Lua - Beschreibung und Handbuch

Inhaltsverzeichnis:

Allgemeine Informationen

Mit dem Plug-In 2 zu „Eisenbahn-X“ wurde eine Neuerung eingeführt, welche die Steuerungsmöglichkeiten in EEP in neue Höhen hebt. Es handelt sich um eine Skriptsprache, also eine Programmiersprache, die während der Ausführung des Programms in Computerbefehle übersetzt wird. Der Name dieser Skriptsprache ist "Lua" und sie wurde von der PUC Rio (Pontifical Catholic University of Rio de Janeiro) entwickelt. Weitere Hintergrundinformationen dazu können Sie unter „http://de.wikipedia.org/wiki/Lua“ nachlesen.

Die in EEP implementierte Version ist 5.2 und sie enthält zusätzlich eine Anzahl spezifischer Funktionen, die eine Verbindung zur EEP Anlage herstellen und diese beeinflussen können.

Die Sammlung EEP-spezifischer Funktionen wird ständig erweitert. In Eisenbahn X (EEP 10) mit Plugin 2 ließen sich Weichen und Signale durch Lua steuern. Mit EEP 11.0 kamen Zuggeschwindigkeit, Kupplungskontrolle bei Rollmaterial und Datenslots für die permanente Speicherung von Werten hinzu. Plugin 1 für EEP 11 enthielt Funktionen für die Steuerung von Immobilien. Mit Plugin 2 wurde der Funktionsumfang erneut vergrößert. Kontaktpunkte übermitteln jetzt bei jedem Funktionsaufruf den Namen des auslösenden Zugverbands. Weiterhin ist es nun möglich Rauch, Licht und Warnton eines Zuges, sowie die Kupplungen, Achsen und - wenn vorhanden - den Haken eines Zuges zu beeinflussen. Und EEP 13 bietet zusätzlich die Manipulation von Tipp-Texten.

Auf den folgenden Seiten finden Sie eine Erläuterung des Skript Editors sowie Beschreibungen der EEP-spezifischen Funktionen, die mit Lua implementiert wurden. Die Tutorial Anlagen mit funktionierenden Lua Skripten werden ebenfalls erläutert. Der Befehlssatz von Lua selbst ist umfangreich und eine komplette Erklärung würde den Rahmen dieser Anleitung sprengen. Eine deutsche Dokumentation, inklusive der deutschen Übersetzung des originalen Reference Manuals, finden Sie auf der folgenden Website:

http://lua.coders-online.net

Wir möchten Ihnen nahe legen diese oder andere Websites aufzusuchen und sich dort mit den Grundlagen dieser Skriptsprache vertraut zu machen.

Einbettung in EEP 13

In EEP wurden Schnittstellen geschaffen, über die es mit Lua Daten austauschen kann. Das bedeutet, dass EEP Informationen an Lua liefern und Befehle von Lua empfangen kann. Für diesen Zweck haben wir spezielle Lua Funktionen geschaffen. Sie beginnen alle mit den Buchstaben EEP.

EEP enthält nun ein Ereignisfenster für die Ausgabe von Meldungen (Bild 1) und ein Skriptfenster für die Erstellung von Programmen (Bild 2).

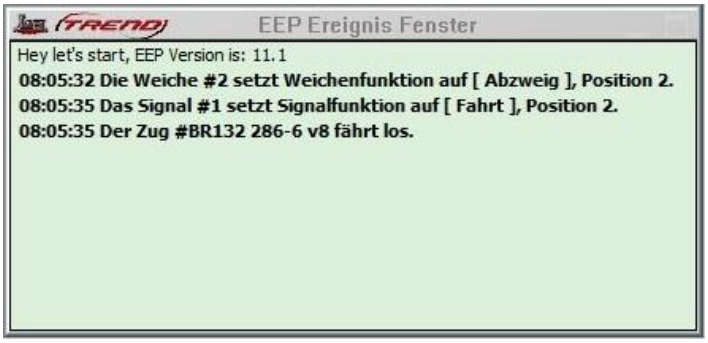

Bild 1

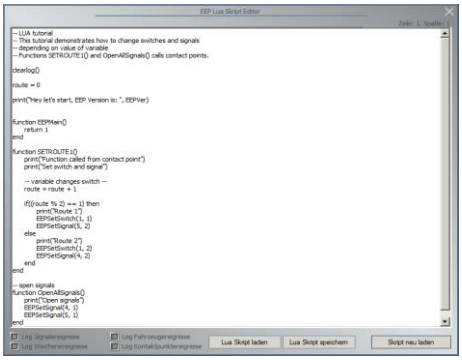

Bild 2

Das Ereignisfenster können Sie in den Programmeigenschaften aktivieren. (Bild 3)

Das Skriptfenster öffnen Sie mit dem Knopf in der Menüleiste. (Bild 4)

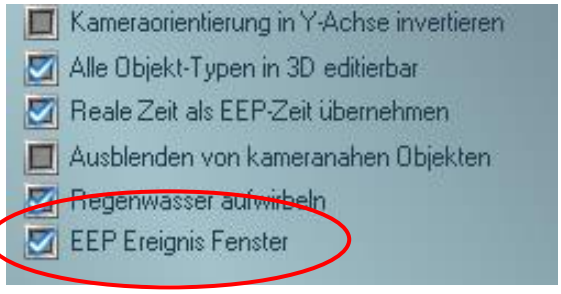

Bild 3

Bild 4

Das Ereignisfenster kann sowohl Systemmeldungen als auch selbst erstellte Texte ausgeben.

Zu den Systemmeldungen gehören Statusmeldungen von Weichen, Signalen, Zügen und Kontaktpunkten sowie Fehlermeldungen des Lua Interpreters.

Für eigene Textausgaben verwenden Sie bitte die Funktion `print()`.

Statusmeldungen können im Skriptfenster ein- und ausgeschaltet werden (Bild 5).

