische Wort für "wenn" oder "falls" und das zweite ist das englische Wort für "dann".

Schlüsselworte sagen einem Interpreter, was mit dem nachfolgenden Teil zu tun ist. Bei `if` bedeutet das "Prüfe, ob das folgende stimmt". Das Schlüsselwort `then` zeigt dem Interpreter, dass die Bedingung, welche `if` prüfen soll, hier endet und nun das kommt, was der Interpreter tun soll, wenn das Ergebnis richtig also in Computersprache "wahr" bzw. `true` ist.

Übersetzt in Umgangssprache steht in der Zeile:

Falls `(route % 2) == 1`, dann ...

Lua benutzt also die Strichliste `route` und entscheidet danach, wie es weitergehen soll. Das doppelte `==` Zeichen ist kein Schreibfehler, sondern gehört ebenfalls zur Syntax von Lua. Es teilt dem Interpreter mit, dass zwei Werte miteinander verglichen werden sollen. Das Ergebnis ist entweder `true` (für "wahr", also richtig) oder `false` (für falsch).

Im vorliegenden Beispiel wird `route % 2` mit der Zahl 1 verglichen. Anders ausgedrückt wird geprüft ob die Berechnung `route % 2` den Wert 1 ergibt.

Das Prozentzeichen gehört ebenfalls zur Syntax von Lua. Es hat nichts mit normaler Prozentrechnung zu tun, sondern ist der sog. **Modulo Operator**.

Es gibt in der Programmierung häufig Situationen, in denen nur Zahlen aus einem kleinen Bereich benötigt werden. Im Zusammenhang mit EEP könnten das beispielsweise die Gleisnummern eines Bahnhofs sein. Der Modulo Operator sorgt dafür, dass ab einem vorgegebenen Grenzwert wieder von vorne begonnen wird. Dieser Grenzwert muss hinter dem `%` Zeichen stehen.

Grob kann man das mit einer analogen Uhr vergleichen. Von 1 Uhr bis 12 Uhr zeigt sie genau diese Zahlen an. Aber um 13 Uhr zeigt sie wieder 1 Uhr, um 14 Uhr zeigt sie 2 Uhr und so weiter.

Ein Unterschied zur Uhr ist, dass Modulo immer wieder mit 0 beginnt und nicht mit 1; denn mathematisch ist Modulo der Restwert einer Division. 8 dividiert durch 12 ist 0 mit einem Rest von 8 ( $8 \% 12 = 8$ ). 15 dividiert durch 12 ist 1 mit einem Rest von 3 ( $15 \% 12 = 3$ ). 12 dividiert durch 12 ist 1 mit einem Rest von 0 ( $12 \% 12 = 0$ ).

Der Ausdruck `route % 2` kann damit als Ergebnis nur 0 oder 1 haben. Damit ist das Ergebnis der Prüfung abwechselnd wahr oder falsch. Und das nutzt Lua in den folgenden Zeilen, um zu entscheiden, ob der Zug die äußere oder innere Runde fahren soll.

Wenn das Ergebnis `true` ist, dann führt Lua alles aus, was nach `then` folgt, bis es auf ein `elseif`, `else` oder `end` stößt. Ist das Ergebnis hingegen `false`, dann springt Lua zum nächsten `elseif`, `else` oder `end` und macht an dem Punkt weiter.

Vereinfacht gesagt ist das ganze `if`-Konstrukt nichts anderes als eine Weiche im Schaltkreis. Wenn `route` eine ungerade Zahl gespeichert hat, dann ist das Ergebnis von `route % 2` gleich 1. In diesem Fall führt Lua die Funktionen

```
print("Route 1")
EEPSetSwitch(1, 1)
EEPSetSignal(5, 2)
```

aus.

Die Funktion `EEPSetSwitch(1, 1)` stellt die Weiche 1 in Stellung 1, also Fahrt (Bild 33\_9). Der Zug wird also die innere Runde fahren.



Bild 33\_9

Die Funktion `EEPSetSignal(5, 2)` stellt das Signal 5 auf Stellung 2. Signal 5 ist die Schranke an der inneren Runde und Stellung 2 ist "Halt", also geschlossen. Die Ziffern, welche man in der Funktion `EEPSetSignal()` für die Signalstellung benutzt, entsprechen der Position der Signalstellungen in der Liste (Bild 33\_10), welche man in Objekteigenschaften () findet.



Bild 33\_10

Mehr passiert nicht, wenn die Bedingung erfüllt war. Denn in der nächsten Zeile steht das Schlüsselwort `else`, also das englische Wort für "ansonsten" und teilt dem Interpreter mit, dass die folgenden Aktionen nur auszuführen sind, wenn die Bedingung nicht erfüllt war.

Wenn `route` eine gerade Zahl gespeichert hat, dann ist das Ergebnis von `route % 2` nicht gleich 1 und die Prüfung ergibt `false`. In diesem Fall springt der Interpreter sofort zum `else` und macht dort mit den folgenden Anweisungen weiter:

```
print("Route 2")
EEPSetSwitch(1, 2)
EEPSetSignal(4, 2)
```

Die Funktion `EEPSetSwitch(1, 2)` stellt die Weiche 1 in Stellung 2, also Abzweig. Der Zug wird deshalb die äußere Runde fahren. Mit `EEPSetSignal(4, 1)` wird die Schranke an der äußeren Runde geschlossen.

Die nächsten Zeilen im Skript lauten

```
    end
end
```

Das Wort `end` ist ebenfalls ein Schlüsselwort aus der Syntax von Lua. Es kennzeichnet das Ende eines Anweisungsblocks. Die Definition einer Funktion kann, wie Sie gerade gesehen haben, mehrere Anweisungen enthalten. Auf eine `if`-Verzweigung können ebenfalls mehrere Anweisungen folgen.

Deshalb muss dem Interpreter gezeigt werden, wo der zugehörige Teil jeweils endet.

Im vorliegenden Fall ist die Definition von `SETROUTE1()` ein Block, in dem ein zweiter Block mit einer `if`-Verzweigung steckt. Das erste `end` schließt also den Block mit der `if`-Verzweigung ab und das zweite kennzeichnet das Ende der Funktionsdefinition. Denn ein innerer Block wird selbstverständlich beendet, bevor man den äußeren Block verlässt.

Vielleicht haben Sie sich schon gefragt, warum manche Zeilen in einem Skript etwas versetzt geschrieben sind? Durch diese Einrückungen verdeutlichen Programmierer die Blöcke, welche im Skript vorhanden sind. Dem Interpreter ist es egal ob Zeilen versetzt stehen. Aber für denjenigen, der ein Skript liest, sind sie eine große Hilfe. Und die Definition von `SETROUTE1()` zeigt sehr schön, wie hilfreich Einrückungen sein können.

Es folgt noch eine letzte Funktionsdefinition:

```
-- open signals
function OpenAllSignals()
    print("Open signals")
    EEPSetSignal(4, 1)
    EEPSetSignal(5, 1)
end
```

Die Funktion heißt `OpenAllSignals()` (auf Deutsch: Öffne alle Signale) und wird im Kontaktpunkt 1 (Bild 33\_11) aufgerufen, welcher oben hinter der Weiche 2 sitzt (Bild 33\_1).

Der Funktionsnamen unterscheidet sich in seiner Schreibweise deutlich vom ersten. Während `SETROUTE1()` komplett groß geschrieben wurde, wechseln sich in `OpenAllSignals()` die Groß- und

Kleinbuchstaben ab. Für den Interpreter macht das keinen Unterschied, solange man bei Funktionsdefinition und Funktionsaufruf exakt die gleiche Schreibweise verwendet. Die Schreibweise von `OpenAllSignals()` ist bei Programmierern aber sehr populär. Sie macht Funktionsnamen gut lesbar.

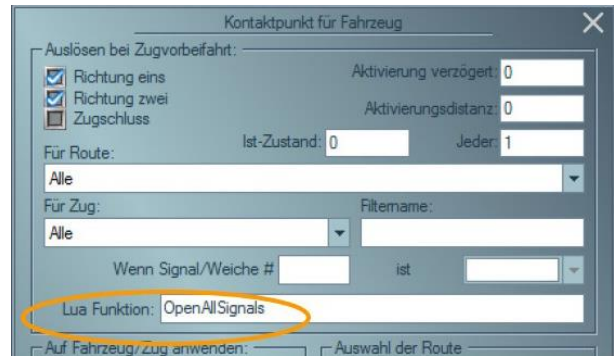


Bild 33\_11

Die Funktion `OpenAllSignals()` gibt einen Text aus und schaltet anschließend beide Schranken auf Fahrt. Alles in allem haben Sie jetzt viel Text zu einer sehr einfachen Anlage gelesen. Aber dafür haben Sie auch eine große Menge an Grundlagen gelernt. Und dieses Wissen sollten Sie verfestigen, indem Sie das Skript dieser Tutorial Anlage weiter verändern, bevor Sie sich den übrigen Tutorials widmen.

Sie können zum Beispiel den Modulo Operator genauer unter die Lupe nehmen. Ersetzen Sie doch mal `route % 2` durch `route % 3` und schauen, was sich dadurch ändert. Sie müssen den Zug mehrere Runden drehen lassen, bevor Sie den Unterschied feststellen können.

Auflösung:

Der Zug fährt nun einmal die innere und zweimal die äußere Runde. Denn mit `route % 3` bekommt man als Ergebnis die Zahlen 0, 1 und 2. Also liefert der Vergleich `(route % 3) == 1` einmal `true` und zweimal `false`.

Und wenn der Zug jede Runde zweimal fahren soll?

Dann kann man `(route % 4) < 2` schreiben. Denn `route % 4` liefert die Zahlen 0, 1, 2 und 3. Die ersten beiden sind kleiner als 2, die anderen zwei sind es nicht. Also ist die Bedingung zweimal `true` und zweimal `false`.

Es gibt noch viele weitere Experimente, die Sie versuchen können. Zum Beispiel können Sie absichtlich herbeiführen, dass die Schranken genau falsch stehen. Dass sie geöffnet sind, wenn der Zug durchfährt und geschlossen, wenn der Zug weg ist. Je mehr Varianten Sie durchspielen, desto mehr wird Ihnen der Umgang mit einer Skriptsprache in Fleisch und Blut übergehen.

Wichtig ist bei solchen Experimenten, dass Sie Fehler schätzen lernen. Fehler sind nicht ärgerlich, sondern im Gegenteil äußerst nützlich, weil sie Ihnen wertvolle Informationen liefern.

## Tutorial\_34\_LUA\_2 Callback(Rückruf)-Funktionen 1

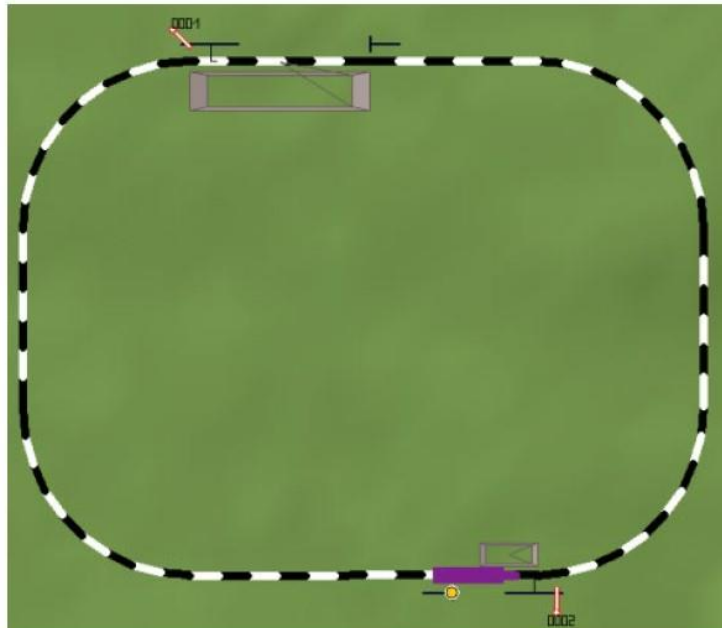


Bild 34\_1

Dieses Tutorial erläutert den Funktionsaufruf durch Signale.

- [EEPRegisterSignal\(\)](#)
- [EEPOnSignal\\_x\(\)](#)

Im ersten Tutorial haben Sie eine Menge Grundlagen kennengelernt. Außerdem haben Sie von zwei Methoden erfahren, mit denen EEP Funktionen aufrufen kann, die im Lua Skript definiert sind. Erstens die `EEPMain()` Funktion, welche automatisch fünf Mal je Sekunde aufgerufen wird. Und zweitens die Möglichkeit in einem Kontaktpunkt eine Funktion aufzurufen, deren Namen man selbst vergibt.

Die Anlage "Tutorial\_34\_LUA\_2" stellt Ihnen den dritten Weg vor, auf dem EEP eine Lua Funktion aufrufen kann: Einen Schaltvorgang bei Signalen oder Weichen.

Man nennt das eine "Callback"-Funktion. "Callback" ist das englische Wort für "Rückruf". Eine Callback-Funktion wird dann aufgerufen, wenn sich der Status eines Signals oder einer Weiche ändert. Man kann sie mit den Wenn-dann Verknüpfungen vergleichen, die in den Eigenschaften zu finden sind.

Die ersten Zeilen aus dem Skript kennen Sie schon. Zunächst wird eine Variable namens `I` initialisiert und dann ein Text ausgegeben. Die Variable `I` ist nur ein Überbleibsel aus dem Basisskript und in dieser Anlage ohne Bedeutung.

Die folgenden beiden Zeilen lauten

```
EEPRegisterSignal(1)
EEPRegisterSignal(2)
```

Die Funktion `EEPRegisterSignal()` aktiviert den Aufruf einer Callback-Funktion für ein Signal. Eine Funktion `EEPRegisterSwitch()` würde das selbe für eine Weiche tun. Die Zahl in den Klammern ist die Nummer des Signals oder der Weiche, die man aktivieren möchte. Die führenden Nullen können hier weggelassen werden.

Diese Aktivierung ist erforderlich, weil sonst jedes Signal und jede Weiche bei allen Schaltvorgängen eine Callback-Funktion aufrufen würde. Sie wären dann gezwungen für alle Signale und Weichen auf der Anlage entsprechende Funktionen in Ihr Skript zu schreiben damit der Aufruf dieser Funktionen keine Fehlermeldung zur Folge hat.

Nach der Registrierung folgt die zwingend notwendige Definition von `EEPMain()` und wie in der vorherigen Tutorial Anlage hat sie auch hier keine weitere Funktion als die, ein Lebenszeichen von sich zu geben.

```
function EEPMain()  
    return 1  
end
```

Den Rest des Skripts bilden die Definitionen der Funktionen `EEPOnSignal_1()` und `EEPOnSignal_2()`. Sie beschreiben was zu tun ist, wenn das jeweilige Signal umgeschaltet wird.

Die Funktion `EEPOnSignal_1()` wird aufgerufen, wenn man per Mausklick oder Kontaktpunkt das Signal 1 umschaltet. Einen Kontaktpunkt gibt es in dieser Tutorial Anlage nicht, also müssen Sie das Signal per Mausklick umschalten, um den Effekt zu beobachten.

Beim Funktionsaufruf gibt das Signal die neue Signalstellung mit. So, wie Sie bei `print()` einen Text in die Klammern schreiben, schreibt das Signal eine Zahl in die Klammern hinter dem Funktionsnamen. Diese Zahl muss man in der Funktionsdefinition auffangen, um sie weiter verarbeiten zu können. Deshalb steht in den Klammern hinter dem Funktionsnamen eine Variable namens `status`.

```
function EEPOnSignal_1(status)
```

Beachten Sie bitte, dass die Signal ID kein Parameter, sondern Teil des Funktionsnamens ist. Das bedeutet, dass jedes Signal eine eigene Callback-Funktion aufruft. Auf diese Weise vermeidet EEP, dass mehrere Signale bei gleichzeitigem Umschalten die Callback-Funktionen überschreiben. Kein Rückruf geht verloren.

Die Funktion gibt als erstes eine Meldung im Ereignisfenster aus.

```
print(" SIGNAL 1 CALLBACK STATUS: ")
```

Dann folgt eine Verzweigung, in welcher der Signalstatus geprüft wird.

```
    if (status == 1) then
```

Ist die Stellung des Signals 1, also Fahrt, dann wird ein entsprechender Text ins Ereignisfenster geschrieben

```
        print("          OPEN (", status, ")") ;
```

und das Signal 2 auf Stellung 2, also "Halt" gestellt.

```
EEPSetSignal(2, 2, 1)
```

Ansonsten

```
else
```

wird ein anderer Text ins Ereignisfenster geschrieben

```
print(" CLOSE (", status, ")" );
```

und das Signal 2 auf 1, also Fahrt gestellt.

```
EEPSetSignal(2, 1, 1)
```

Und damit ist das Ende der Verzweigung

```
end
```

und der Funktion

```
end
```

erreicht.

Für Signal 2 gibt es eine ähnliche, aber einfachere Funktion

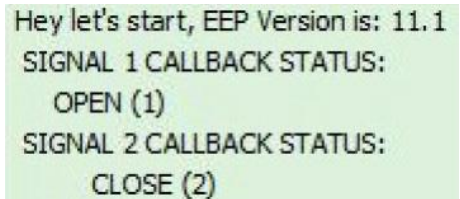
```
function EEPOnSignal_2(status)
    print(" SIGNAL 2 CALLBACK STATUS: ")
    if (status == 1) then
        print("          OPEN (", status, ")" );
    else
        print(" CLOSE (", status, ")" );
    end
end
```

Sie gibt nur je nach Signalstellung unterschiedliche Texte aus.

Wenn sie die Anlage starten, dann hält der Zug am Signal 1. Es steht auf Halt und das Signal 2 auf der anderen Seite des Ovals steht auf Fahrt. Wenn sie jetzt mit [Shift]-Klick das vordere Signal auf Fahrt stellen, dann wechselt das hintere Signal auf Halt.

Im Ereignisfenster stehen dazu die folgenden Zeilen (Bild 34\_2):

Mit einem weiteren [Shift]-Klick auf das vordere Signal stellen Sie es zurück auf Halt und das hintere wechselt auf Fahrt. Die Ähnlichkeit zur bekannten wenn-dann-Verknüpfung von Signalen ist leicht zu erkennen. Allerdings kann die Callback-Funktion viel mehr und das nächste Tutorial wird Ihnen einen kleinen Einblick dazu geben.



```
Hey let's start, EEP Version is: 11.1
SIGNAL 1 CALLBACK STATUS:
OPEN (1)
SIGNAL 2 CALLBACK STATUS:
CLOSE (2)
```

Bild 34\_2

Aber zuvor möchten wir Ihnen den Funktionsaufruf `EEPSetSignal()` in der Funktionsdefinition für `EEPOnSignal_1()` noch etwas genauer erklären. Denn in den Klammern stehen diesmal nicht zwei Parameter wie in der vorherigen Tutorial Anlage, sondern drei:

```
EEPSetSignal(2, 1, 1)
```

Die erste Zahl steht für das Signal, welches umgeschaltet werden soll.

Die zweite Zahl ist die gewünschte Signalstellung.

Und eine 1 als dritte Zahl bewirkt, dass das angesprochene Signal seinerseits auch die Callback-Funktion aufruft. Sie ist der Grund dafür, dass im Ereignisfenster neben den Texten zum Signal 1 auch die Texte zum Signal 2 stehen.

Zeit für ein kleines Experiment:

Löschen sie die dritte Zahl in beiden `EEPSetSignal()` Funktionen. Denken Sie bitte daran auch das Komma hinter der zweiten Zahl mit zu löschen. Sonst erwartet EEP eine dritte Zahl und beschwert sich, wenn diese ausbleibt.

Wenn Sie die Anlage jetzt neu starten, dann können Sie wieder mit [Shift]-Klick auf das vordere Signal beide Signale umschalten. Soweit hat sich nichts geändert. Aber wenn Sie sich den Text im Ereignisfenster anschauen, dann werden Sie feststellen, dass dort die Texte vom Signal 2 fehlen.

Stellen Sie das hintere Signal direkt um, indem sie daraufklicken, dann erscheinen die Texte. Die Callback-Funktion von Signal 2 existiert und funktioniert also weiterhin. Sie haben ja nichts daran geändert. Aber sie wird nur aufgerufen, wenn das Signal direkt oder per Kontaktpunkt umgestellt wird.

Möchte man, dass die Lua Funktion `EEPSetSignal()` in Folge eine Callback-Funktion aufruft, dann muss man das explizit verlangen, indem man bei Funktionsaufruf als dritten Parameter eine 1 mitgibt.

Sie können noch weitere Versuche mit dieser Tutorial Anlage anstellen.

Wie wäre es zum Beispiel, wenn Sie die Callback-Funktion für Signal 2 so erweitern, dass es beim Umschalten nicht nur einen Text ausgibt, sondern auch Signal 1 mit schaltet?

Dazu müssen Sie nur die passenden `EEPSetSignal()` Funktionen dort eintragen, wo bisher nur Text ausgegeben wird.

## Tutorial\_35\_LUA\_3 Callback(Rückruf)-Funktionen 2

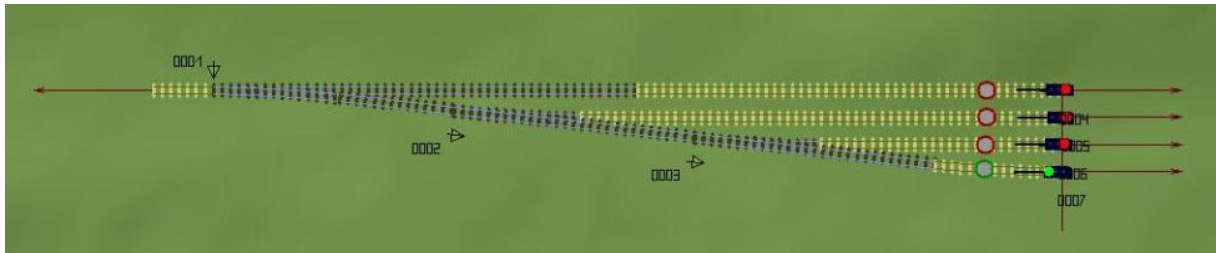


Bild 35\_1

Dieses Tutorial enthält ein zweites Beispiel für die Verwendung von Callback-Funktionen (auf Deutsch: Rückruf-Funktionen). Die verwendeten Methoden sind dieselben wie in der Anlage "Tutorial\_34\_LUA\_2".

- [EEPRegisterSignal\(\)](#)
- [EEPOnSignal\\_x\(\)](#)

Auf dieser Tutorial Anlage fährt keine Lok. Am Ende der Gleisharfe stehen vier Ampeln, welche Sie per [Shift]-Klick umschalten können. Beobachten Sie dabei bitte das Verhalten der übrigen drei Ampeln und der Weichen. Immer, wenn Sie eine der vier Ampeln auf Grün schalten wechseln die anderen auf Rot und die Weichen stellen sich so um, dass er Weg vom Einfahrgleis zur grünen Ampel führt.

Zunächst müssen die vier Ampeln registriert werden, damit sie beim Umschalten eine Callback-Funktion aufrufen. Signale (und Weichen), die nicht registriert sind, rufen bei Schaltvorgängen keine Lua Funktion auf.

```
EEPRegisterSignal(4)
EEPRegisterSignal(5)
EEPRegisterSignal(6)
EEPRegisterSignal(7)
```

Jedes Lua Skript für EEP benötigt die Definition der Funktion `EEPMain()`. In diesem Tutorial hat diese Funktion keine weiteren Aufgaben außer der, dass sie ein Lebenszeichen von sich gibt:

```
function EEPMain()
    return 1
end
```

Die Callback-Funktion für ein Signal heißt `EEPOnSignal_x()`, wobei das `x` im Namen durch die Nummer des Signals ersetzt werden muss. Die führenden Nullen einer Signalnummer müssen im Funktionsnamen weggelassen werden.

Diese Callback-Funktion muss im Skript definiert werden, damit Lua weiß was zu tun ist, wenn das betreffende Signal umgeschaltet wird.

```
function EEPOnSignal_4(status)
```

Die Variable `status` in den Klammern hinter dem Funktionsnamen nimmt den Signalbegriff auf, auf den das Signal umgestellt wurde.

In diesem Tutorial soll geprüft werden, ob die Ampel auf Grün geschaltet wurde.

```
if(status == 1) then
```

Wenn die Ampel auf Grün gestellt wurde, dann werden die Weiche 1 sowie die Signale 5, 6 und 7 umgestellt:

```
    print("Set route 1")
    EEPSetSwitch(1, 1)
    EEPSetSignal(5, 2)
    EEPSetSignal(6, 2)
    EEPSetSignal(7, 2)
```

Andernfalls geschieht nichts.

```
end
```

Und das ist das Ende der Definition der Callback-Funktion für Signal 4.

```
end
```

Die gleichen Callback-Funktionen müssen auch für die übrigen drei Ampeln definiert werden. Sie unterscheiden sich nur durch die Weichen und Signale, welche umgestellt werden sowie durch die Stellungen der Weichen:

```
function EEPOnSignal_5(status)
    if(status == 1) then
        print("Set route 2")
        EEPSetSwitch(1, 2)
        EEPSetSwitch(2, 2)
        EEPSetSignal(4, 2)
        EEPSetSignal(6, 2)
        EEPSetSignal(7, 2)
    end
end
```

```
function EEPOnSignal_6(status)
    if(status == 1) then
        print("Set route 3")
        EEPSetSwitch(1, 2)
        EEPSetSwitch(2, 1)
        EEPSetSwitch(3, 2)
        EEPSetSignal(4, 2)
        EEPSetSignal(5, 2)
        EEPSetSignal(7, 2)
    end
end
```

```
function EEPOnSignal_7(status)
    if(status == 1) then
        print("Set route 4")

        EEPSetSwitch(1, 2)
        EEPSetSwitch(2, 1)
        EEPSetSwitch(3, 1)

        EEPSetSignal(4, 2)
        EEPSetSignal(5, 2)
        EEPSetSignal(6, 2)
    end
end
```

Folgende Experimente können Sie mit diesem Tutorial durchführen:

Sie können die Ampeln im 2D Planfenster umstellen. Dabei werden Sie feststellen, dass die übrigen Signale und Weichen nicht reagieren. Das ist ein wesentlicher Unterschied zwischen den klassischen Verknüpfungen durch Wenn-dann Bedingungen in den Eigenschaften von Signalen und Weichen:

Lua arbeitet nur, wenn eine Anlage im 3D Modus ist.

Sie können die Registrierung einer Ampel deaktivieren, indem Sie die betreffende Zeile in einen Kommentar umwandeln:

```
-- EEPRegisterSignal(4)
```

Wenn Sie diese Ampel anschließend auf der Anlage bedienen, dann wird sie keinen Einfluss mehr auf die übrigen Ampeln und Weichen ausüben. Die nötige Funktion dazu ist zwar noch vorhanden, aber ein Signal, welches nicht registriert wurde, ruft keine Callback-Funktion auf.

Sie können den Namen einer Callback-Funktion ändern. Ersetzen Sie

```
function EEPOnSignal_4(status)
durch
function EEPOnSignal_x(status)
```

Wenn Sie anschließend das Signal 4 im 3D Modus bedienen, dann werden Sie eine Fehlermeldung erhalten. Denn dieses Signal ruft die Funktion mit Namen `EEPOnSignal_4()` auf. Und diese Funktion ist nun nicht mehr definiert. Definiert ist jetzt stattdessen eine Funktion `EEPOnSignal_x()`, welche jedoch nie aufgerufen wird.

Der Lua Interpreter stört sich nicht an Definitionen von Funktionen, welche nie aufgerufen werden. Aber der Aufruf einer Funktion, die nicht zuvor definiert wurde, führt zu einer Fehlermeldung.

Bitte denken Sie daran das Ereignisfenster zu öffnen ([Bild 33\\_5](#)), damit Sie bei Ihren Versuchen die Fehlermeldungen sehen können.

## Tutorial\_36\_LUA\_4 Tabellen und Zufallsgenerator

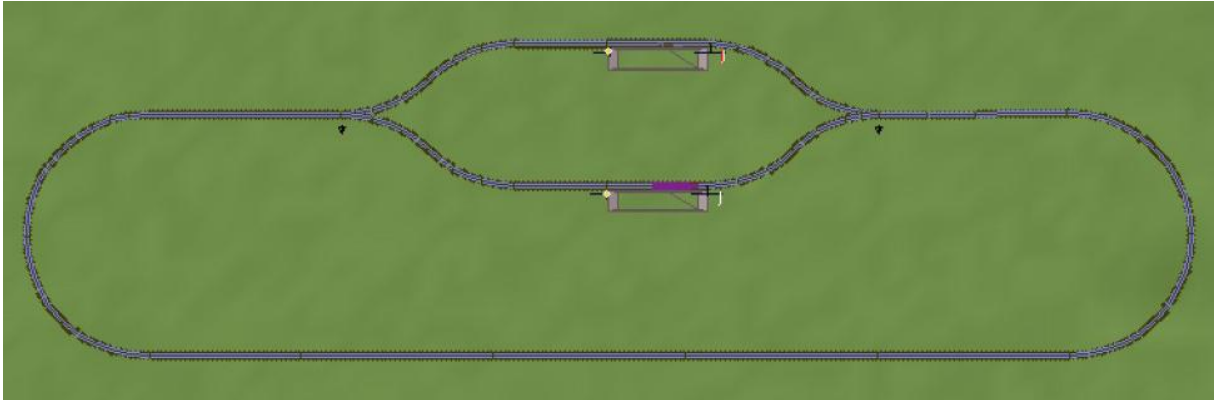


Bild 36\_1

Dieses Tutorial zeigt Ihnen ein einfaches Anwendungsbeispiel für die Verwendung von Tabellen in einem Lua Skript.

Vier Kontaktpunkte auf der Anlage rufen unterschiedliche Funktionen auf. Jede dieser vier Funktionen ändert den Text, der einmal pro Sekunde durch die [EEPMain\(\)](#) Funktion ausgegeben wird.

Zunächst werden zwei Tabellen definiert und beiden werden Elemente zugewiesen:

```
arr_lr = {"RIGHT", "LEFT"}  
arr_oc = {"OPEN", "CLOSE"}
```

Tabellen sind nichts anderes als Variablen mit mehreren Speicherplätzen. Weil diese Plätze alle unter einem Namen zusammengefasst sind, eignen sie sich gut für einen organisierten Umgang mit Daten.

*In vielen Computersprachen werden solche Datenfelder mit dem englischen Wort Array bezeichnet. Diese Arrays sind in der Regel numerisch indiziert. In Lua werden solche Datenfelder Tabellen (in Englisch: table) genannt. Sie unterscheiden sich von Arrays dadurch, dass sie Indizes jedes beliebigen Datentyps mit Ausnahme von `nil` erlauben. Neben Zahlen kann also auch mit Strings (Zeichenketten), booleschen Werten und sogar mit Funktionen indiziert werden. Auch die Werte können dabei jeden beliebigen Datentyp haben.*

*Leider wurden bei der Ersterstellung dieses Tutorials die beiden Tabellen `arr_lr` und `arr_oc` aus "alter Gewohnheit" mit dem Präfix `arr` für Array bezeichnet. Besser wären die Bezeichnungen `tab_lr` und `tab_oc` oder einfach nur `lr` und `oc` gewesen.*

Dann folgt die Definition von drei Variablen:

```
s = 0  
TXT_info_1 = "no train"  
TXT_info_2 = "no train"
```

Die erste Variable mit Namen `s` wird in der Funktion `EEPMain()` als Strichliste für die Durchläufe genutzt werden. Die anderen beiden werden unterschiedliche Texte für die Ausgabe bereithalten.

Die Funktion `EEPMain()` dient in diesem Tutorial als Timer. Da diese Funktion von EEP fünfmal je Sekunde (= alle 200 Millisekunden) aufgerufen wird, eignet sie sich gut für Aufgaben, die ständig wiederholt werden müssen. Wenn man die Durchläufe mitzählt, dann hat man einen Takt, an den man Aktionen binden kann.

```
function EEPMain()
    s=s+1
    if (s>5) then
        PrintInfo()
        s=0
    end
    return 1
end
```

Der Wert in der Variablen `s` wird bei jedem Durchlauf um 1 erhöht. Denn der aktuelle Wert wird ausgelesen, dann 1 addiert und das Ergebnis wieder derselben Variablen zugewiesen.

Anschließend wird geprüft, ob der Wert größer als 5 ist. Falls er das ist, dann wird eine Funktion namens `PrintInfo()` aufgerufen und der Variablen `s` wieder der Wert 0 zugewiesen. Ansonsten geschieht nichts. Zum Schluss gibt `EEPMain()` den Wert 1 an EEP zurück, damit sie erneut aufgerufen wird.

Die folgende Funktionsdefinition erstellt eine Funktion namens `SetSwitch()`, welche die Weiche 1 zufällig auf Fahrt oder Abzweig stellt. Aufgerufen wird diese Funktion durch den Kontaktpunkt kurz vor der Einfahrt in den zweigleisigen Bahnhof.

```
function SetSwitch1()
    current_state = math.random(1, 2)
    EEPSetSwitch(1, current_state)
end
```

Die Funktion `math.random()` gehört zur Lua-Bibliothek. Das bedeutet, dass der Interpreter die Definition kennt und die Funktion sofort benutzt werden kann. Sie erzeugt eine Zufallszahl zwischen den beiden Werten, welche in Klammern und durch ein Komma getrennt angegeben werden. Im Beispiel erzeugt die Funktion also zufällig eine 1 oder 2. Dieser Wert wird dann der Variablen `current_state` (auf Deutsch: "Ist-Zustand") zugewiesen. In der folgenden Zeile benutzt `EEPSetSwitch()` diese Variable, um Weiche 1 entweder auf Fahrt oder Abzweig zu stellen.

Die nachfolgend definierte Funktion `PrintInfo()` wird bei jedem fünften Durchlauf der `EEPMain()`-Funktion aufgerufen. Sie sammelt verschiedene Textzeilen und gibt sie im Ereignisfenster aus. Bitte denken Sie daran das Ereignisfenster zu öffnen ([Bild 33 5](#)), damit sie diese Textausgaben verfolgen können.

```
function PrintInfo()
```

Zuerst wird der aktuelle Inhalt des Ereignisfensters gelöscht.

```
clearlog()
```

Dann wird das aktuelle Datum und die Uhrzeit des PCs (nicht EEP-Zeit!) ermittelt und ausgegeben.