Bild 5

Das Skriptfenster ist der Ort, an dem ein Skript erstellt wird. Außerdem enthält es Knöpfe zum Speichern und Laden eines Skripts sowie zur Aktivierung des aktuellen Skripts. Ein erstelltes Skript wird automatisch gemeinsam mit der Anlage gespeichert und geladen. Die Knöpfe "Lua Skript laden" und "Lua Skript speichern" (Bild 6) sind nur für den Fall gedacht, dass man ein Skript unabhängig - zum Beispiel als Backup - speichern beziehungsweise ein extern erstelltes Skript laden möchte.

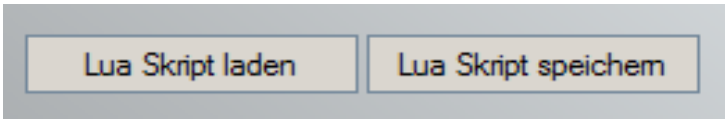

Bild 6

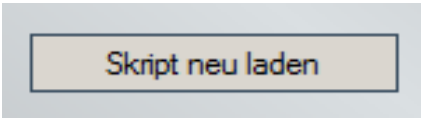

Bild 7

Wichtig: Ein erstelltes Skript muss an EEP übertragen werden.

Für diesen Zweck drücken Sie bitte stets den Knopf "Skript neu laden" (Bild 7) nachdem Sie ein Skript erstellt oder Änderungen daran vorgenommen haben! Ansonsten sind ihr Skript oder Ihre Änderungen daran verloren wenn Sie das Skriptfenster schließen.

Für erste Erkundungen in der neuen Welt einer Skript-gesteuerten Anlage finden Sie im Anlagenordner unter den Tutorials fünfzehn kleine Beispiele funktionierender Anlagen, die jeweils nur wenige Funktionen in schlichtem Umfeld demonstrieren. Damit haben Sie die Möglichkeit genau zu verfolgen, was die einzelnen Elemente eines Skripts bewirken. Jede dieser Anlagen wird im Kapitel Mitgelieferte Beispiele genauer beschrieben.

1. Tutorial_33_LUA_1 Signale und Weichen, Funktionsaufruf per KP
2. Tutorial_34_LUA_2 Callback Funktionen 1
3. Tutorial_35_LUA_3 Callback Funktionen 2
4. Tutorial_36_LUA_4 Tabellen und Zufallsgenerator
5. Tutorial_37_LUA_5 EEPMain(), Geschwindigkeit, Kuppeln
6. Tutorial_38_LUA_6 Achsenslots
7. Tutorial_39_LUA_7 Rollmaterial-Achsen
8. Tutorial_40_LUA_8 Data Slots
9. Tutorial_44_LUA_1 Immobilien Rauch, Licht, Feuer
10. Tutorial_45_LUA_2 Immobilien Achsen-Animation
11. Tutorial_46_LUA_3 Immobilien Achsen setzen, Immobilien positionieren
12. Tutorial_48_LUA Zug Rauch, Licht, Warnton, Zugnamen im Funktionsaufruf per KP
13. Tutorial_49_LUA Kranzug Achsen und Haken
14. Tutorial_50_LUA Zug-Route ermitteln und zuweisen
15. Tutorial_51_LUA Zugteil abkuppeln, Zugkupplungen schalten

Syntax von Lua

Programmiersprachen sind Texte, die in Computerbefehle übersetzt werden. Bei Skriptsprachen übernimmt ein Interpreter diese Aufgabe. Der Interpreter erfordert eine bestimmte Schreibweise um erkennen zu können, was das Programm tun soll. Und diese Schreibweise nennt man die Syntax.

Wie jede Programmiersprache so erfordert auch Lua absolute Genauigkeit in der Syntax. Ein einzelnes fehlendes oder falsches Zeichen führt schon zur fehlerhaften Ausführung oder gar zum Abbruch. Selbst Groß- und Kleinschreibung wird unterschieden.

Das folgende Beispiel soll Ihnen verdeutlichen, wie präzise Sie beim Entwurf eines Skripts sein müssen:

Mit zwei aufeinander folgenden Bindestrichen wird in Lua ein Kommentar eingeleitet.

```
-- Dies ist ein Kommentar
```

Das ist ein Text, der vom Interpreter ignoriert wird. Solche Kommentare sind nützliche Gedächtnisstützen und können auch anderen Usern, die ein Skript lesen, wichtige Informationen liefern.

Lässt man versehentlich einen der zwei Bindestriche weg, dann bekommt man bei der Übertragung des Skripts die folgende Fehlermeldung (Bild 8):

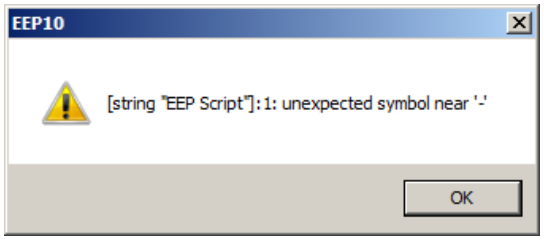

Bild 8

Die Meldung enthält eine Reihe nützlicher Informationen.

- [string "EEP Script"] sagt, in welcher Datei der Fehler auftrat.
- :1: ist die Zeilennummer, an der der Fehler auftrat.
- unexpected symbol near '-' (übersetzt: unerwartetes Zeichen in der Nähe von '-') besagt, dass der Interpreter vor oder nach dem Bindestrich etwas anderes vorgefunden hat als erwartet.

Bitte beachten Sie, dass der Interpreter nicht meldet *"Da fehlt der zweite Bindestrich!"* Denn zur Syntax von Lua gehört, dass ein einzelner Bindestrich als "Minus" angesehen wird. Bei der Übersetzung des Skripts versucht der Interpreter das, was hinter dem Strich steht von dem, was davor steht zu subtrahieren. Entsprechend erwartet er sowohl vor als auch hinter dem Minuszeichen eine Zahl. Was er jedoch in unserem Beispiel vorfindet ist etwas anderes und somit unerwartet. Ein Interpreter kann nicht "verstehen" was der Autor des Skripts erreichen möchte, sondern nur stur abarbeiten, was im Skript steht.

Deshalb soll dieses Handbuch Ihnen die Syntax von Lua vermitteln. Wir werden Ihnen anhand von Beispielen zeigen, wie Sie ein Lua Skript selbst erstellen können. Schritt für Schritt machen wir Sie mit der Syntax von Lua vertraut.

Ein weiteres Besipiel soll Ihnen die Wichtigkeit von Groß- und Kleinschreibung zeigen:

In Lua gibt es eine Funktion `print()`, welche das, was zwischen den runden Klammern steht, im Ereignisfenster ausgibt. Die richtige Schreibweise ist:

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

Falsch wäre also:

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

Denn es gehört zur Syntax von Lua, dass eine Funktion namens **`P`**`rint()` als eigenständige Funktion betrachtet und von der Funktion **`p`**`rint()` unterschieden wird. Daher lautet die Fehlermedlung in diesem Fall:

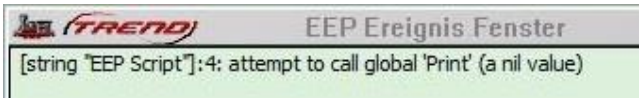

Bild 9

Der Interpreter hat nach der Definition einer Funktion namens `Print()` gesucht und nichts finden können. In der Sprache von Lua heißt "nichts" `nil`. Die Fehlermeldung sagt daher, dass der Interpreter bei seiner Suche `nil` zurück bekommen hat.

Wir möchten an dieser Stelle auf einen wichtigen Unterschied zwischen beiden Fehlermeldungen hinweisen: Während die erste Fehlermeldung ein eigenes Fenster geöffnet hat, wurde die zweite Fehlermeldung ins Ereignisfenster geschrieben. Der Grund dafür ist der Zeitpunkt, zu dem der Fehler vom Interpreter bemerkt wurde.

Bei der Übertragung eines Skripts an den Interpreter wird es auf Vollständigkeit geprüft. Ist das Skript unvollständig, dann wird es vom Interpreter abgelehnt und eine entsprechende Fehlermeldung wird im separaten Fenster ausgegeben. Die Zeile mit dem einzelnen Minuszeichen war in diesem Sinne unvollständig, weil vor dem Minuszeichen nichts stand und somit eine Subtraktion unmöglich wäre.

Fehler, die während der Übertragung des Skripts an den Interpreter erkannt werden, meldet EEP in einem eigenen Hinweis-Fenster.

Der Funktionsaufruf `Print("Hey let's start, EEP Version is: ", EEPVer)` war jedoch vollständig. Dass diese Funktion zum Zeitpunkt der Übertragung des Skripts noch nicht definiert war, ist in diesem Zusammenhang bedeutungslos. Diese Funktion könnte nämlich an anderer Stelle definiert sein oder zu einem späteren Zeitpunkt definiert werden. Daher wird der Fehler erst während der Ausführung des Skripts bemerkt.

Fehler, die erst während der Ausführung des Skripts erkannt werden, meldet EEP im Ereignisfenster.

Bitte denken Sie daran, dass das Ereignisfenster geöffnet sein muss, damit Lua Sie über Fehler informiert, die erst bei der Ausführung des Skripts erkannt werden.

Je größer ein Skript ist, desto schwerer wird es natürlich solche Schreibfehler zu finden. Daher möchten wir Ihnen empfehlen, Skripte in sehr kleinen Schritten zu entwickeln und jeden dieser Schritte in einem Testlauf zu prüfen. Auf diese Weise lässt sich die Ursache eines Fehlers leichter eingrenzen.

Welche Befehle stehen zur Verfügung

Eine Programmiersprache besteht im Wesentlichen aus Schlüsselworten, Operatoren, Variablen und Funktionen.

- Schlüsselworte sind festgelegte Worte, die den Interpreter veranlassen etwas zu tun.
- Operatoren sind die Zeichen, welche den Interpreter veranlassen etwas auszurechnen oder logisch zu verknüpfen.
- Variablen sind Speicherplätze, deren Inhalt durch das Skript verändert werden kann.
- Funktionen sind Sammlungen von Anweisungen, die unter einem Namen zusammengefasst werden.

Beispiele für Schlüsselworte sind

```
if, then, return, do, function, end
```

Beispiele für Operatoren sind

```
+, -, *, /, <, >
```

Beispiele für Variablen sind

```
EEPVer, EEPTime
```

Beispiele für Funktionen sind

```
clearlog(), print(), EEPSetSignal()
```

Die Schlüsselworte und Operatoren sind alle Bestandteil von Lua. Bei Variablen und Funktionen gibt es neben denen, die Lua von Hause aus mitbringt auch welche, die wir geschaffen haben damit Lua mit EEP kommunizieren kann. Darüber hinaus können Sie eigene Variablen und Funktionen erstellen.

Allgemeine Lua-Befehle

Eine vollständige Erklärung aller Elemente, die Lua mitbringt, würde den Rahmen dieses Handbuchs sprengen. Wir möchten deshalb auf Webseiten verweisen, welche detaillierte Auflistungen, Erläuterungen und auch Beispiele für die Anwendung enthalten:

http://lua.coders-online.net

http://lua.gts-stolberg.de/

http://www.lua.org/docs.html

Einige wichtige Elemente von Lua haben wir aber in dieses Handbuch aufgenommen. Sie stehen im Anhang I und die Beschreibungen zu den Tutorial Anlagen enthalten Links zu diesen Erläuterungen.

EEP-spezifische Befehle

Eine Auflistung aller EEP-spezifischen Variablen und Funktionen finden Sie im Anhang II

Mitgelieferte Beispiele

Die Anlagen selbst sind unspektakulär und nur mit wenigen Funktionen ausgestattet. Gerade deshalb eigenen sie sich besonders gut zum Studium der Skriptsprache Lua sowie für eigene Experimente. Wir möchten Ihnen empfehlen jede einzelne der Anlagen genau zu studieren.

Die Ereignisse auf den Tutorial-Anlage sind schnell verstanden. 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 erfordert hätten. Außerdem haben Sie mit einem Skript alles, was die Anlage steuert, an zentraler Stelle zusammengefasst.