Die verwendete Funktion `os.date()` stammt ebenfalls aus der Lua-Bibliothek. Die Parameter für diese Funktion bestimmen den Ausgabebetext und das Format der Ausgabe.

```
print(os.date("current date and time: %c"))
```

Eine Leerzeile wird eingefügt

```
print("")
```

Der Status von Signal 4 wird ermittelt und in einer Variablen gespeichert.

```
signal4_state = EEPGetSignal(4)
```

Dann gibt `print()` eine Zeile aus, die sich aus zwei Teilen zusammensetzt. Zuerst ein festgelegter Text, dann ein Eintrag aus der Tabelle `arr_oc`. Der Wert in der Variablen `signal4_state` bestimmt, ob der erste oder zweite Eintrag in der Tabelle benutzt wird.

```
print("Signal 4 status: ", arr_oc[signal4_state])
```

Anschließend wird das Prozedere für Signal 3 wiederholt.

```
signal3_state = EEPGetSignal(3)  
print("Signal 3 status: ", arr_oc[signal3_state])
```

Dann wird erneut eine Leerzeile eingefügt ...

```
print("")
```

... und anschließend mit Weiche 1 genauso verfahren wie zuvor mit den beiden Signalen.

```
print( "Switch 1 state:  ", arr_lr[EEPGetSwitch(1)] )
```

Allerdings wurde hier auf die Variable als Zwischenspeicher verzichtet. Stattdessen wird die Stellung der Weiche in dem Augenblick ermittelt, in dem man diese Zahl benötigt um das richtige Element aus der Tabelle `arr_lr` zu wählen.

Es folgt eine weitere Leerzeile ...

```
print("")
```

... und zuletzt die Ausgabe zweier Texte, die in den Variablen `TXT_info_1` und `TXT_info_2` gespeichert sind.

```
print ("Station 1 info: ", TXT_info_1)
print ("Station 2 info: ", TXT_info_2)
end
```

Die nächsten zwei Funktionen werden durch Kontaktpunkte am Anfang der beiden Bahnhofsgleise aufgerufen. Sie weisen den Variablen `TXT_info_1` und `TXT_info_2` neue Texte zu.

```
function SetInfoFromCP1()
    TXT_info_1 = "train arrived to station 1"
end
```

```
function SetInfoFromCP2()
    TXT_info_2 = "train arrived to station 2"
end
```

(auf Deutsch: "Zug in Gleis **X** angekommen")

Und zuletzt folgen zwei Funktionen, die durch Kontaktpunkte am Ende der beiden Bahnhofsgleise aufgerufen werden. Wie die zwei Funktionen zuvor ändern sie die Texte, welche in den Variablen `TXT_info_1` und `TXT_info_2` gespeichert sind. Außerdem stellen sie das Signal am Gleis auf Halt.

```
function SetInfoFromCP3()
    TXT_info_1 = "train leave station 1"
    EEPSetSignal(4, 2)
end
```

```
function SetInfoFromCP4()
    TXT_info_2 = "train leave station 2"
    EEPSetSignal(3, 2)
end
```

(auf Deutsch: "Zug hat Gleis **X** verlassen")

## Tutorial\_37\_LUA\_5 Zeittakt, Geschwindigkeit, Kuppeln

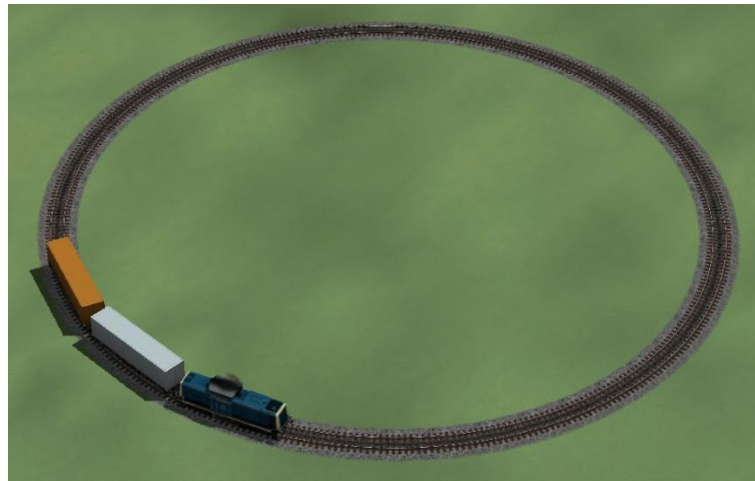


Bild 37\_1

- [EEPSetTrainSpeed\(\)](#)
- [EEPRollingstockSetCouplingRear\(\)](#)
- [EEPRollingstockGetCouplingRear\(\)](#)

In diesem Tutorial fährt eine Lok mit 2 Waggons im Kreis. Nach dem Start des Zuges wird er wieder abgebremst. Die Lok wird abgekuppelt und kurz von den Waggons entfernt. Danach wird die hintere Kupplung der Lok wieder scharf gestellt und die Lok an die Waggons herangefahren. Wenn Lok und Waggons wieder zusammengekuppelt sind beginnt das Spiel von vorne. Diese Aktionen werden jeweils über einen Zeitgeber ausgelöst.

Hierzu wird zu Beginn des Lua-Codes der Zeitgeber "cycle" (auf Deutsch: "Zyklus") auf 0 gestellt und alle Anzeigen im Ereignisfenster gelöscht.

```
cycle=0  
clearlog()
```

Die Funktion `EEPMain()` wird 5 mal pro Sekunde aufgerufen. Um `cycle` als Zeitgeber nutzen zu können, wird `cycle` bei jedem Aufruf der `EEPMain()` – also alle 1/5 Sekunden – um 1 hochgezählt.

```
function EEPMain()  
    cycle = cycle + 1
```

Dies wird nun dazu benutzt, um die Abfolge der Ereignisse zu programmieren. Dies geschieht in einer `if - elseif - else - end`-Verzweigung, die Ihnen schon aus dem Tutorial\_33\_LUA\_1 bekannt ist.

Direkt im 1. Zyklus (`cycle == 1`) wird der Zug "#DB 212 309" mit der Funktion `EEPSetTrainSpeed(Name, Geschwindigkeit)` auf eine Geschwindigkeit von 30 km/h gebracht. Hierbei ist zu beachten, dass die Namen von Zügen (und auch anderen Fahrzeugverbänden) als Text übergeben werden, deshalb steht er in Anführungszeichen. Außerdem beginnen Namen von Fahrzeugverbänden immer mit

einem #-Zeichen. (Hierdurch ist es möglich, dass Fahrzeugverbände den gleichen Namen wie ihr 1. Fahrzeug erhalten können; nur eben mit einem # Zeichen zur Unterscheidung.)

```
if (cycle == 1) then
    -- GO --
    hResult = EEPSetTrainSpeed("#DB 212 309", 30)
    print("GO at cycle: ", cycle, "result: ", hResult)
```

Im 30. Zyklus (`cycle == 30`) wird der Zug wieder angehalten, in dem die Geschwindigkeit auf 0 gesetzt wird.

```
elseif (cycle == 30) then
    -- STOP --
    hResult = EEPSetTrainSpeed("#DB 212 309", 0)
    print("STOP at cycle: ", cycle, "result: ", hResult)
```

15 Zyklen später, also wenn `cycle == 45` ist, wird mit der Funktion `EEPRollingstockSetCouplingRear(Name, °Kupplungszustand)` die hintere Kupplung der Lok gelöst. Hierzu muss der Kupplungszustand auf 2 gesetzt werden. 2 bedeutet: Kupplung inaktiv (abkuppeln). Der Name der Lok wird ebenso als Text eingegeben, nur hat er wie bei allen Fahrzeugen kein #-Zeichen.

Zusätzlich wird die Geschwindigkeit wieder auf 30 km/h gesetzt, so dass sich die Lok von den abgekuppelten Waggonen entfernt.

```
elseif (cycle == 45) then
    -- UNCONNECT --
    hResult = EEPRollingstockSetCouplingRear("DB 212 309", 2)
    print("UNCONNECT at cycle: ", cycle, "result: ",
        hResult)
    -- GO --
    hResult = EEPSetTrainSpeed("#DB 212 309", 30)
    print("GO at cycle: ", cycle, "result: ", hResult)
```

Weitere 10 Zyklen später (`cycle == 55`) wird die hintere Kupplung der Lok wieder "scharf" gestellt, in dem für den Kupplungszustand eine 1 (= Kupplung aktiv) eingegeben wird. Außerdem wird die Fahrrichtung des Zuges umgekehrt, in dem die Geschwindigkeit in `EEPSetTrainSpeed` negativ (d.h. mit -30) eingetragen wird.

```
elseif (cycle == 55) then
    -- ACTIVE CONNECTION --
    hResult = EEPRollingstockSetCouplingRear("DB 212 309",
        1)
    print("ACTIVE CONNECTION at cycle: ", cycle, "result: ",
        hResult)
    -- GO BACK --
    hResult = EEPSetTrainSpeed("#DB 212 309", -30)
    print("GO BACK at cycle: ", cycle, "result: ", hResult)
```

Nun fährt der Zug wieder mit einer "scharfen" hinteren Kupplung auf die Waggonen zu.

Ab dem nächsten Zyklus (`cycle > 55`) wird jedes Mal überprüft, ob die Lok wieder gekuppelt hat. Dies ist möglich, weil `EEPRollingstockGetCouplingRear` als 2. Rückgabewert – d.h. hier in die Variable `value` (auf Deutsch: Wert) – neben den bekannten Kupplungszuständen 1 = Kupplung aktiv und 2 = Kupplung = inaktiv den Wert 3 liefert, wenn eine gekuppelte Kupplung vorliegt.

```
elseif (cycle > 55) then
    hResult, value = EEPRollingstockGetCouplingRear("DB 212
    309")
```

Sowohl `EEPRollingstockGetCouplingRear` als auch `EEPSetTrainSpeed` und `EEPRollingstockSetCouplingRear` geben als ersten bzw. einzigen Wert zurück, ob die Funktion erfolgreich ausgeführt wurde.

Um sicher zu gehen, dass der Wert in `value` korrekt ist, sollte vorher überprüft werden, ob die Funktion erfolgreich ausgeführt worden ist. Wenn das der Fall ist, erfolgt die Prüfung, ob `value == 3`. Ist beides der Fall setzt man den "Timer" `cycle` wieder auf 0.

```
if(hResult) then
    if(value == 3) then
        -- DETECTED CONNECTION --
        print("DETECTED CONNECTION at cycle: ", cycle)
        print("RESET counter")
        cycle = 0
    end
end
```

Dies wird nicht direkt im 56. Zyklus der Fall sein, da ja die Lok erst im 55. Zyklus auf Rückwärtsfahrt geschickt wurde, aber irgendwann in einem Zyklus `> 55`.

Abschließend schließt man die `if – elseif –` Verzweigung mit `end` und die Funktion `EEPMain()` gibt den Wert 1 zurück, damit Lua weiterläuft.

```
end
return 1
end
```

Da jetzt `cycle == 0` ist, wird beim nächsten Aufruf `cycle` um 1 auf 1 erhöht und das Manöver beginnt von vorn.

## Tutorial\_38\_LUA\_6 Achsenslots

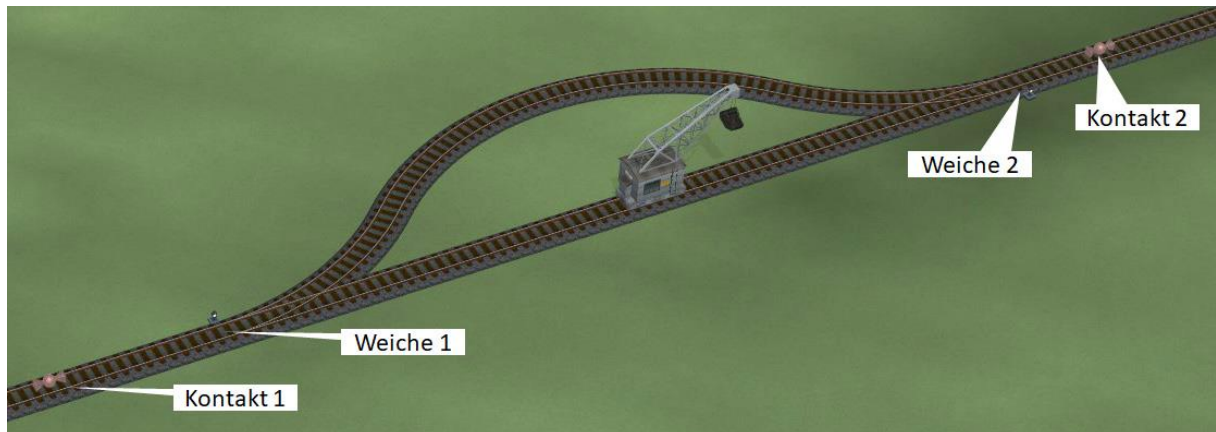


Bild 38\_1

Der Ladekran2 Greifer fährt hin und her. Dabei nimmt er einmal die gerade Strecke und auf der Rückfahrt die gebogene Strecke. Sein Greifer ragt dabei immer nach vorne in Fahrtrichtung aus. Die beiden entsprechenden Achsstellungen wurden bei der Erstellung der Anlage jeweils in einer Achsgruppe des Krans gespeichert (siehe unten).

In diesem Tutorial spielt die Funktion [EEPMain\(\)](#) keine Rolle. Aus diesem Grund wurde ihr Rückgabewert auf 0 gesetzt. Dadurch wird die Funktion nicht erneut aufgerufen. Alle anderen Funktionsaufrufe funktionieren aber weiterhin.

```
clearlog()
print("Hey let's start, EEP Version is: ", EEPVer)
function EEPMain()
    return 0
end
```

Die beiden Funktionen `Contact1` und `Contact2` werden beide über Kontaktpunkte aufgerufen. Hierzu wird in den Objekteigenschaften der jeweiligen Weiche der Funktionsname ohne Klammern eingetragen (rote Markierung in Bild 38\_2).

In Funktion `Contact1()` wird zuerst über [EEPRollingstockSetSlot\("Fahrzeugname", Achsengruppe\)](#) die in der Achsgruppe 1 des Fahrzeugs "Ladekran2 Greifer" gespeicherte Achsstellung eingestellt.

Danach wird mit [EEPSetTrainSpeed\("#Name", Geschwindigkeit\)](#) für den Zug (Fahrzeugverband) "#Ladekran2 Greifer" (beachten Sie hier das #-Zeichen im Namen) die Geschwindigkeit auf positive 30 km/h gesetzt.

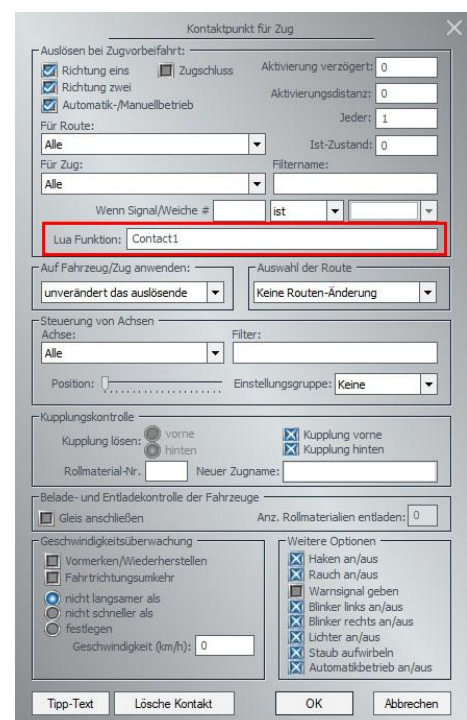


Bild 38\_2

Anschließend wird jeweils mit [EEPSetSwitch\(Weichennummer, Stellung\)](#) die Stellung der beiden Weichen so eingestellt, dass der Kran anschließend die gerade Strecke befährt.

```
function Contact1()
    EEPRollingstockSetSlot("Ladekran2 Greifer", 1)
    EEPSetTrainSpeed("#Ladekran2 Greifer", 30)
    EEPSetSwitch(1, 2)
    EEPSetSwitch(2, 1)
end
```

Fährt der Kran über den Kontakt 2, wird die Funktion `Contact2` aufgerufen. In ihr werden die gleichen EEP-spezifischen Funktionen nur mit unterschiedlichen Parametern verwendet. Jetzt wird die in der Gruppe 2 gespeicherte Achsstellung aktiviert. Die Geschwindigkeit wird auf -30 km/h gesetzt, so dass der Kran umkehrt. Und die Weichen werden so eingestellt, dass der Kran die gebogene Strecke befährt.

```
function Contact2()
    EEPRollingstockSetSlot("Ladekran2 Greifer", 2)
    EEPSetTrainSpeed("#Ladekran2 Greifer", -30)
    EEPSetSwitch(1, 1)
    EEPSetSwitch(2, 2)
end
```

Damit Sie das hier Gelernte in eigenen Anlagen anwenden können, müssen Sie natürlich noch wissen, wie man Achsen einstellt und diese Stellungen in einer Achsgruppe speichert.



Bild 38\_3

Klicken Sie im 3D-Fenster im manuellen Modus des Steuerdialogs auf die Schaltfläche für die Achsensteuerung (gelbe Markierung in Bild 38\_3). Wählen Sie dann in der Dropdown-Liste *Achsenauswahl* (rote Markierung) die zu setzende Achse aus. Verschieben Sie danach unter *Achsensteuerung* (blaue Markierung) mit der Maus den Einsteller auf der Skala von 0 bis 100% auf die gewünschte Position. Dies wiederholen Sie gegebenenfalls für alle weiteren Achsen, die Sie in einer Achsgruppe speichern möchten.

Wenn Sie alle Positionen festgelegt haben, klicken Sie auf die Aufgleis-Schaltfläche Editor (gelbe Markierung), und danach mit der rechten Maustaste auf das Fahrzeug, d.h. hier den Kran, um das Kontextmenü zu öffnen. Wählen Sie dort die Option *Achsenstellung speichern* (rote Markierung). Nun müssen Sie nur noch eine der 16 verfügbaren Gruppen auswählen. Ein Häkchen vor einer Gruppe bedeutet, dass sie bereits konfiguriert wurde (blaue Markierung).