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 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. Dafür sind die Tutorial Anlagen da. Aus Fehlern kann man viel lernen.

Die Erläuterungen zu den Tutorial Anlagen bauen aufeinander auf. Was wir beispielsweise im Kapitel zur Anlage "Tutorial_33_Lua_1" erläutern, dass setzen wir im Kapitel zur Anlage "Tutorial_34_Lua_2" als bekannt voraus.

Beschreibung der Anlage „Tutorial_33_LUA_1”

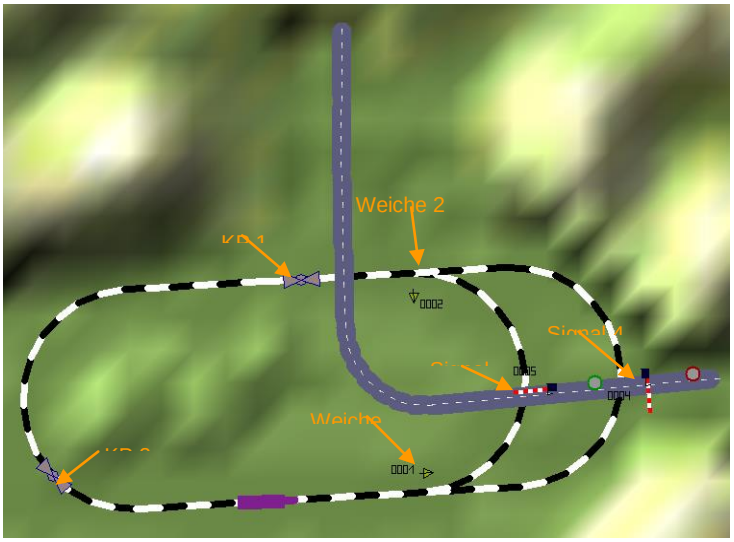

Bild 10

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 von 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. Wir möchten Ihnen empfehlen 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.

Auf der 2D Ansicht zur dieser Anlage (Bild 10) 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 Skripteditor (Bild 4) und betrachten Sie die ersten vier Zeilen des Skripts.

Sie enthalten Kommentare, die vom Interpreter ignoriert werden und nur als Merkhilfen dienen.

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

Sie können die Kommentarzeilen für einen ersten, eigenen Versuch nutzen. Löschen sie in einer Zeile eins der zwei Minuszeichen am Zeilenanfang.

Übertragen Sie anschließend das geänderte Skript an den Interpreter (Bild 7)

Ein Fenster wird sich öffnen und Ihnen melden, dass Ihr Skript einen Fehler enthält.

Der Skripteditor bleibt geöffnet, weil der Interpreter das fehlerhafte Skript abgelehnt hat.

Korrigieren Sie bitte den Fehler, übertragen Sie das korrigierte Skript erneut an den Interpreter 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 weiter läuft, aber Lua pausiert. Probieren Sie es 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.

Wir möchten Ihnen empfehlen 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 kennzeichenen `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 1) geöffnet werden (Bild 3). 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 übergeben hat und in den 3D Modus wechselt.

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 Schlüsselworten 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 Anführunsgzeichen 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 wir EEP mitgegeben haben, damit es mit Lua Daten austauschen kann. Sie ist im Anhang II beschrieben. `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 im 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 jedesmal eine Zahl zurück geben. 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 hinein schreibt. 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.

Wir möchten Ihnen an dieser Stelle empfehlen sich eingehend mit der Funktion `EEPMain()` zu befassen. Sie ist ein elementar wichtiges Bindeglied zwischen EEP und Lua. Im Anhang finden Sie weiterführende Erläuterungen dazu. Studieren Sie die bitte sorgfältig und nutzen Sie diese und andere Tutorial Anlagen, um experimentell zu lernen, welche Konsequenzen Änderungen an der Definition der Funktion haben.

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 11) aufgerufen, welcher in der linken Kurve sitzt. Öffnen Sie bitte das Eigenschaftsfenster des Kontaktpunktes.

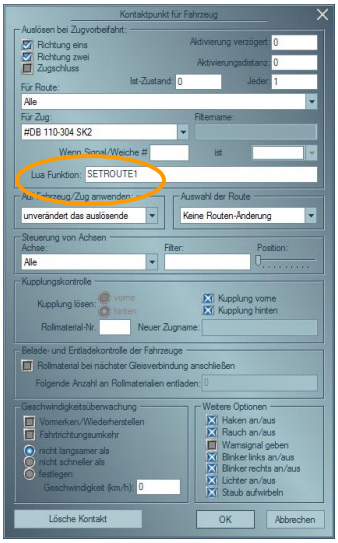

Bild 11

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!

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

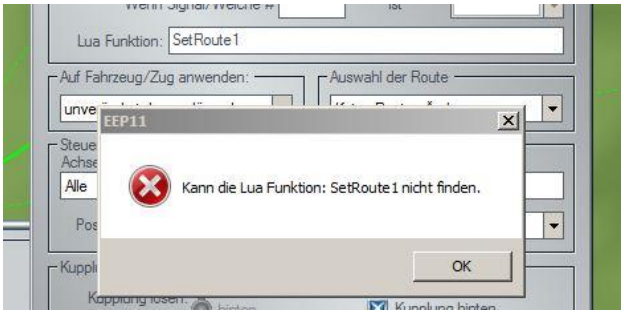

Bild 12

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 Fehler tritt erst in dem Moment auf, in dem der Zug den Kontaktpunkt überfährt. Daher erscheint die Fehlermeldung diesmal im Ereignisfenster:

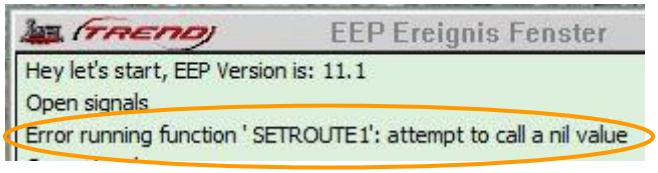

Bild 13

Man muss schon sehr genau hinschauen um zu erkennen, dass in der Fehlermeldung vor `SETROUTE1` ein Leerzeichen steht. Deshalb möchten wir auch hier empfehlen: 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.

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.

Dann folgt ein Kommentar. 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 der selben Variablen abspeichern kann.

Im konkreten Beispiel wird zur Variablen `route` jedesmal `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 "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 Modulo Operator. Der Ausdruck `route % 2` kann 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 `else`, `elseif` oder `end` stößt. Ist das Ergebnis hingegen false, dann springt Lua zum

nächsten `else`, `elseif` 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 14) Der Zug wird also die innere Runde fahren.

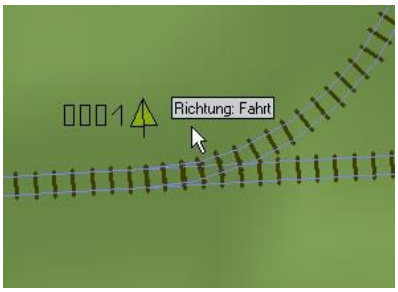

Bild 14

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 15), welche man im Eigenschaftsmenü findet.

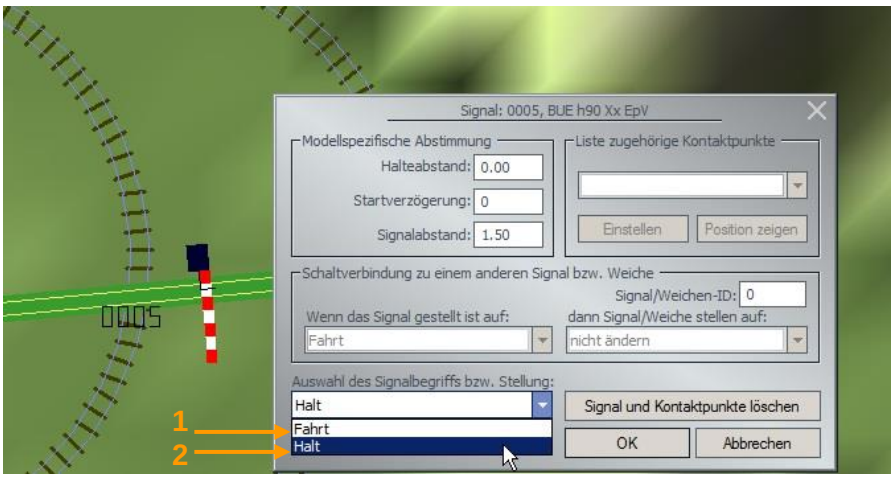

Bild 15

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 keinzeichnet 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()` und wird im Kontaktpunkt 1 (Bild 16) aufgerufen, welcher oben hinter der Weiche 2 sitzt. (Bild 10)

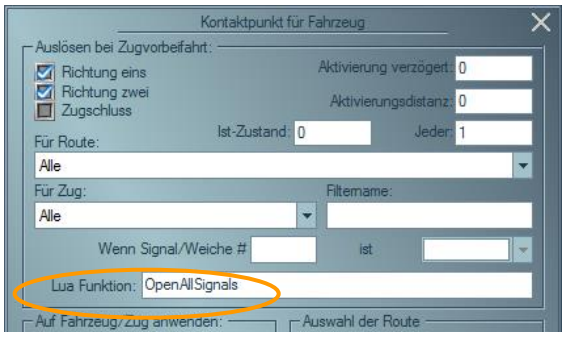

Bild 16

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.

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.

Beschreibung der Anlage „Tutorial_34_LUA_2”

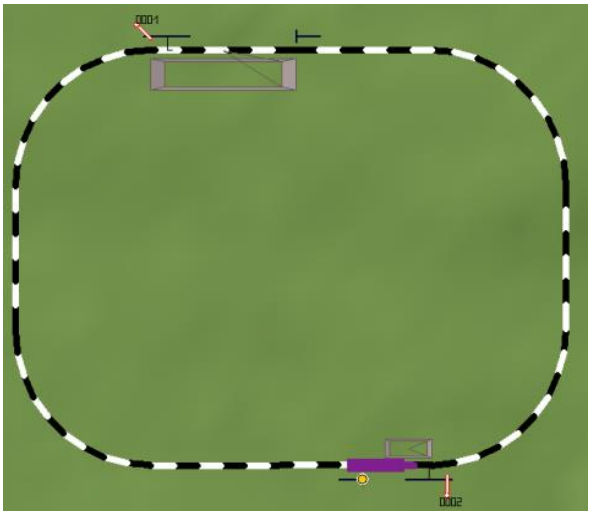

Bild 17

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 Callback 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:

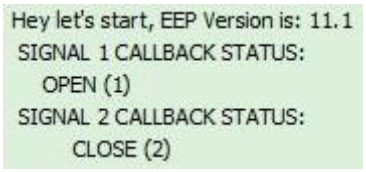

Bild 18

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.

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 darauf klicken, 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.

Beschreibung der Anlage „Tutorial_35_LUA_3”