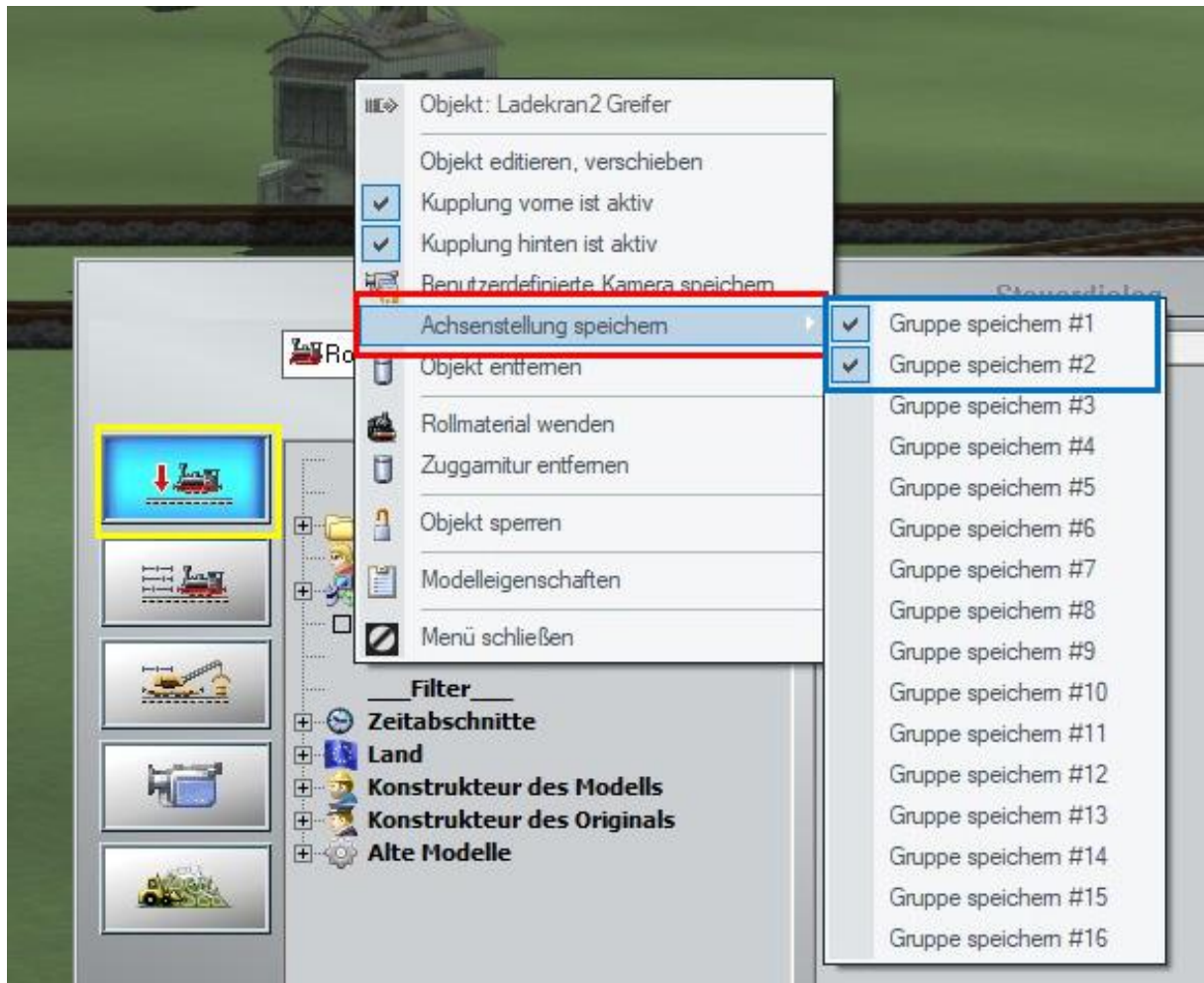


Bild 38\_4

(Siehe auch [EEP18-Handbuch Kapitel 6.5.4 Beladungsfunktion für Fahrzeuge](#).)

## Tutorial\_39\_LUA\_7 Rollmaterialachsen

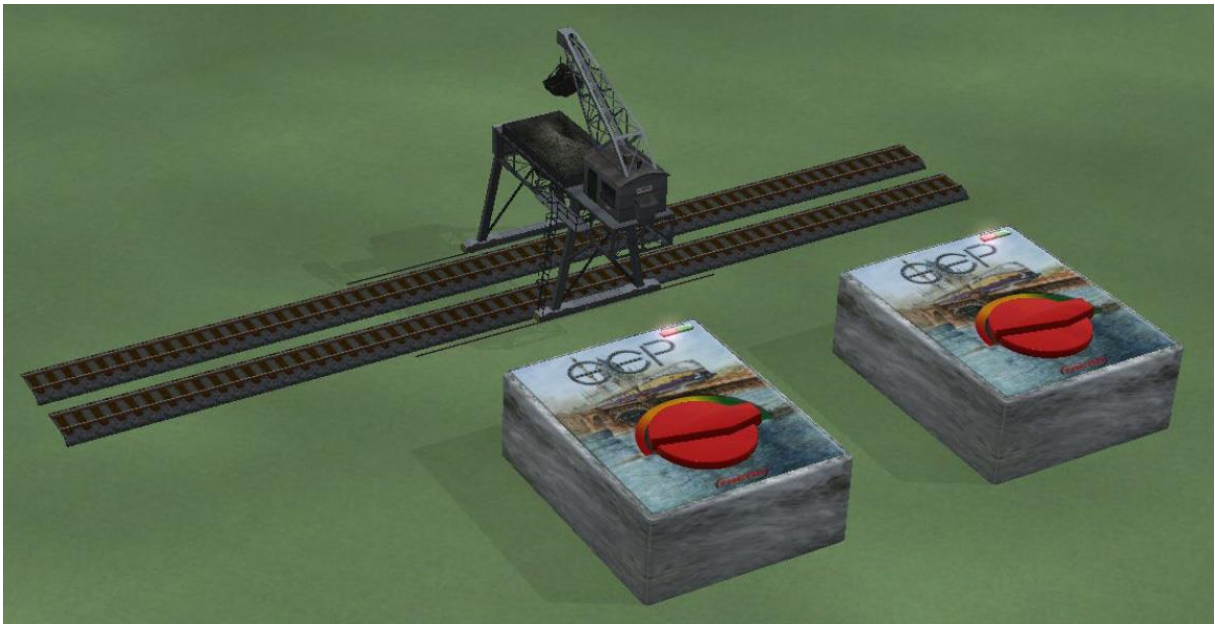


Bild 39\_1

Dieses Tutorial demonstriert die Achsensteuerung von Rollmaterialien mit Lua-Skripten

Wenn Sie den linken Schalter betätigen, werden alle Achsen der "Bekohlungskranbrücke 1" auf einmal angesteuert. Betätigen Sie dagegen den rechten Schalter, wird eine Achse nach der anderen angesteuert.

Der zeitliche Ablauf der Steuerung erfolgt über die Variablen `counter1` und `counter2`. `counter` ist das englische Wort für "Zähler". (Würde man anstatt des englischen Wortes das deutsche benutzen, müsste die Variable aber `Zaehler` heißen, denn Lua akzeptiert keine deutschen Umlaute als Variablennamen.) `counter1` steuert die Abfolge des linken Schalters und `counter2` des rechten. Beide werden am Beginn des Skripts initialisiert. Warum es sinnvoll ist, dies bei `counter2` mit 100 anstatt mit 0 zu tun, dazu später mehr.

```
counter1 =0
counter2 =100
clearlog()
print("Hey let's start, EEP Version is: ", EEPVer)
```

Wie bereits erwähnt, erfolgt der Start der Achsensteuerungen durch die Betätigung von einem der Schalter. Diese Schalter sind EEP-intern Signale. Um den Zeitpunkt einer Änderung der Signalstellung abzufangen, wurde in EEP die spezifische Lua-Funktion [EEPOnSignal X\(\)](#) implementiert, wobei das **X** für die Signalnummer (ohne vorangestellte Nullen) steht. Damit EEP aber weiß, dass es auf die Signaländerung reagieren muss, ist eine Registrierung dieser Signale erforderlich.

```
EEPRegisterSignal(1)
EEPRegisterSignal(2)
```

Die Funktion `EEPMain` wird alle 0,2 Sekunden aufgerufen und mit jedem Aufruf werden die Zähler um 1 erhöht.

```
function EEPMain()
    counter1 = counter1+1
    counter2 = counter2+1

    -- SWITCH OFF SIGNAL 1 AUTOMATICALLY --

    SIGNAL 1 AUTOMATISCH AUSSCHALTEN
```

Es ist natürlich sinnvoll, den Schalter nach einer Betätigung wieder zurückzusetzen. Dies kann mit Lua automatisch erfolgen.

Den Schalter können Sie bei gedrückter [Shift]-Taste auf 3 Arten einschalten: 1. durch drehen des roten Knopfes mit der Maus, 2. durch einfaches Anklicken des Drehknopfs oder 3. durch Anklicken der grünen LED rechts oben. Bei den letzten beiden Möglichkeiten dreht sich der Knopf selbständig. Während dies geschieht, läuft EEP weiter und die `EEPMain()` wird dabei auch alle 0,2 Sekunden aufgerufen.

Damit man überhaupt den Effekt des Einschaltens sieht, sollte man die automatische Rückstellung des Knopfes nicht sofort im 1. `EEPMain`-Zyklus starten, sondern erst einige Zyklen später, z.B. wenn `counter1 > 12` ist. Ab dann wird bei jedem `EEPMain`-Aufruf mit `EEPGetSignal` die Stellung des Signals 1 abgefragt. Ist der ermittelte Wert (englisch: `value`) gleich 1 (d.h. der Schalter ist "an"), wird mit `EEPSetSignal` das Signal 1 in die Stellung 2 ("aus") gebracht.

```
if(counter1 > 12) then
    value = EEPGetSignal(1)
    if (value == 1) then
        hResult = EEPSetSignal(1, 2)
    end
end

-- SWITCH OFF SIGNAL 2 AUTOMATICALLY --

SIGNAL 2 AUTOMATISCH AUSSCHALTEN
```

Der gleiche Code wird auch für das Signal 2 angewandt. Beachten Sie aber, dass der Rückgabewert `value` bei der Funktion `EEPGetSignal(1)` mit einem kleinen "v" und der Rückgabewert `Value` bei der Funktion `EEPGetSignal(2)` mit einem großen "V" beginnt. Das sind 2 verschiedene Variablen. Das ist in diesem Skript nicht zwingend nötig. Falls sie aber einmal in die Situation kommen, beide Variablen miteinander zu vergleichen, ist dies ein möglicher Weg.

```
if (counter2 > 12) then
    Value = EEPGetSignal(2)
    if (Value == 1) then
        hResult = EEPSetSignal(2, 2)
    end
end
```

```
-- CALL PROPER FUNCTION DEPENDING ON counter2 VARIABLE --
```

AUFRUF DER RICHTIGEN FUNKTION IN ABHÄNGIGKEIT DER VARIABLEN  
counter2

Da bei Betätigung des rechten Schalters (2) die einzelnen Achsbewegung in jeweils zeitlichem Abstand nacheinander erfolgen soll, muss dies über den Wert der Variablen counter2 erfolgen. Wenn er gleich 1 ist soll die 1. Achsbewegung stattfinden, ist er gleich 20 die 2., bei 30 die 3. und die 4. bei 60. Welche Achse in welche Stellung gebracht werden soll, ist in den Funktionen GoStep1 bis GoStep4 (siehe unten) definiert, die hier nur bei dem jeweiligen counter2-Wert aufgerufen werden.

```
if(counter2 == 1) then GoStep1()  
elseif(counter2 == 20) then GoStep2()  
elseif(counter2 == 30) then GoStep3()  
elseif(counter2 == 60) then GoStep4()  
end
```

Dies soll natürlich nicht sofort beim Anlagenstart los gehen, sondern erst nach Betätigung des rechten Schalters. Aus diesem Grund wurde counter2 am Anfang des Skripts mit 100 initialisiert. Einem Wert, der weit größer ist als der Wert für die letzte Achsbewegung.

Abschließend gibt die EEPMain den Wert 1 zurück, damit sie EEP weiterhin aufruft.

```
return 1  
end
```

Da die Schaltersignale registriert sind (siehe oben), wird bei jeder Zustandsänderung die dem Signal entsprechende Funktion [EEPOnSignal X\(\)](#) aufgerufen. In dem Parameter status wird die aktuelle Signalstellung (also die nach der Änderung) übergeben. In dieser Anlage ist bei beiden Schaltern natürlich die Stellung "an" (status == 1) abzufragen.

In EEPOnSignal\_1 besagt die ausgegebene Meldung *"Alle Achsen gleichzeitig einstellen"* was in ihr passiert. Die Achsbewegungen werden alle mit der Funktion [EEPRollingstockSetAxis\("Fahrzeugname", "Achsname", Stellung\)](#) ausgeführt.

```
function EEPOnSignal_1(status)  
  if(status == 1) then  
    print("Set all axis at the same time")  
    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",  
                           "Drehung links", 50)  
    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",  
                           "Greifer hoch/runter", 50)  
    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",  
                           "Greifer auf/zu", 0)  
    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",  
                           "Kohlenstaub Greifer", 0)
```

Am Ende der Funktion wird der Zähler counter1 wieder auf 0 gesetzt.

```
        counter1 = 0
    end
end
```

Auch in `EEPOnSignal_2` besagt die ausgegebene Meldung *"Sequenz Schritt für Schritt ausführen"* was passieren soll, wenn EEP sie aufruft. Hierfür reicht es aber aus, den entsprechenden Zähler `counter2` einfach auf 0 zu setzen; denn die zeitliche Abfolge der Sequenz wurde bereits am Ende der `EEPMain()` in einer `if-elseif-end`-Verzweigung mit dem Zähler `counter2` definiert (siehe oben).

```
function EEPOnSignal_2(status)
    if(status == 1) then
        print("run sequence step by step")
        counter2 = 0
    end
end
```

In der `if-elseif-end`-Verzweigung in der `EEPMain()` werden die Funktionen `GoStep1()`, `GoStep2()`, `GoStep3()` und `GoStep4()` aufgerufen. Hierin erfolgt Steuerung der Achsen ebenso mit der Funktion `EEPRollingstockSetAxis()`. Der Rückgabewert `state` ist *true*, wenn Rollmaterial und Achse existieren oder *false*, falls mindestens eins von beidem nicht existiert. Diese wird jeweils über einen `print()`-Befehl im Ereignisfenster ausgegeben.

Beachten Sie aber bitte, dass hier zum Teil andere Achsen gestellt werden als in der Funktion `EEPOnSignal_1`.

```
function GoStep1()
    state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
                                   "Greifer hoch/runter", 100)
    print("Step 1: ", state)
end
```

```
function GoStep2()
    state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
                                   "Drehung links", 0)
    print("Step 2: ", state)
end
```

```
function GoStep3()
    state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
                                   "Greifer auf/zu", 100)
    print("Step 3/1: ", state)

    state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
                                   "Kohlenstaub Greifer", 100)
    print("Step 3/2: ", state)
end
```

```
function GoStep4()  
    state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",  
                                   "Kohlenstaub Greifer", 0)  
    print("Step 4: ", state)  
end
```

Um den richtigen Ablauf der einzelnen Bewegung angezeigt zu bekommen, sollten Sie zuerst den linken Schalter, danach den rechten und dann wieder der linken Schalter u.s.w. betätigen.

## Tutorial\_40\_LUA\_8 Data-Slots

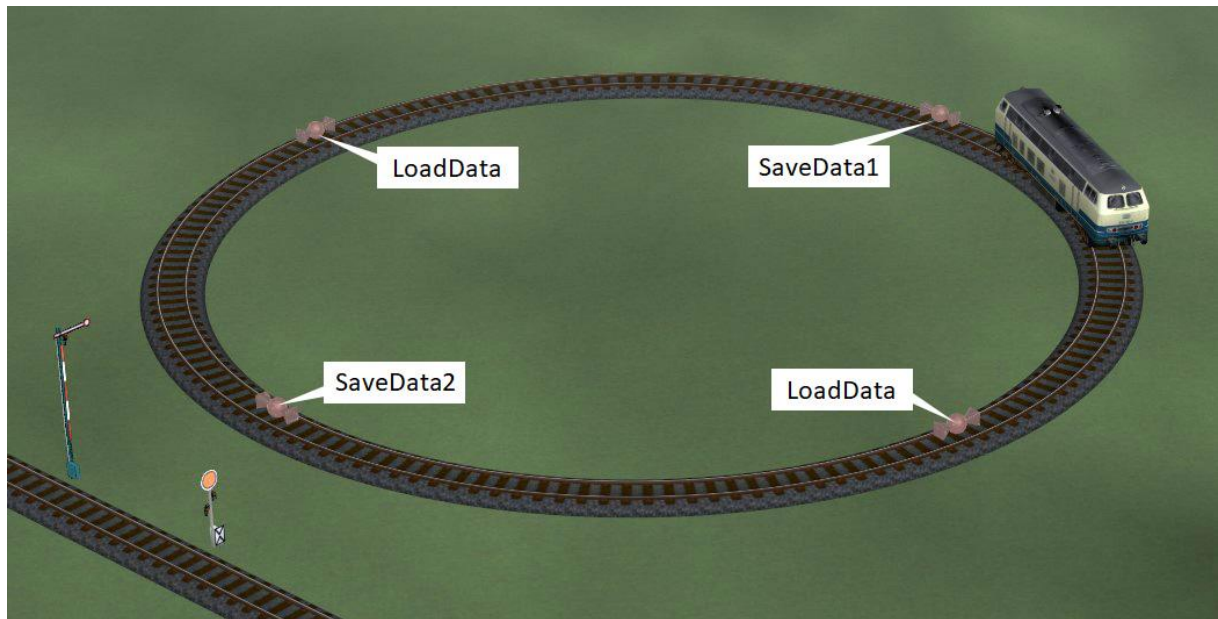


Bild 40\_1

EEP stellt in jeder Anlage 1000 Speicherplätze ("Slots") zur Verfügung, durchnummeriert von 1 bis 1000. Man kann in ihnen entweder Booleans, Zahlen oder Zeichenketten ("Strings") speichern, wobei letztere keine Formatierungszeichen enthalten dürfen.