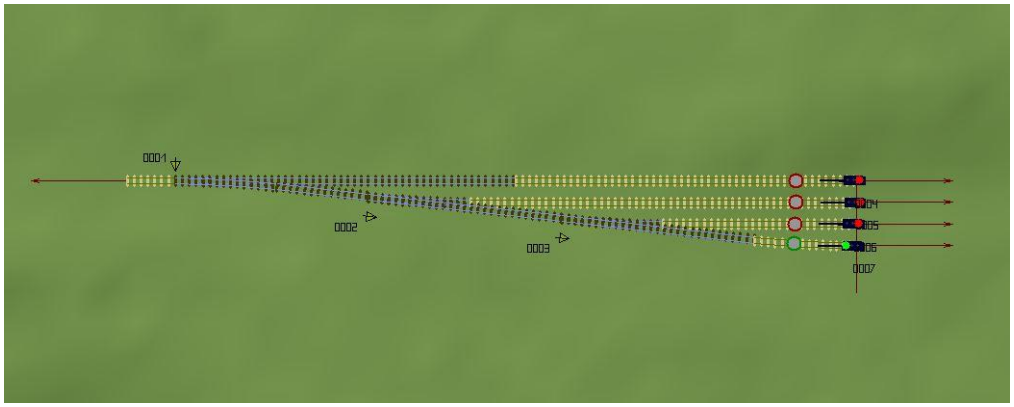

Bild 19

Dieses Tutorial enthält ein zweites Beispiel für die Verwendung von Callback Funktionen. Die verwendeten Methoden sind die selben 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 Weichen 1 und 5 sowie die Signale 6, 2 und 7 umgestellt:

```
        print("Set route 1")
        EEPSetSwitch(1, 1)
        EEPSetSwitch(5, 2)
        EEPSetSignal(6, 2)
        EEPSetSignal(2, 1, 1)
        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)
        EEPSetSignal(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 vorfanden, 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 Defnitionen 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 3), damit Sie bei Ihren Versuchen die Fehlermeldungen sehen können.

Beschreibung der Anlage „Tutorial_36_LUA_4”

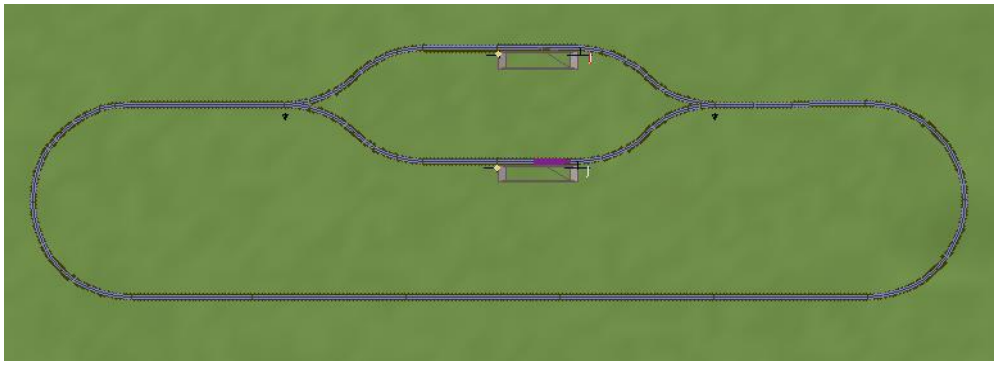

Bild 20

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

`EEPMain()`

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 Arrays definiert und beiden werden Elemente zugewiesen:

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

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

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 der selben Variable 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` 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 3), 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 Ausgabetext 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 dem Array `arr_oc`. Der Wert in der Variablen `signal4_state` bestimmt, ob der erste oder zweite Eintrag im Array 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 dem Array `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
```

Und zuletzt folgen zwei Funktionen, die durch Kontaktpunkte am Ende der beiden Bahnhofsgleise aufgerufen werden. Wie die zwei Funkltionen 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
```

Beschreibung der Anlage „Tutorial_37_LUA_5”

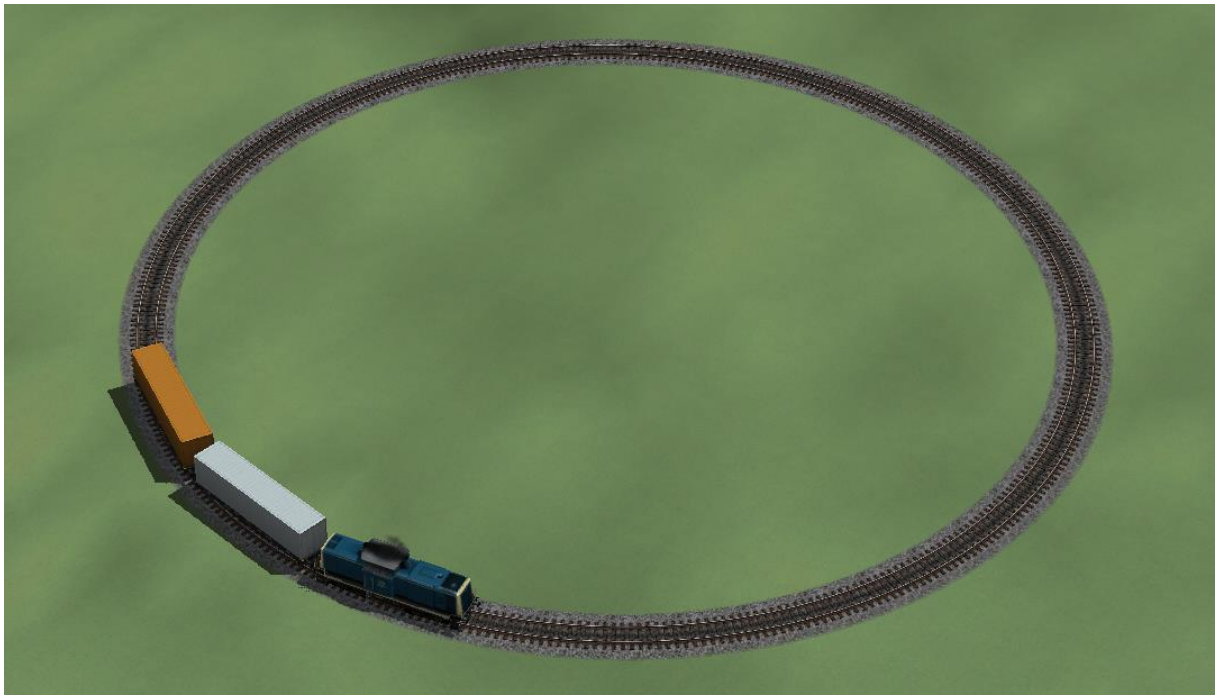

Bild 21

Benutzung eines Zeitgebers "cycle" um eine Aktion auszulösen.

Die Funktion EEPMain() wird 5 mal pro Sekunde aufgerufen, also wird der Zeitgeber "cycle" alle 1/5 Sekunden hochgezählt.

Dieses Feature wird hier benutzt, um Abfolgen von Ereignissen zu programmieren, wie etwa das Ändern der Zuggeschwindigkeit - EEPSetTrainSpeed(Name, Geschwindigkeit) oder Kuppeln und Entkuppeln von Fahrzeugen - EEPRollingstockSetCouplingRear(Name, Kupplungszustand) wobei Kupplungszustand bedeutet: 1 - Kupplung aktiv, 2 - Kupplung inaktiv (abkuppeln). Die Anweisung hResult, value = EEPRollingstockGetCouplingRear(Name) liefert den Wert 3, wenn eine gekuppelte Kupplung vorliegt.

Code aus dem Beispiel:

```
cycle=0
clearlog()

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

function EEPMain()

      cycle = cycle + 1

      if (cycle == 1) then

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

      elseif (cycle == 30) then

            -- STOP --
```

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

      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 )


      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 )

      elseif (cycle > 55) then

            hResult, value = EEPRollingstockGetCouplingRear("DB 212 309")
            if(hResult) then
                  if(value == 3) then
                        -- DETECTED CONNECTION --
                        print("DETECTED CONNECTION at cycle: ", cycle)
                        print("RESET counter")
                        cycle = 0
                  end
            end

      end

      return 1
end
```

Beschreibung der Anlage „Tutorial_38_LUA_6”

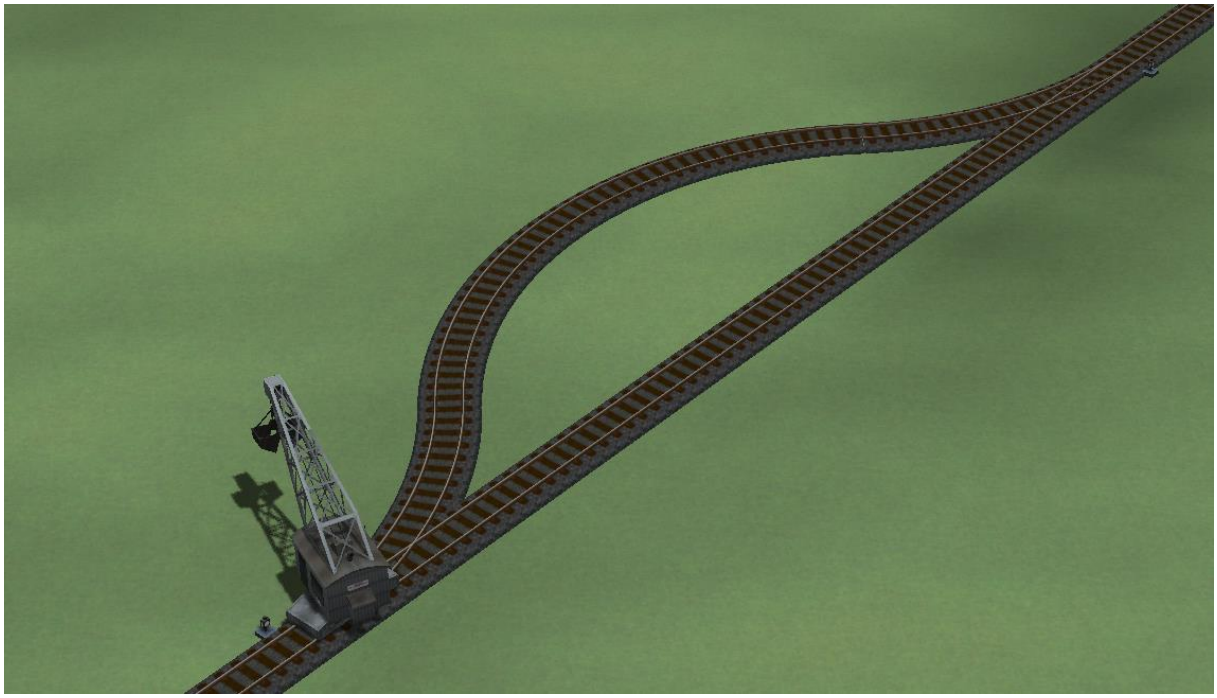

Bild 22

Kontaktpunkte rufen in ihren Dialogen definierte Funktionen auf: Contact1() und Contact2().

Jede Funktion enthält eine Liste von Anweisungen, welche Ereignisse auslösen:

Zuggeschwindigkeit - EEPSetTrainSpeed(Name, Geschwindigkeit),

Aktive Achsengruppe (Achsengruppen enthalten Einstellungen für Achsen und werden im 3D - Editor für ein Fahrzeug definiert) - EEPRollingstockSetSlot(Name, Achsengruppe)

Weiche stellen - EEPSetSwitch(Weichennummer, Stellung)

Code aus dem Beispiel:

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

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

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

Beschreibung der Anlage „Tutorial_39_LUA_7”