Dieses Tutorial demonstriert, wie man mit

- [hResult = EEPSaveData\(Speichernummer, Wert\)](#)

einen Wert speichert und mit

- [hResult, Wert = EEPLoadData\(Speichernummer\)](#)

wieder ausliest.

Diese Funktionen können benutzt werden, um Daten mit der Anlagendatei (\*.anl3) zu speichern und zu laden. In diesem Beispiel werden Informationen und Werte im Ereignisfenster angezeigt - bitte aktivieren Sie dieses unter "Programmeinstellungen".

In dieser Anlage fährt eine Lok im Kreis und überfährt dabei nacheinander 4 Kontaktpunkte, in denen jeweils eine von drei im Lua-Skript definierte Funktionen aufgerufen werden (Bild 40\_1). Es sei noch einmal angemerkt, dass in den Kontaktpunkten die Funktionsnamen ohne Klammern dahinter eingetragen werden müssen.

Beim Anlagenstart werden zunächst die Variablen A, B und C vordefiniert und der Inhalt des Ereignisfensters gelöscht.

In der `EEPMain()` wird einmalig die Funktion `LoadData()` (siehe unten) aufgerufen. Danach wird die `EEPMain()` nicht mehr benötigt, da die Funktionen über

Kontaktpunkte aufgerufen werden. Aus diesem Grund wurde ihr Rückgabewert auf 0 gesetzt. Dadurch wird die Funktion nicht erneut aufgerufen. Alle anderen Funktionsaufrufe funktionieren aber weiterhin.

```
A = 0
B = 10.0
C = "EEP is COOL"
clearlog()
print("Hey let's start, EEP Version is: ", EEPVer)

function EEPMain()
    LoadData()
    return 0
end
```

In der Funktion `LoadData()` wird zunächst wieder der Inhalt des Ereignisfensters gelöscht. Danach wird der Inhalt des Speichers 1 ausgelesen und in die Variable `A` übertragen. War dies erfolgreich wird im Ereignisfenster die Meldung "*A loaded OK:*" (auf Deutsch: "*A geladen OK:*") ausgegeben und dahinter der Inhalt von `A`, also das, was im Speicher 1 steht. Anschließend wird das an dem Nebengleis stehende Signal 1 mit `EEPSetSignal` in die Stellung gebracht, die der Variablen `A` entspricht. War das Auslesen des Speicherplatzes nicht erfolgreich, wird die Meldung "*A no exist*" (auf Deutsch: "*A nicht vorhanden*") ausgegeben.

Als nächstes wird der Inhalt des Speichers 2 nach `B` und der des Speichers 3 nach `C` ausgelesen und mit den entsprechenden Meldungen angezeigt.

```
function LoadData()
    clearlog()
    hResult, A = EEPLoadData(1)
    if (hResult) then
        print("A loaded OK: ", A)
        EEPSetSignal(1, A)
    else
        print("A no exist")
    end

    hResult, B = EEPLoadData(2)
    if (hResult) then
        print("B loaded OK: ", B)
    else
        print("B no exist")
    end

    hResult, C = EEPLoadData(3)
    if (hResult) then
        print("C loaded OK: ", C)
    else
        print("C no exist")
    end
end
```

In den Funktionen `SaveData1` und `SaveData2` wird den Variablen `A` und `B` neue Werte zu gewiesen. In `C` wird immer der gleiche Text geschrieben. Vielleicht ändern sie ihn ja mal, um zu sehen was passiert.

Doch Vorsicht bei einer Änderung der Werte für `A`, denn über den Wert von `A` wird ja das Signal auf dem Nebengleis in der Funktion `LoadData` gesteuert (siehe oben).

```
function SaveData1()
    A = 1
    hResult, B = EEPGetTrainSpeed("#DB_218_202")
    C = "EEP is COOL"
    SaveData()
end

function SaveData2()
    A = 2
    B = 10
    C = "EEP is COOL"
    SaveData()
end
```

Am Ende der beiden Funktionen wird jeweils die Funktion `SaveData` aufgerufen. Hierin findet die Speicherung der Werte von `A`, `B` und `C` in den Speichern 1, 2 und 3 statt. Da der Code in beiden Funktionen derselbe wäre, ist es günstiger diesen in eine eigenständige Funktion auszulagern.

Ist die Speicherung mit `EEPSaveData` jeweils erfolgreich, wird die Meldung "**X** saved OK" (in Deutsch: (**X** gespeichert OK)) ausgegeben, wobei **X** für `A`, `B` bzw. `C` steht.

```
function SaveData()
    clearlog()
    hResult = EEPSaveData(1, A)
    if (hResult) then
        print("A saved OK")
    end
    hResult = EEPSaveData(2, B)
    if (hResult) then
        print("B saved OK")
    end
    hResult = EEPSaveData(3, C)
    if (hResult) then
        print("C saved OK")
    end
end
```

## Tutorial\_44\_LUA\_1 Rauch, Licht und Feuer in Immobilien

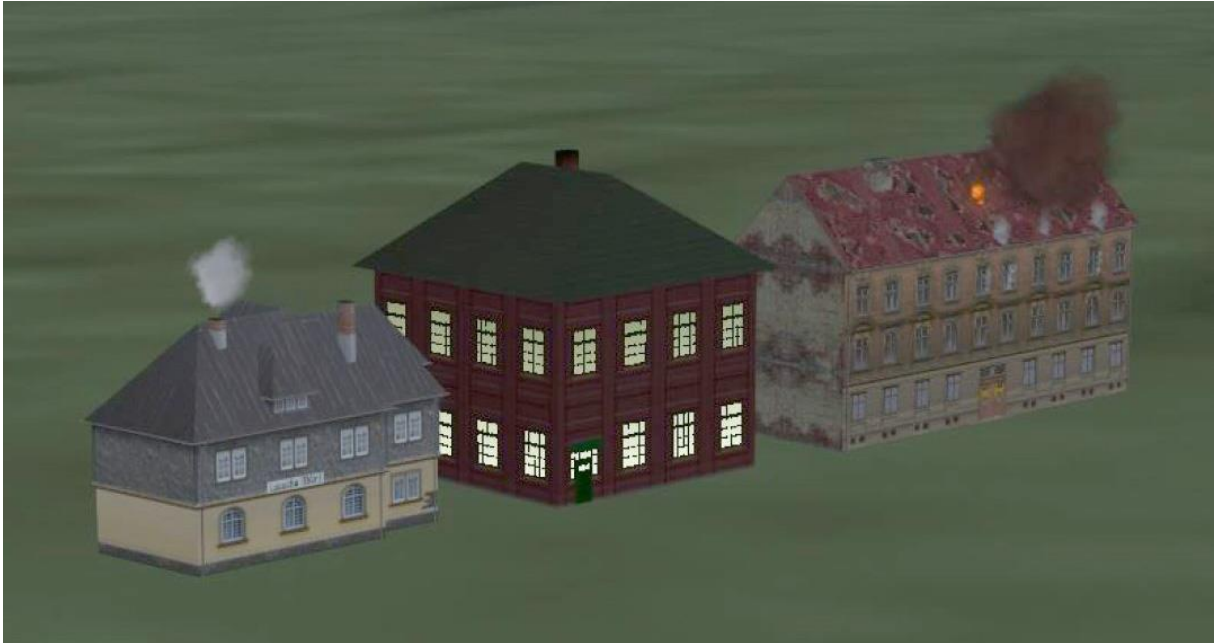


Bild 44\_1

Dieses Tutorial demonstriert die Steuerung der Immobilien-Eigenschaften Rauch, Licht und Feuer (Brand) mit den EEP-spezifischen Funktionen

- [EEPStructureSetSmoke\(\)](#)
- [EEPStructureSetLight\(\)](#)
- [EEPStructureSetFire\(\)](#)
- [EEPStructureGetSmoke\(\)](#)
- [EEPStructureGetLight\(\)](#)
- [EEPStructureGetFire\(\)](#)

Die Vorgänge auf der Anlage geschehen automatisch. Beim linken Gebäude wird der Rauch, beim mittleren das Licht und beim rechten das Feuer abwechselnd ein- und ausgeschaltet.

Zu Beginn des Skripts werden mehrere Variablen initialisiert:

```
I=0
hResult = 0      -- Variable um Ergebnisse aufzunehmen
IsSmoke = 0      -- Variable fuer die Rauchfunktion
IsLight = 0      -- Variable fuer die Lichtfunktion
IsFire = 0       -- Variable fuer die Feuerfunktion
```

Die Variable `hResult` wird die Erfolgsmeldungen der verschiedenen Funktionen aufnehmen. Sie erfüllt keinen weiteren Zweck.

In den Variablen `IsSmoke`, `IsLight` und `IsFire` wird gespeichert, ob die jeweilige Eigenschaft der Gebäude aktuell ein- oder ausgeschaltet ist.

*Die o.g. spezifischen EEP-Funktionen geben diese Werte als Booleschen Ausdruck zurück. Dieser kann nur zwei verschiedene Werte enthalten: **true***

(dt. wahr bzw. an) oder **false** (dt. falsch bzw. aus). Bei der Arbeit mit Lua ist es wichtig, zwischen diesen beiden Werten und Werten, die als **true** oder **false** ausgewertet werden, zu unterscheiden. In Lua gibt es nur zwei Werte, die als **false** ausgewertet werden: **nil** und **false**, während alles andere, einschließlich der numerischen 0, als **true** ausgewertet wird. In anderen Computersprachen ist das zum Teil nicht so.

Der Autor dieses Tutorials wollte diese Variablen mit "falsch" initialisieren, hat aber – wohl aus alter Programmiergewohnheit - durch Verwendung der numerischen 0 genau das Gegenteil getan. Richtig wäre folgende Initialisierung gewesen:

```
hResult = false    -- Variable um Ergebnisse aufzunehmen
IsSmoke = false    -- Variable fuer die Rauchfunktion
IsLight = false    -- Variable fuer die Lichtfunktion
IsFire = false     -- Variable fuer die Feuerfunktion
```

Die Funktion `EEPMain()` wird anschließend genutzt, um zu unterschiedlichen Zeiten die einzelnen Eigenschaften der Gebäude ein- und auszuschalten.

Die Variable `I` dient dabei als Zähler für die Durchläufe der `EEPMain()`-Funktion. Sie wird mit jedem Durchlauf um 1 erhöht. Mehrere `if`-Verzweigungen prüfen den Wert dieser Variablen und entscheiden danach, welche Aktionen durchzuführen sind.

```
function EEPMain()
    I=I+1

    if (I == 15) then
        SetSmoke(true)        -- Aufruf der Funtion
        SetLight(false)       -- Aufruf der Funtion
    else
        if (I == 30) then
            SetSmoke(false)    -- Aufruf der Funtion
            SetLight(true)     -- Aufruf der Funtion
        end
    end

    if (I == 10) then
        SetFire(true)         -- Aufruf der Funtion
    else
        if (I == 20) then
            SetFire(false)     -- Aufruf der Funtion
        end
    end
end
```

Abhängig davon, ob der Wert von `I` aktuell 15, 30, 10 oder 20 ist, ruft die `EEPMain()`-Funktion verschiedene Unterfunktionen auf. Dabei gibt sie entweder **true** oder **false** als Parameter mit.

Dann ermittelt sie bei jedem Durchlauf den aktuellen Zustand der Eigenschaften, speichert ihn in den Variablen `IsSmoke`, `IsLight` und `IsFire`:

```
hResult, IsSmoke = EEPStructureGetSmoke("#1_Lauscha")
hResult, IsLight = EEPStructureGetLight(
    "#2_Betriebsdienstgebäude")
hResult, IsFire = EEPStructureGetFire("#3_Brandhaus_01_SB1")
```

und gibt ihn als Text im Ereignisfenster aus.

```
print("Counter:", I, "Smoke: ", IsSmoke, " Light: ",
    IsLight, "Fire: ", IsFire)
```

Zuletzt wird der Zähler `I` auf 0 zurückgesetzt, wenn er den Wert 30 erreicht hat.

```
if (I == 30) then
    I = 0
end
```

Dann gibt `EEPMain()` den Wert 1 zurück, damit sie erneut von EEP aufgerufen wird.

```
return 1
end
```

Die folgenden drei Funktionen schalten Rauch, Licht und Feuer ein oder aus. Beim Aufruf wird jeder Funktion entweder der Wert `true` oder `false` mitgegeben. Dieser Wert wird in der Variablen `switch` (dt. *Schalter*) aufgefangen und dann in den EEP Funktionen verwendet, um die Eigenschaft entweder ein- (`true`) oder auszuschalten (`false`).

```
function SetSmoke (switch)
    EEPStructureSetSmoke("#1_Abfertigung Lauscha", switch)
end

function SetLight (switch)
    EEPStructureSetLight("#2_Betriebsdienstgebäude", switch)
end

function SetFire(switch)
    EEPStructureSetFire("#3_Brandhaus_01_SB1", switch)
end
```

Beachten Sie bitte, dass die Namen der Modelle, welche in den Funktionen als Parameter stehen müssen, im Gegensatz zu den wirklichen Modellnamen eine vorangestellte Nummer haben. Sie finden diesen "Lua-Namen" im Eigenschaften-Menü der Modelle. Bitte verwenden Sie für alle Lua-Funktionen, die Immobilien betreffen, nur diese Lua-Namen. Ohne die vorangestellte Nummer kann Lua die Immobilie nicht finden.

Die Nummer ist essenziell wichtig, der Namen dahinter hingegen optional.

## Tutorial\_45\_LUA\_2 Achsen-Animation in Immobilien und GOs



Bild 45\_1

Dieses Tutorial zeigt den Umgang mit animierbaren Achsen bei Immobilien und Gleisobjekten (GO) mit den EEP-spezifischen Funktionen

- [EEPStructureAnimateAxis\(\)](#)
- [EEPStructureGetAxis\(\)](#)

Vorne dreht eine Lok ihre Kreise und wird auf der Drehscheibe bei jedem Durchgang angehalten und um 180° gedreht. Hinten fährt ein Auto um die Mühle herum und schaltet mit zwei Kontaktpunkten die Mühle ein oder aus.

Zuerst werden eine Reihe Variablen im Skript initialisiert.

```
UNLIMITED = 1000      -- Variable für endlose Bewegung
hResult = 0            -- Variable um Ergebnisse aufzunehmen
Angle = 0              -- Variable für den Winkel der Drehscheibe
PreviousAngle = -1     -- Variable für den aktuellen Winkel der
                        Drehscheibe
Speed = 0              -- Variable für die Geschwindigkeit
```

*Da die o.g. EEP-spezifischen Funktionen das Ergebnis, ob Objekt und Achse existieren als Boolean zurückgeben, müsste hResult richtigerweise mit = **false** vordefiniert werden – wenn überhaupt. Siehe hierzu auch Anmerkung in [Tutorial 44 LUA 1](#).*

Die `EEPMain()` Funktion gibt bei jedem Durchlauf erneut den aktuellen Winkel der Drehscheibe aus und ruft die Funktion `CheckAndRunTrain()` auf. Dann gibt sie den Wert 1 zurück, damit sie erneut aufgerufen wird.

```
function EEPMain()
    hResult, Angle = EEPStructureGetAxis("#10_Drehscheibe",
                                         "Brücke")
    print("Turntable angle: ", Angle)

    CheckAndRunTrain() -- Aufruf der Funktion

    return 1
end
```

Ein Kontaktpunkt auf der Straße, welche um die Mühle herumführt, ruft die Funktion `CallFromContactPoint1()` auf. Darin befindet sich ein Funktionsaufruf, der die Mühle anhält.

```
function CallFromContactPoint1()
    EEPStructureAnimateAxis("#12_Windmühle", "Muehlrad", -1);
end
```

Die Funktion `EEPStructureAnimateAxis` benötigt drei Parameter: Den Namen der Immobilie, den Namen der Achse und einen Wert, der die Bewegung der Achse bestimmt. Beachten Sie bitte, dass die Namen der Modelle, welche in den Funktionen als Parameter stehen müssen, im Gegensatz zu den wirklichen Modellnamen eine vorangestellte Nummer haben. Sie finden diesen "Lua-Namen" im Eigenschaften-Menü der Modelle. Bitte verwenden Sie für alle Lua-Funktionen, die Immobilien betreffen, nur diese Lua-Namen. Ohne die vorangestellte Nummer kann Lua die Immobilie nicht finden.

Die Nummer ist essenziell wichtig, der Namen dahinter hingegen optional.

Der Wert -1 an dritter Stelle bewirkt, dass die Mühle angehalten wird.

Die folgende Funktion `CallFromContactPoint2()` wird ebenfalls durch einen Kontaktpunkt auf der Straße aufgerufen.

```
function CallFromContactPoint2()
    EEPStructureAnimateAxis("#12_Windmühle", "Muehlrad",
                           UNLIMITED);
end
```

Diese Funktion unterscheidet sich von der vorherigen nur dadurch, dass der dritte Parameter in `EEPStructureAnimateAxis()` diesmal die Variable `UNLIMITED` ist. In dieser Variablen wurde eingangs der Wert 1000 gespeichert. Die Zahl 1000 bewirkt eine endlose Drehung der angesprochenen Achse.

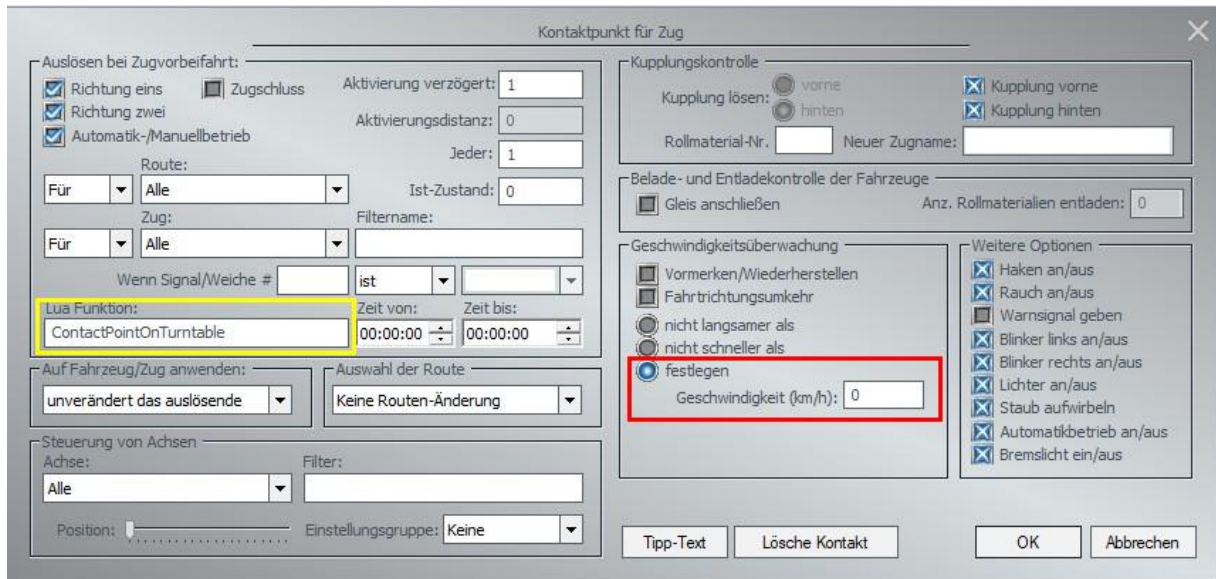


Bild 45\_2

Die folgende Funktion `ContactPointOnTurntable()` wird durch einen Kontaktpunkt aufgerufen, der mitten auf der Drehscheibe sitzt. Die Lok ruft also diese Funktion auf, wenn sie auf die Drehscheibe fährt. Der Kontaktpunkt setzt außerdem die Geschwindigkeit der Lok auf 0 (siehe Bild 45\_2).

```
function ContactPointOnTurntable()
    EEPStructureAnimateAxis("#10_Drehscheibe", "Brücke", 9)
end
```

Der erste Parameter der Funktion `EEPStructureAnimateAxis()` ist der "Lua-Name" der Drehscheibe, der zweite ist der Name der Achse. Der dritte Parameter gibt an, um wie viele Schritte sich die Drehscheibe weiterbewegen soll. Beim verwendeten Modell entsprechen 9 Schritte einer halben Drehung.

Die Funktion `CheckAndRunTrain()`, welche bei jedem Durchlauf von der `EEPMain()` aufgerufen wird, liest zuerst die aktuelle Position der Drehscheibe aus und speichert sie in der Variablen `Angle`. Dann ermittelt sie die aktuelle Geschwindigkeit der Lok und speichert sie in der Variablen `Speed`.

```
function CheckAndRunTrain()
    hResult, Angle = EEPStructureGetAxis("#10_Drehscheibe",
                                         "Brücke")
    hResult, Speed = EEPGetTrainSpeed("#DB 120-119 or EpIV")
```

Nachdem die aktuelle Position der Drehscheibe, eine in `PreviousAngle` gemerkte Position und die aktuelle Geschwindigkeit der Lok im Ereignisfenster ausgegeben wurden, prüft die Funktion, ob die Drehscheibe und die Lok zum Stillstand gekommen sind, um dann die Lok wieder in Bewegung zu setzen oder andernfalls gar nichts zu tun.

```
print("Angle:", Angle, "Previous Angle:", PreviousAngle,
      "Speed:", Speed)
```

```
if(Angle == PreviousAngle and Speed == 0) then
    EEPSetTrainSpeed("#DB 120-119 or EpIV", 50)
end
```

Zuletzt wird der aktuelle Winkel der Drehscheibe an die Variable `Previous Angle` übergeben. Solange die Drehscheibe in Bewegung ist, wird sich der in jedem Durchgang neu ermittelte Positionswert in `Angle` von dem, der hier auf `PreviousAngle` übertragen wurde unterscheiden. Diese Tatsache macht sich der Vergleich oben zunutze, um zu prüfen, ob die Drehscheibe stillsteht.

```
    PreviousAngle = Angle
end
```

Für den aller ersten Vergleich war die Variable `Previous Angle` ganz am Anfang des Skripts auf den Wert `-1` vordefiniert worden, ein Wert, der außerhalb des normalen Bereichs liegt.

## Tutorial\_46\_LUA\_3 Immobilien positionieren und Achsen setzen



Bild 46\_1

Dieses Tutorial demonstriert die Möglichkeit Immobilienachsen plötzlich, also ohne Animation in eine neue Position zu bringen. Es zeigt weiterhin, wie Immobilien mittels Lua auf der Anlage verschoben werden können.

- [EEPStructureSetAxis\(\)](#)
- [EEPStructureGetAxis\(\)](#)
- [EEPStructureSetPosition\(\)](#)

Der Wasserkran auf der linken Seite läuft automatisch. Er dreht sich über den Teich und das Wasser läuft, bis der Teich voll ist. Dann dreht der Wasserkran sich zurück und der Teich läuft wieder leer.

Den Wasserkran auf der rechten Seite können Sie selbst bedienen. Drehen Sie ihn mit einem Mausklick über den Teich und der wird sich füllen.

Zunächst werden einige Variablen initialisiert.

```
WaterLevel = -8          -- Variable für den Wasserstand
FillTempo = 0.3          -- Variable für die
                           Füllgeschwindigkeit
Water = 1                 -- Variable für das Wasser selbst
FillNone = 0              -- Variable für die Füllung
FillUp = 1                -- Variable, ob gefüllt werden soll
FillOut = 2               -- Variable, ob geleert werden soll
WaterLevelCondition = -8  -- Variable zum Wasserzustand
AxisValueCondition = 50   -- Variable zum Achswert
hResult = 0               -- Variable um Ergebnisse aufzunehmen
```

*Da die o.g. EEP-spezifischen Funktionen das Ergebnis, ob Objekt und Achse existieren als Boolean zurückgeben, müsste hResult richtigerweise mit*

= **false** vordefiniert werden – wenn überhaupt. Siehe hierzu auch Anmerkung in [Tutorial 44 LUA 1](#).

Die erste der beiden Funktionen, welche EEPMain() aufruft, regelt die Animation auf der linken Seite. Die zweite ist für den Wasserkran und den Teich auf der rechten Seite zuständig.

Anschließend gibt EEPMain() die Zahl 1 zurück, damit EEP sie erneut aufruft.

```
function EEPMain()  
    ControllWaterFillingCycle()      -- Aufruf der Funktion  
    ControllWaterFillingCondition()  -- Aufruf der Funktion  
    return 1  
end
```

ControllWaterFillingCycle() prüft, ob der linke Teich gefüllt werden oder leer laufen soll. Sie liest den Wert der Variablen Water, vergleicht ihn mit den Variablen FillUp und FillOut und ruft anschließend eine Funktion Waterfill() mit dem Parameter Filltempo auf. Einmal mit und einmal ohne vorangestelltes Minus.

Durch die Verwendung der Variablen FillTempo können Sie die Geschwindigkeit leicht ändern, indem Sie ihr bei der Initialisierung einen anderen Wert zuweisen. Sie müssen nicht das ganze Skript durchsuchen und jeden Eintrag einzeln ändern, der das Fülltempo bestimmt, sondern ändern die Zahl nur dort, wo sie der Variablen zugewiesen wird.

```
function ControllWaterFillingCycle()  
    if( Water == FillUp ) then WaterFill( FillTempo )  
    else  
        if( Water == FillOut ) then WaterFill( -FillTempo ) end  
    end  
end
```

Die eigentliche Animation steckt in der Funktion WaterFill() und die ist entsprechend etwas komplexer.

```
function WaterFill( fill )
```

Weil der Teich keine Animationsachse hat um den Pegel ansteigen zu lassen, versetzt diese Funktion ihn bei jedem Aufruf ein kleines Stück nach oben oder unten. Dafür muss die neue vertikale Position aus der alten gebildet werden:

```
WaterLevel = WaterLevel + fill
```

Dann bekommt der Teich seine neue Position. Dafür müssen alle drei Positionswerte übertragen werden und nicht nur die Höhe, welche verändert werden soll. Die Funktion verlangt alle vier Parameter: Den Namen der Immobilie und die X-, Y- und Z-Position.

Den Namen wurde in dieser Dokumentation gekürzt, weil die Zeile sonst zu lang wäre. Tatsächlich würde die Nummer der Immobilie auch ausreichen. Der Name dahinter ist optional.

```
EEPStructureSetPosition("#2", 2.16, -13.26, WaterLevel)
```

Anschließend wird geprüft, ob der Teich voll ist. Falls ja, dann wird die Funktion `StopFilling()` aufgerufen.

```
if ( Water == FillUp ) then
    if (WaterLevel > -2.0) then
        StopFilling()
    end
else
```

Ebenso wird geprüft, ob der Teich leer ist. Falls ja, dann wird die Funktion `StartFilling()` aufgerufen

```
    if ( Water == FillOut ) then
        if (WaterLevel < -8.0) then
            StartFilling()
        end
    end
end
```

Zuletzt wird der äußere `if`-Block und die Funktion beendet.

```
end
end
```

Die Funktion `StopFilling()` stoppt den Wasserfluss und verdreht den Wasserkran.

```
function StopFilling()
    EEPStructureSetAxis("#1_Wasserkran", "Wasser", 0)
    EEPStructureSetAxis("#1_Wasserkran", "Dreharm", 50)
```

Und dann wird die Variable verändert, in der EEP sich merkt, ob der Teich befüllt oder geleert wird.

```
    Water = FillOut
end
```

Die Funktion `StartFilling()` arbeitet nach demselben Prinzip.

```
function StartFilling()
    EEPStructureSetAxis("#1_Wasserkran", "Wasser", 100)
    EEPStructureSetAxis("#1_Wasserkran", "Dreharm", 0)
    Water = FillUp
end
```

Damit ist alles erklärt, was den Zyklus auf der linken Seite steuert.

Der Wasserkran auf der rechten Seite ist für eine manuelle Bedienung gebaut. Die Funktion `ControllWaterFillingCondition()`, welche die `EEPMain()` bei jedem Durchlauf aufruft, steuert sein Verhalten.

```
function ControllWaterFillingCondition()
```

Zunächst wird geprüft, ob der Wasserkran über dem Teich steht oder nicht. Hierzu wird die Position ermittelt und im Ereignisfenster ausgegeben.

```
    hResult, AxisValueCondition = EEPStructureGetAxis("#3",
        "Dreharm")
```

```
print("Axis: ", AxisValueCondition)
```

Dann wird nachgeschaut, ob die Drehung mindestens den Wert 80 hat. Wenn ja, dann wird geprüft, ob noch Wasser in den Teich passt. Da der Teich keinen echten Füllstand hat, sondern die Füllung des Teichs durch die Höhe der Immobilie simuliert wird, prüft die if-Verzweigung, ob die Höhe noch unter -2 Metern liegt.

```
if( AxisValueCondition > 80 ) then
    if ( WaterLevelCondition < -2) then
```

In diesem Fall wird der Wasserfluss beim Wasserkran gestartet und der Wert für die vertikale Position der Wasseroberfläche um 0,75 Meter angehoben.

```
EEPStructureSetAxis("#3_Wasserkran", "Wasser", 100)
WaterLevelCondition = WaterLevelCondition + 0.75
```

Dann wird die Immobilie neu positioniert.

```
EEPStructureSetPosition("#4", 4.33, 24,
    WaterLevelCondition)
```

Ist die Höhe der Immobilie nicht unter -2 Metern, dann passiert nichts weiter.

```
end
```

Da die ganze Funktion `ControllWaterFillingCondition()` von `EEPMain()` immer wieder aufgerufen wird, steigt der Pegel im Teich mit jedem Durchlauf ein Stückchen mehr, bis die maximale Füllhöhe erreicht ist.

Ist der Wasserkran hingegen nicht über dem Teich, dann wird beim Wasserkran der Wasserfluss gestoppt und geprüft, ob im Teich noch Wasser ist.

```
else
    EEPStructureSetAxis("#3_Wasserkran", "Wasser", 0)
    if ( WaterLevelCondition > -8 ) then
```

Wenn ja, dann wird der Pegel gesenkt. Das Prinzip ist identisch mit dem, das zum Befüllen verwendet wurde.

```
WaterLevelCondition = WaterLevelCondition - 0.5
EEPStructureSetPosition("#4", 4.33, 24,
    WaterLevelCondition)
end
```

Wenn nicht, dann passiert nichts weiter.

```
end
end
```

## Tutorial\_48\_Lua

## Rauch, Licht und Warnton eines Zuges

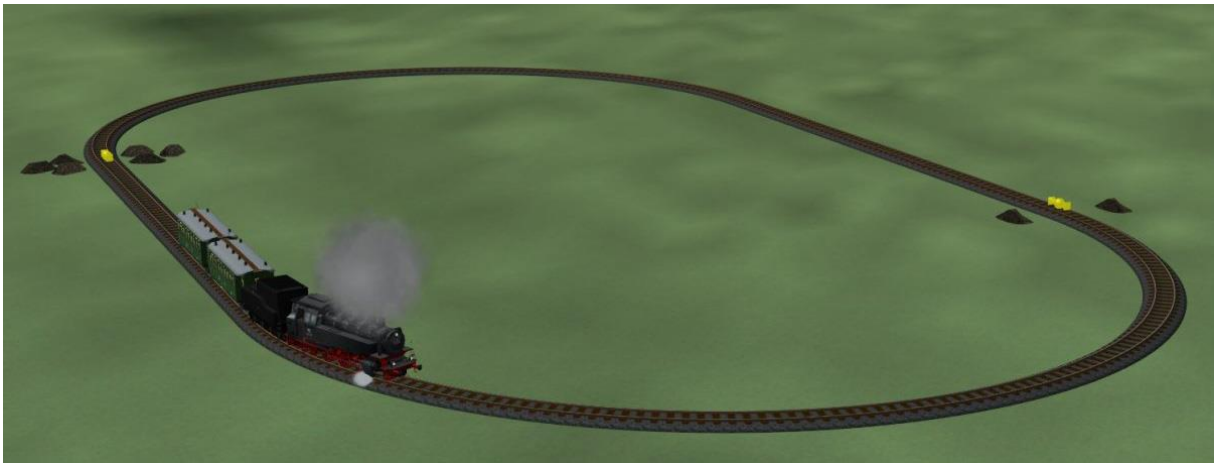


Bild 48\_1

Dieses Tutorial erläutert Ihnen die Verwendung der EEP Funktionen für Licht, Rauch und den Warnton eines Zuges.

- [EEPSetTrainLight\(\)](#)
- [EEPSetTrainSmoke\(\)](#)
- [EEPSetTrainHorn\(\)](#)

Es gibt zwei Kontaktpunkte auf der Anlage, die jeweils eine Lua Funktion aufrufen. Kontaktpunkte geben seit Einführung des Plug-In 2 zu EEP 11 bei Aufruf den Namen des auslösenden Zuges mit.

In der Funktionsdefinition wird der Name in einer Variablen `name` aufgenommen:

```
function TurnOnEffects(name)
```

Die Zeile

```
hResult1 = EEPSetTrainLight(name, true)
```

schaltet das Licht ein. Als Parameter benötigt die Funktion den Namen des Zuges und ein `true` um das Licht einzuschalten. Bei Erfolg liefert die Funktion ein `true` zurück, andernfalls ein `false`.

Dieser Rückgabewert wird in der Variablen `hResult1` aufgenommen und ein paar Zeilen später mit `print()` ausgegeben.

Die Zeilen

```
hResult2 = EEPSetTrainSmoke(name, true)
hResult3 = EEPSetTrainHorn(name, true)
```

funktionieren auf die gleiche Weise und schalten den Rauch und den Warnton ein.

```
    print("Einschalten von  Licht (", hResult1, "), Rauch (",  
        hResult2, " )    und Signalhorn (", hResult2,")")  
end
```

Die bereits erwähnte Ausgabe der Rückgabewerte beschließt die Funktion.

In der folgenden Funktion `TurnOffEffects(name)` wird dasselbe Prinzip verwendet, um Licht und Rauch auszuschalten. Der Warnton wird nicht ausgeschaltet, weil er zu diesem Zeitpunkt längst verklungen ist.

```
function TurnOffEffects(name)  
    hResult1 = EEPSetTrainLight(name, false)  
    hResult2 = EEPSetTrainSmoke(name, false)  
    print("Ausschalten von Licht (", hResult1, ") und Rauch (",  
        hResult2, " )")  
end
```



Bild 49\_1

Dieses Tutorial demonstriert die Steuerung von beweglichen Achsen eines Zuges und das Ein- und Ausschalten eines Hakens oder Greifers für Ladegüter

- [EEPSetTrainAxis\(\)](#)
- [EEPSetTrainHook\(\)](#)

Der Kranzug fährt über einen Kontaktpunkt und ruft damit eine Funktion auf, die ihn stoppt, den Ausleger in Position bringt, die Kiste aufnimmt und an anderer Stelle wieder absetzt. Zum Schluss kehrt der Ausleger in seine Ruhelage zurück.

**ACHTUNG:** Bei den EEP-Versionen bis einschließlich EEP 16 hat sich im Lua-Skript dieser Tutorialanlage leider ein Fehler teufel eingeschlichen. Der führt dazu, dass der Kran zwar seine Bewegungen vollzieht, aber nicht die Kiste aufnimmt und daher auch nicht wieder absetzt.

Bitte fügen Sie zur Korrektur im Lua-Skripteditor alle im Folgenden rot markierten Zeilen an den entsprechenden Stellen ein und löschen Sie die hier durchgestrichenen Zeilen.

EEP 17 enthält bereits ein korrigiertes Skript.

Zu Beginn des Skripts werden mehrere Variablen initialisiert.

```
TimeCounter = 0
TrainName = ""
bCraneIsReady = false
```

Die Variable `TimeCounter` dient als Zähler und wird bei jedem Aufruf der `EEPMain()`-Funktion um 1 erhöht. Dieser Zähler dient im Beispiel dazu den Zeitpunkt zu bestimmen, an dem eine bestimmte Aktion durchgeführt werden soll.

Die Variable `TrainName` wird dazu dienen den Namen des Zuges zu speichern. Die Variable `bCraneIsReady` dient als Merker dafür, ob der Zug in Ausgangsposition ist.

Der komplette Ablauf dieser Tutorial Anlage ist wie folgt:

Der Kranzug fährt über einen Kontaktpunkt, welcher die Lua Funktion `TakeGoodsStart(name)` aufruft. Sie dient dazu den Kranzug anzuhalten und den Ausleger über der Kiste zu positionieren.

Funktionsaufrufe aus Kontaktpunkten geben seit Einführung des Plug-In 2 zu EEP 11 immer den Namen des auslösenden Zuges mit. Im Beispiel nimmt die Variable `name` diesen Namen auf.

Wenn Sie das Eigenschaftsfenster des Kontaktpunktes öffnen, dann werden Sie dort nur den Funktionsnamen im Feld "Lua Funktion" finden, aber keine Klammern dahinter und keinen Parameter. Den Zugnamen ergänzt EEP selbständig. In den Kontaktpunkten darf wie zuvor nur der Name der Funktion stehen!

Die Funktion für den Kontaktpunkt ist im Skript wie folgt definiert:

```
-- step 1 - stop vehicle(train) and turn crane arm
--                                     [in deutsch: Schritt 1 - Fahrzeug(Zug) anhalten und Kranarm drehen]

function TakeGoodsStart(name)
    TrainName = name

    hResult = EEPSetTrainSpeed(name, 0)
    print("-- Kran anhalten: ", hResult)

    hResult = EEPSetTrainAxis(name, "Haupthaken senken/heben",
    80)
    print("-- Haken runter : ", hResult)

    hResult = EEPSetTrainAxis(name, "Ausleger heben/senken",
    100)
    print("-- Ausleger hoch: ", hResult)

    hResult = EEPSetTrainAxis(name, "Drehung nach rechts", 90)
    print("-- Kran dreht rechts: ", hResult)

    hResult = EEPSetTrainHook(name, false)
    print("-- Hook Funktion ausschalten: ", hResult)

    TimeCounter = 0
    bCraneIsReady = true
end
```

Die Zeile `TrainName = name` sorgt dafür, dass der Name des Zuges auch nach Verlassen der Funktion noch zur Verfügung steht. Er wird später in der Funktion `EEPMain()` benötigt.

Die Funktion `EEPSetTrainSpeed(name, 0)` stoppt den Kranzug und speichert den Erfolg der Aktion in der Variablen `hResult`.

Die Funktion `EEPSetTrainAxis(name, "Ausleger heben/senken", 100)` hebt den Ausleger an.

Die Funktion `EEPSetTrainAxis(name, "Drehung nach rechts", 90)` dreht den Turm nach rechts.

Die Funktion `EEPSetTrainHook(name, false)` schaltet den Haken aus.

Alle genannten Funktionen benutzen die Variable `name` als erstes Argument um den Zug anzusprechen, der die Aktion per Kontaktpunkt ausgelöst hat.

Die Zeile `TimeCounter = 0` setzt den Zähler zurück auf Null, damit die Zeitmessung für die folgenden Aktionen, welche in der `EEPMain()` ausgelöst werden, erst jetzt beginnt.

Die Zeile `bCraneIsReady = true` sorgt dafür, dass die `EEPMain()`-Funktion jetzt die Erlaubnis hat den Kran zu bedienen.

Bedenken Sie bitte, dass alle Aktionen in der Funktion `TakeGoodsStart()` sofort passieren. Der Zähler wird also in dem Augenblick zurückgesetzt, wenn der Zug den Kontaktpunkt überfährt und nicht erst, wenn die Aktionen "Zug stoppen, Ausleger heben und Drehen, Haken ausschalten" abgeschlossen sind! Das gleiche gilt für `bCraneIsReady`.

Die nächsten Aktionen müssen in einer zeitlichen Abfolge durchgeführt werden und passieren deshalb in der Funktion `EEPMain()`.

```
function EEPMain()
    TimeCounter = TimeCounter+1
    --print(TimeCounter)

    if bCraneIsReady then

        if TimeCounter == 30 then
        if TimeCounter == 40 then
            TakeGoods(TrainName)
        end

        if TimeCounter == 60 then
            print("-- Aufnehmen von Ladegut")
            EEPSetTrainHook(TrainName, true)
            print("-- Kran aufwärts")
            EEPSetTrainAxis(TrainName, "Ausleger heben/senken",
                100)
        end
    end
```

```
if TimeCounter == 80 then
    print("-- Links drehen")
    print("-- Rechts drehen")
    EEPSetTrainAxis(TrainName, "Drehung nach rechts", 70)
end

if TimeCounter == 100 then
    print("-- Ladegut ablegen")
    EEPSetTrainHook(TrainName, false)
end

if TimeCounter == 110 then
    print("-- Kran dreh in Ausgangsposition")
    EEPSetTrainAxis(name, "Haupthaken senken/heben", 80)
    EEPSetTrainAxis(TrainName, "Ausleger heben/senken", 0)
    EEPSetTrainAxis(TrainName, "Drehung nach rechts", 100)
end
end

return 1
end
```

Die Verzweigung `if bCraneIsReady then` prüft, ob der Zug bereit ist. Die Variable `bCraneIsReady` wurde mit `false` initialisiert. Deshalb tut die Funktion `EEPMain()` nichts, bis der Zug den Kontaktpunkt überfahren und mit einem Funktionsaufruf den Wert dieser Variablen auf `true` gesetzt hat.

Die folgenden fünf `if`-Verzweigungen prüfen, ob die `EEPMain()` 30, 60, 80, 100 oder 110 mal durchlaufen wurde. Dieser Zähler wird vom Zug zu Beginn des Szenarios auf Null zurückgesetzt, wenn er den Kontaktpunkt überfährt.

Nach ~~30~~ 40 Durchgängen ruft `EEPMain()` die Funktion `TakeGoods()` auf und übergibt ihr den in `TrainName` gespeicherten Zugnamen.

```
function TakeGoods(name)
    hResult = EEPSetTrainAxis(name, "Ausleger heben/senken", 10)
    print("-- Kran runter: ", hResult)
end
```

Diese Funktion enthält die Funktion `EEPSetTrainAxis()`, welche den Ausleger absenkt.

Nach 60 Durchläufen wird der Haken aktiviert

```
EEPSetTrainHook(TrainName, true)
```

und der Ausleger wieder angehoben

```
EEPSetTrainAxis(TrainName, "Ausleger heben/senken", 100)
```

Nach 80 Durchläufen wird der Turm weiter gedreht

```
EEPSetTrainAxis(TrainName, "Drehung nach rechts", 70)
```

Und nach 100 Durchläufen wird der Haken ausgeschaltet

```
EEPSetTrainHook(TrainName, false)
```

Zum Schluss wird der Ausleger nach 110 Durchläufen wieder in die Ausgangslage zurück gebracht

```
EEPSetTrainAxis(name, "Haupthaken senken/heben", 80)  
EEPSetTrainAxis(TrainName, "Ausleger heben/senken", 0)  
EEPSetTrainAxis(TrainName, "Drehung nach rechts", 100)
```

und das Szenario ist beendet.

Zur Erinnerung: Die Funktion `EEPMain()` wird fünfmal je Sekunde aufgerufen. Das bedeutet, dass zum Beispiel nach 30 Durchläufen 6 Sekunden vergangen sind.

## Tutorial\_50\_LUA Zug-Routen ermitteln und zuweisen



Bild 50\_1

Dieses Tutorial demonstriert, wie man die eingestellte Route eines Zuges ermitteln und dem Zug eine neue Route zuweisen kann..

- [EEPGetTrainRoute\(\)](#)
- [EEPSetTrainRoute\(\)](#)

Der Zug wechselt in jeder Runde, die er dreht die Route. Er fährt abwechselnd nach Bergheim und nach Hochspeyer.

Zu Beginn werden die Variablen `hResult` und `Route` initialisiert. `hResult` wird dazu dienen, die Erfolgsmeldung von Funktionen zu speichern. `Route` wird benutzt, um den Namen von Routen zu speichern.

```
hResult = 0  
Route = ""
```

Die Funktion `EEPMain()` hat in diesem Tutorial keine Aufgaben und ihr wiederholter Aufruf wird durch Rückgabe der Zahl 0 abgestellt.

```
function EEPMain()  
    return 0  
end
```

Ein Kontaktpunkt im Gleis ruft die Funktion `ChangeRoute()` auf. Funktionsaufrufe aus Kontaktpunkten geben seit Einführung des Plug-In 2 zu EEP 11 immer den Namen des auslösenden Zuges mit. Im Beispiel nimmt die Variable `name` diesen Namen auf.

Wenn Sie das Eigenschaftsfenster des Kontaktpunktes öffnen, dann werden Sie dort nur den Funktionsnamen im Feld "Lua Funktion" finden, aber keine Klammern dahinter

und keinen Parameter. Den "Zugnamen" ergänzt EEP selbständig. In den Kontaktpunkten darf wie zuvor nur der Name der Funktion stehen!

```
function ChangeRoute(name)
    hResult, Route = EEPGetTrainRoute(name)
    print("Current route: ", Route, "(", hResult, ")")
    if hResult then
        if Route == "To Bergheim" then
            print("Set route to Hochspeyer")
            hResult = EEPSetTrainRoute(name, "To Hochspeyer")
        else
            print("Set route to Bergheim")
            hResult = EEPSetTrainRoute(name, "To Bergheim")
        end
    end
end
```

Zu Beginn der Funktion wird mit `EEPGetTrainRoute()` der Name der eingestellten Route ermittelt.

Dann wird mit `if hResult then` geprüft, ob der Name ermittelt werden konnte. `hResult` kann entweder `true` oder `false` sein. Deshalb muss man bei der `if`-Verzweigung keinen Vergleich anstellen, sondern kann den Wert der Variablen für die Verzweigung nutzen.

Falls `hResult` den Wert `true` liefert wird als nächstes geprüft, ob der ermittelte Name der Route "To Bergheim" lautet:

```
if Route == "To Bergheim" then
```

Wenn ja, dann wird dem Zug die Route "To Hochspeyer" zugewiesen.

```
hResult = EEPSetTrainRoute(name, "To Hochspeyer")
```

Ansonsten wird ihm die Route "To Bergheim" zugewiesen.

```
hResult = EEPSetTrainRoute(name, "To Bergheim")
```

In beiden Fällen wird der Rückgabewert der Funktionen in der Variablen `hResult` aufgefangen, aber nicht weiter verwendet.

Beachten Sie bitte, dass Routennamen Strings, also Textbausteine sind. Sie müssen entsprechend durch Anführungszeichen als String gekennzeichnet werden.

## Tutorial\_51\_LUA Zugteil abkuppeln, Zugkupplungen schalten



Bild 51\_1

Dieses Tutorial demonstriert, wie ein Teil des Zuges abgekoppelt und wie die Kupplungen eines Zuges umgestellt werden können.

- [EEPTrainLooseCoupling\(\)](#)
- [EEPSetTrainCouplingRear\(\)](#)
- [EEPSetTrainCouplingFront\(\)](#) -- Im Tutorial nicht enthalten

Der Zug koppelt am oberen Ende des Kreises den letzten Wagen ab. Ein Stück weiter kommt er dann zum Stehen, wird vom Wagen eingeholt und setzt die Runde fort sobald der Wagen wieder angekoppelt ist.

Die Funktion `EEPMain()` hat in diesem Tutorial keine Aufgaben und ihr wiederholter Aufruf wird durch Rückgabe der Zahl 0 abgestellt.

```
function EEPMain()  
    return 0  
end
```

Zwei Kontaktpunkte steuern die Ereignisse auf der Anlage. Der erste ruft die Funktion `Unconnect()` [auf Deutsch: Verbindung trennen]

Funktionsaufrufe aus Kontaktpunkten geben seit Einführung des Plug-In 2 zu EEP 11 immer den Namen des auslösenden Zuges mit. Im Beispiel nimmt die Variable `name` diesen Namen auf.

Wenn Sie das Eigenschaftsfenster des Kontaktpunktes öffnen, dann werden Sie dort nur den Funktionsnamen im Feld "Lua Funktion" finden, aber keine Klammern dahinter und keinen Parameter. Den "Zugnamen" ergänzt EEP selbständig. In den Kontaktpunkten darf wie zuvor nur der Name der Funktion stehen!

```
function Unconnect(name)
    print("Train: ", name)
    hResult = EEPTrainLooseCoupling(name, true, 3)
    if hResult then
        print("unconnected")
    end
end
```

Die Variable `name` nimmt den Namen des auslösenden Zuges auf.

Dann wird der Zugteil hinter dem dritten Rollmaterial abgekoppelt.

```
hResult = EEPTrainLooseCoupling(name, true, 3)
```

Der Parameter `true` an zweiter Stelle bestimmt, dass von vorne gezählt wird. Stünde er auf `false`, dann würde von hinten gezählt.

Der vordere Zugteil überfährt ein wenig weiter den zweiten Kontaktpunkt. Dieser ruft eine Funktion `Stop()` auf:

```
function Stop(name)
    EEPSetTrainSpeed(name, 0)
    EEPSetTrainCouplingRear(name, true)
end
```

Die erste Zeile dieser Funktion hält den Zug an.

Die zweite Zeile stellt die hintere Kupplung wieder auf "Ankoppeln" zurück. Sie wurde beim Abkoppeln des Waggons auf "Abstoßen" gesetzt.

Da der letzte Waggon in voller Fahrt abgekoppelt wurde, hat er genügend Schwung, um kurz darauf den Zug einzuholen und sich wieder anzukoppeln. Die eingestellte Sollgeschwindigkeit des Wagens beträgt noch immer 50 km/h. Und diese Geschwindigkeit überträgt er beim Ankoppeln an den Zug; denn beim Ankoppeln ist der Zugteil, welcher in Bewegung ist, in EEP immer das bestimmende Element und definiert den "Zugnamen" und die Geschwindigkeit des neu gebildeten Fahrzeugverbands. Daher fährt der Zug wieder mit 50 km/h weiter und der Zyklus beginnt von neuem.

## Tutorial\_53\_LUA

## Gleis besetzt, Kameras aufrufen, Anlage laden

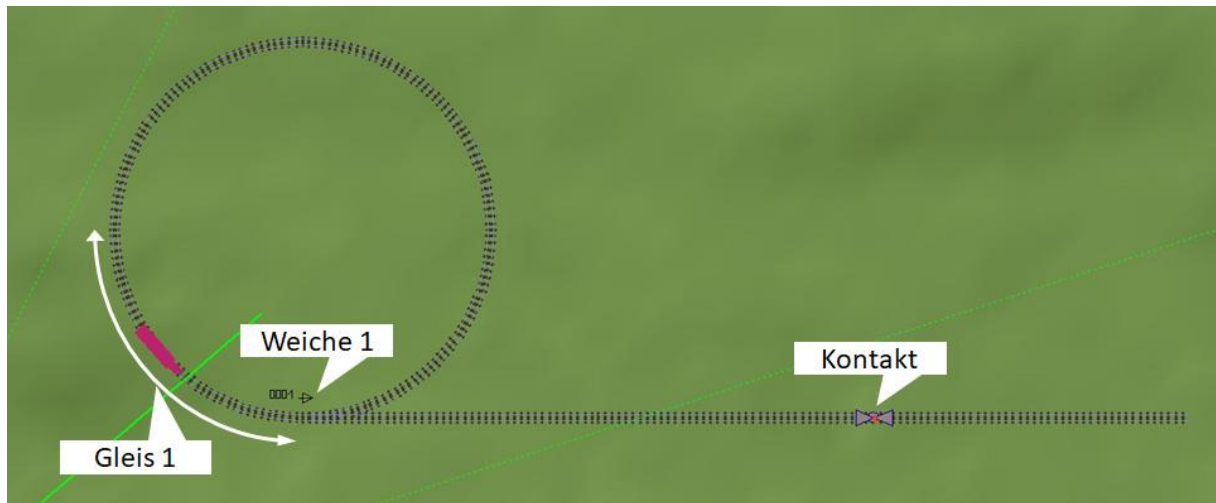


Bild 53\_1

Dieses Tutorial demonstriert, wie man über eine Gleisbesetzt-Meldung Kameras einstellen kann und wie man aus einer Anlage heraus eine andere starten kann.

- [EEPRegisterRailTrack\(\)](#)
- [EEPIsRailTrackReserved\(\)](#)
- [EEPSetCamera\(\)](#)
- [EEPSetPerspectiveCamera\(\)](#)
- [EEPLoadProject\(\)](#)

Zunächst fährt die Lok im Kreis. Immer wenn die Lok Gleis 1 besetzt wird eine der in EEP enthaltenen Verfolgerkameras nach dem Zufallsprinzip eingeschaltet. Sobald die Lok Gleis 1 verlässt, Gleis 1 also frei ist, schaltet EEP auf die vorinstallierte "Camera 1". Wenn die Lok zum 3. Mal Gleis 1 besetzt, wird Weiche 1 umgeschaltet und die Lok verlässt den Kreis. Beim Überfahren des Kontaktes wird die Anlage [Tutorial 54 LUA](#) geladen.

Im Skript werden zu Beginn einige Variablen initialisiert. Was die Variable `Schalter` macht, darauf wird später detailliert eingegangen. Beachten Sie zunächst nur, dass er auf 0 gesetzt wird. Die Variable `SchleifenZaehler` erklärt sich von selbst und im Prinzip auch `AnzahlSchleifen`. Wobei "Schleifen" nicht ganz korrekt ist, denn eigentlich werden die "Besetzt"-Meldungen gezählt bzw. deren max. Anzahl vorgegeben.

```
Schalter = 0
clearlog()
SchleifenZaehler = 0
AnzahlSchleifen = 3
print("Hey let's start, EEP Version is: ", EEPVer)
```

Damit später die Funktion `EEPIsRailTrackReserved(Gleis-ID)` melden kann, ob ein bestimmtes Gleis besetzt ist oder nicht, muss es vorher registriert werden.

```
EEPRegisterRailTrack(1)
```

Die laufende Überprüfung, ob ein Gleis besetzt ist oder nicht, erfolgt in der `EEPMain()` und damit 5 Mal pro Sekunde. Um aber nicht alle 0,2 Sekunden, die bei "besetzt" auszuführenden Aktionen zu tätigen, benötigt man den `Schalter` Dieser war ja mit 0 vorbelegt. Deshalb setzt man ihn als Erstes auf 1.

```
function EEPMain()  
    hr, bIstBesetzt = EEPIsRailTrackReserved(1)  
    if bIstBesetzt then  
        if Schalter == 0 then  
            Schalter = 1
```

Wenn Gleis 1 besetzt ist, wird der `SchleifenZaehler` um 1 erhöht. Anschließend wird überprüft, ob der `SchleifenZaehler` gleich der `AnzahlSchleifen` ist. Wenn das der Fall ist, wird mit `EEPSetSwitch()` die Weiche 1 umgelegt, so dass die Lok den Kreis verlässt. Doch noch ist es nicht so weit, darum wird zunächst im Ereignisfenster die Meldung ausgegeben, dass das Gleis besetzt ist.

```
        SchleifenZaehler = SchleifenZaehler + 1  
        if SchleifenZaehler == AnzahlSchleifen then  
            EEPSetSwitch(1, 2)  
        end  
        print("Gleis 4 besetzt: ", bIstBesetzt)
```

*Nun fällt Ihnen sicherlich auf, dass dem Autor hier ein Tippfehler unterlaufen ist, denn nicht Gleis 4 sondern Gleis 1 wurde auf "besetzt" überprüft. Bitte ändern Sie im Skript die (hier rot markierte) 4 in eine 1.*

Laut Aufgabe soll bei besetztem Gleis auf eine zufällig ausgewählte Verfolgerkamera umgeschaltet werden. In EEP sind 9 bzw. ab EEP 16 Plug-in 4 10 Verfolgerkameran vorprogrammiert. Eine Zufallszahl ermittelt man mit der allgemeinen Lua Funktion `math.random(ersteZahl, letzteZahl)`. Hierbei hat sich der Autor auf die Kameras 1 bis 6 beschränkt. Eingeschaltet wird die Verfolgerkamera mit der EEP-spezifischen Funktion `EEPSetPerspectiveCamera()`. Als ersten Parameter wird dieser Funktion die Nummer der Kamera – also `aktuelle_Kamera` – übergeben und als zweiten Parameter der Lua-Name des Fahrzeugverbandes – hier der Lok. (Ab EEP 15 kann der 2. Parameter weggelassen werden. Dann wird die Mitfahrkamera auf den aktiven Fahrzeugverband ausgerichtet.)

```
        -- zufaellige Verfolgerkamera --  
        aktuelle_Kamera = math.random(1, 6)  
        hr = EEPSetPerspectiveCamera(aktuelle_Kamera,  
                                     "#DB 113-270 TEE SK2")  
        print("Verfolgerperspektive Nr: ", aktuelle_Kamera,  
              " (" , hr, ")")  
    end
```

Irgendwann verlässt nun die Lok Gleis 1 und im folgenden Durchlauf der `EEPMain()` trifft `if bIstBesetzt` nicht mehr zu. Im `else (sonst)` Bereich der Verzweigung, wird definiert, was dann passieren soll.

```
else
    if Schalter == 1 then
        Schalter = 0
        print("Gleis 4 frei")
        hr = EEPSetCamera(0, "Camera 1")
        print("Statische Kamera: Camera 1 (", hr, ")")
    end
end
end
```

Schalter wurde ja beim ersten "Besetzt"-Durchlauf auf 1 gesetzt. D.h. beim ersten "Nicht besetzt"-Durchlauf ist die Bedingung `if Schalter == 1` noch gültig. Damit anschließend dieser Bereich nicht mehr durchlaufen wird (bis Gleis 1 wieder besetzt ist) wird als Erstes `Schalter` wieder zurück auf 0 gesetzt. Es folgt die Ausgabe im Ereignisfenster, dass das Gleis frei ist.

*Auch in diesem print-Befehl hat der Tippfehlerteufel wieder zugeschlagen.  
Bitte ändern Sie im Skript die (hier rot markierte) 4 in eine 1.*

Wenn Gleis 1 nicht besetzt ist, soll die vom Autor eingerichtete Kamera "Camera 1" eingeschaltet sein. Die Umschaltung erfolgt über die EEP-spezifische Funktion `. Der 1. Parameter 0` gibt an die Funktion weiter, dass es sich um eine "statische Kamera" handelt. Abschließend wird im Ereignisfenster über die Variable `hr` (Anmerkung s.o.) ausgegeben, ob der Kameraaufruf erfolgreich war.

Verbleibt nur noch mit dem Rückgabewert 1 dafür zu sorgen, dass die `EEPMain()` alle 0,2 Sekunden aufgerufen wird und sie zu beenden.

```
    return 1
end
```

Wenn die Lok zum dritten Mal Gleis 1 besetzt (wobei zu beachten ist, dass sie zu Beginn des Tutorials bereits auf Gleis 1 steht), wird die Weiche umgestellt und die Lok fährt über die lange Gerade weiter. Hierbei überfährt sie einen Kontakt, in dem die Funktion `LoadNextANL()` aufgerufen wird.

```
-- andere Anlage laden --
function LoadNextANL()
    hr = EEPLoadProject("Tutorials\\Tutorial_54_LUA.anl3");
    print("Lade andere Anlage (", hr, ")")
end
```

Diese Funktion hat nur die Aufgabe, die Anlage zu schließen und mit [EEPLoadProject\(\)](#) die Anlage [Tutorial 54 LUA](#) im Tutorials-Ordner zu laden und den Erfolg bzw. Misserfolg im Ereignisfenster anzuzeigen.

## Tutorial\_54\_LUA      Anlage laden 2

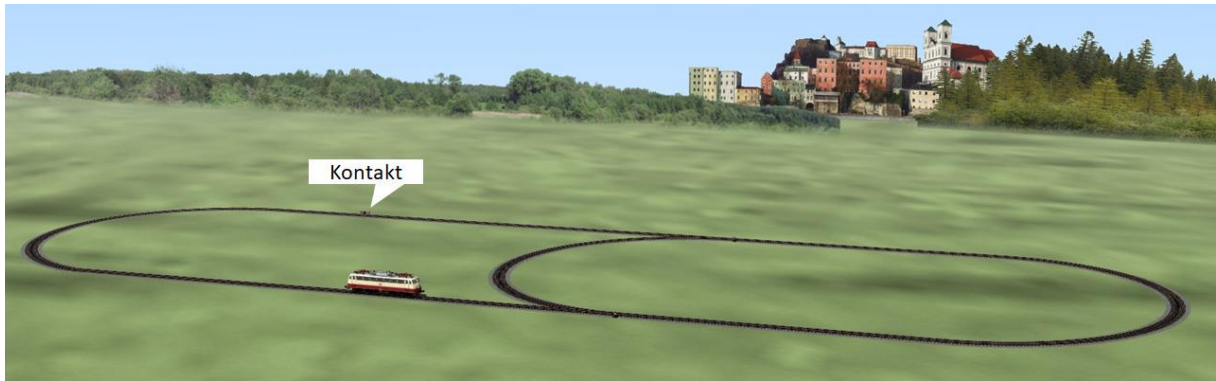


Bild 54\_1

Dieses Tutorial arbeitet im Wechsel mit dem [Tutorial 53 LUA](#).

Die Lok fährt über den rechten Teil des äußeren Halbkreises und löst gegen Ende der Gegengeraden einen Kontakt aus, der die Funktion `LoadNextANL()` aufruft. Diese Funktion hat nur die Aufgabe die EEP-spezifische Lua-Funktion

- [EEPLoadProject\(\)](#)

aufzurufen, die wiederum diese Anlage schließt und die vorherige Tutorial-Anlage öffnet.

Der Pfad der zu öffnenden Anlage wird der Funktion als Parameter übergeben. Der Pfad muss sich dabei auf den Ordner "Ressourcen\Anlagen" oder dem unter *Programmeinstellungen* eingetragenen Anlagenpfad beziehen.

Da in den in den Kontaktpunkten eingetragenen Lua-Funktionen keine Parameter aufgeführt sein dürfen (außer dem "Zugnamen" seit EEP 11.2 Plug-in 2 und der "Gleis\_ID" seit EEP 16.3 Plug-in 3), muss die Funktion `EEPLoadProject("Anlagenpfad")` in eine andere Funktion ohne Parameter – hier `LoadNextANL` - eingebunden werden.

```
clearlog()

print("Hey let's start, EEP Version is: ", EEPVer)

function EEPMain()
    return 0
end

function LoadNextANL()
    EEPLoadProject("Tutorials\\Tutorial_53_LUA.anl3");
end
```

In diesem Tutorial spielt die Funktion [EEPMain\(\)](#) keine Rolle. Aus diesem Grund wurde ihr Rückgabewert auf 0 gesetzt. Dadurch wird die Funktion nicht erneut aufgerufen. Der Funktionsaufruf `LoadNextANL()` funktioniert aber weiterhin.

## Tutorial\_55\_Gleisbesetzt



Bild 55\_1

Dieses Tutorial demonstriert die Verwendung der Gleisbesetzt-Funktionen für Gleise.

- [EEPRegisterRailTrack\(\)](#)
- [EEPisRailTrackReserved\(\)](#)

Da diese Funktionen erst im Plug-in 3 zu EEP 11.3 bereitgestellt wurden, ist dies auch die Mindestvoraussetzung für diese Tutorial-Anlage. Zunächst wird daher mit der EEP-spezifischen Variablen [EEPVer](#) geprüft, ob mindestens EEP 11.3 installiert ist. Falls nicht wird eine Warnmeldung ausgegeben.

```
if EEPVer < 11.3 then    -- prüfen, ob EEP-Version geeignet ist
    print("Dieses Tutorial erfordert mindestens EEP 11 mit
                                     Update 3 und Plugin 3!")
else
    print("Dieses Tutorial zeigt die Verwendung der neuen
          \"Gleis besetzt\" Lua-Funktionen")
end
```

Leider kann durch Lua aber nicht direkt überprüft werden, ob auch das Plug-In 3 zu EEP 11.3 installiert ist. Die Funktion `EEPRegisterRailTrack()` gibt aber als Rückgabewert `true` zurück, wenn die Registrierung eines Gleises erfolgreich war bzw. `false`, wenn nicht. Gleise zu registrieren geht aber nur, wenn das Plug-in 3 installiert ist. Dies wird später im Skript ausgenutzt, um den wiederholten Aufruf der `EEPMain()`-Funktion zu stoppen, wenn die entsprechenden Gleise nicht registriert werden konnten.

Voreingestellt wird der Rückgabewert der `EEPMain()` auf "wiederholen", d.h. 1.

```
MainWiederholen = 1  -- Variable steuert den wiederholten
                        Aufruf der EEPMain Funktion
```

In der Tutorialanlage versperrt der Kühlwagen dem Güterzug den Weg (siehe Bild 55\_1). Die beiden Gleise zwischen den Weichenlaternen werden bei jedem Durchlauf der `EEPMain()` Funktion geprüft, ob sie durch Rollmaterial besetzt sind.

Wenn ja, dann bleibt das Signal für den Güterzug auf "Halt". Räumt man den Kühlwagen aus dem Weg, dann erkennt Lua, dass die Gleise frei sind und schaltet das Signal für den Güterzug auf "Fahrt".

Um den Weg freizugeben kann man entweder den großen Trafo rechts bedienen. Dann kommt eine Rangierlok aus dem Schuppen und räumt das Hindernis weg. Oder man schiebt den Waggon mit gedrückter [Strg]-Taste auf das Stumpfgleis. Oder man löscht ihn komplett.

Die beiden Gleise müssen registriert werden, bevor geprüft werden kann, ob sie besetzt sind.

```
Gleis1Reg = EEPRegisterRailTrack(20)  -- vorderes Gleis
                                           zwischen den beiden Weichen registrieren
Gleis2Reg = EEPRegisterRailTrack(2)   -- hinteres Gleis
                                           zwischen den beiden Weichen registrieren
```

Die Variablen `Gleis1Reg` und `Gleis2Reg` dienen der anschließenden Prüfung, ob Gleise, die registriert werden sollten, tatsächlich existieren und erfolgreich registriert sind. Zur einfacheren Weiterbehandlung werden `Gleis1Reg` und `Gleis2Reg` so in der Variablen `RegOkay` zusammengefasst, dass nur wenn beide `true` sind, auch `RegOkay true` ist.

```
RegOkay = Gleis1Reg and Gleis2Reg
if RegOkay then
    print("Beide Gleise wurden registriert")
else
    print("Mindestens ein Gleis wurde nicht gefunden!")
end
```

```
function EEPMain()
    if RegOkay then                -- nur ausführen, wenn beide Gleise
                                    registriert sind
        Signal5Automatik()         -- Aufruf der Unterfunktion für die
                                    Signalsteuerung
    else                            -- wenn Registrierung der Gleise
                                    nicht hResultreich war
        MainWiederholen = 0        -- Aufruf der EEPMain Funktion
                                    beenden
        print("Signal 5 wird nicht von Lua gesteuert!")
        print("EEPMain Funktion wird nicht mehr aufgerufen")
    end
    return MainWiederholen
end
```

Nur wenn die Registrierung der Gleise erfolgreich war, wird bei jedem Durchlauf der `EEPMain()` die Unterfunktion `Signal5Automatik()` zur Signalsteuerung aufgerufen.

Andernfalls (`else`) wird der Rückgabewert `MainWiederholen` der `EEPMain()` auf 0 gesetzt, wodurch die `EEPMain()` kein 2. Mal aufgerufen wird. Damit ist sichergestellt, dass das Tutorial nur mit installiertem Plug-in 3 zu EEP 11.3 funktioniert.

Die eigentliche Besetzt-Prüfung findet in der Funktion `Signal5Automatik()` statt.

```
function Signal5Automatik()
    hResult1,Besetzt1 = EEPIsRailTrackReserved(20)
    hResult2,Besetzt2 = EEPIsRailTrackReserved(2)
    if hResult1 and hResult2 then
        if Besetzt1 or Besetzt2 then
            EEPSetSignal(5,2)        -- Signal 5 auf "Halt" stellen
        else
            EEPSetSignal(5,1)        -- Signal 5 auf Fahrt stellen
        end
    else
        print("Prüfung fehlgeschlagen!")
    end
end
```

Da die Gleisbesetzt-Funktion `EEPIsRailTrackReserved()` zwei Werte zurück liefert, werden diese zunächst zwischengespeichert:

Der erste Rückgabewert besagt, ob das angesprochene Gleis gefunden wurde, also existiert und registriert ist.

Der zweite Rückgabewert besagt, ob Rollmaterial auf dem Gleis steht (`true`) oder nicht (`false`).

In der äußeren `if`-Verzweigung wird der Fall aufgefangen, dass die Gleisbesetztmeldung nicht ermittelt werden konnte. Das wäre dann der Fall, wenn das abgefragte Gleis entweder nicht existiert oder nicht registriert wurde.

Entscheidend ist die innere `if`-Verzweigung. Wenn mindestens eines der beiden Gleise besetzt ist, d.h. `true` zurück melden, bleibt das Signal 5 auf "Halt" bzw. wird auf "Halt" gesetzt. Nur, wenn beide Gleise `false`, also "nicht besetzt" zurück melden, wird die Fahrt frei gegeben.

Nachdem der Güterzug die 2. Weiche passiert hat, fährt die Rangierlokomotive den Kühlwagen wieder zwischen die Weichen, kuppelt ihn dort ab und fährt zurück in den Lokschuppen. Die komplette Fahrt der Rangierlokomotive wird nicht mit Lua sondern über Kontaktpunkte und unsichtbare Signale gesteuert.