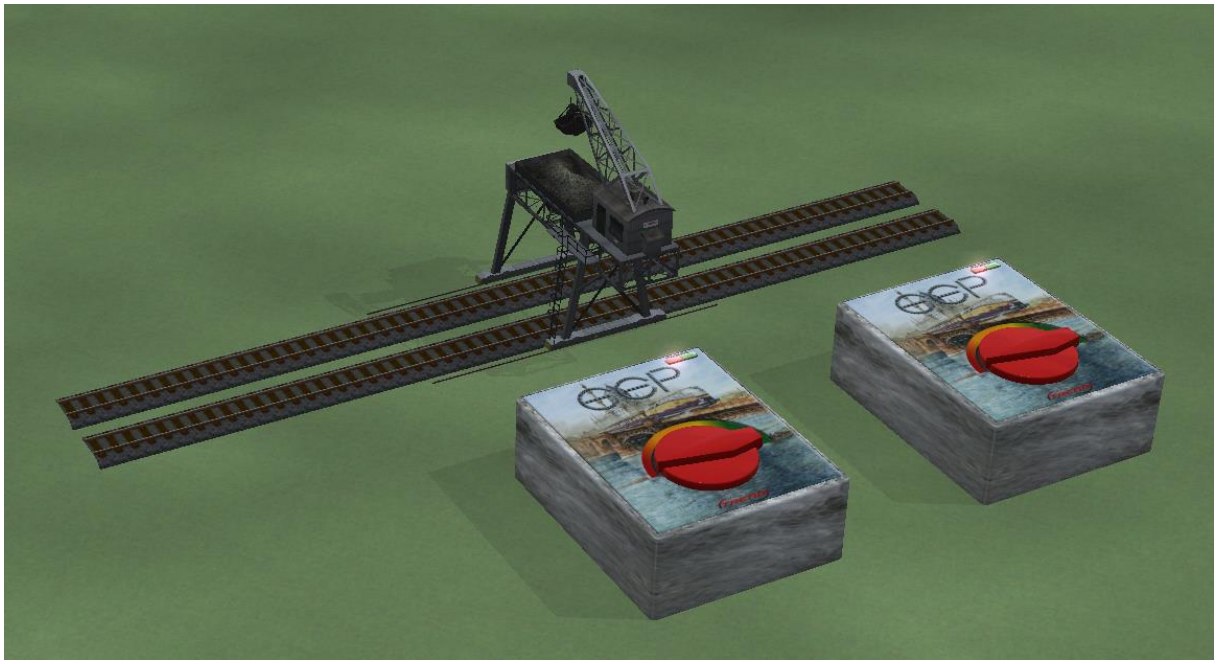

Bild 23

Achsensteuerung mit Lua Skripten

Wenn der Nutzer einen von zwei Schaltern betätigt werden entweder alle Achsen auf einmal angesteuert oder aber eine nach der Anderen.

Code aus dem Beispiel:

```
counter1 =0
counter2 =100
clearlog()
print("Hey let's start, EEP Version is: ", EEPVer)
EEPRegisterSignal(1)
EEPRegisterSignal(2)

function EEPMain()
      counter1 = counter1+1
      counter2 = counter2+1
      -- SWITCH OFF SIGNAL 1 AUTOMATICALLY --
      if(counter1 > 12) then
            -- hResult, value = EEPGetSignal(1)
            -- print("-->>", hResult, "    ", value)
            value = EEPGetSignal(1)
            -- if (hResult) then
                  if (value == 1) then
                        hResult = EEPSetSignal(1, 2)
                  end
            -- end
      end
      -- SWITCH OFF SIGNAL 2 AUTOMATICALLY --
      if (counter2 > 12) then
            -- hResult, value = EEPGetSignal(2)
            Value = EEPGetSignal(2)
            if (Value == 1) then
                  hResult = EEPSetSignal(2, 2)
            end
```

```
	end
	-- CALL PROPER FUNCTION DEPENDING ON counter2 VARIABLE --
	if(counter2 == 1) then GoStep1()
	elseif(counter2 == 20) then GoStep2()
	elseif(counter2 == 30) then GoStep3()
	elseif(counter2 == 60) then GoStep4()
	end
	return 1
end

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)
		counter1 = 0
	end
end

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

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
```

Beschreibung der Anlage „Tutorial_40_LUA_8”

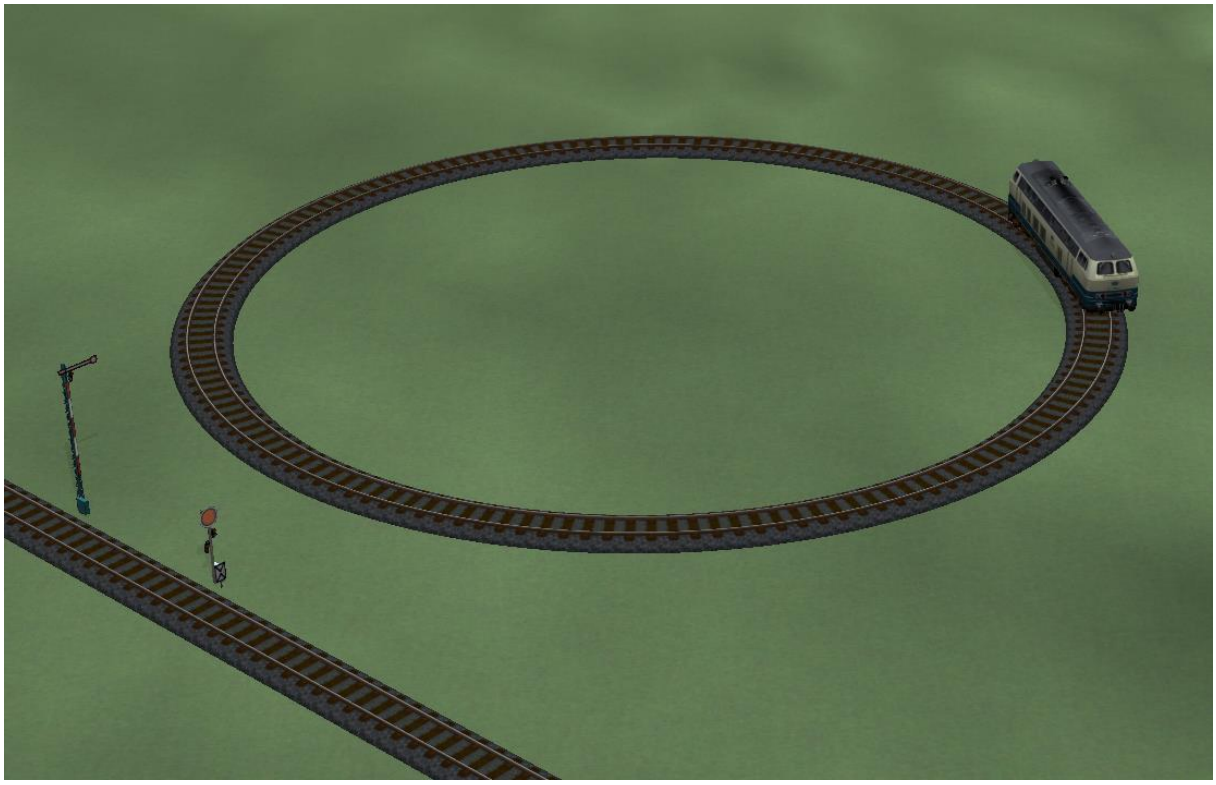

Bild 24

Neue Anweisungen zum Speichern und Laden.

```
hResult, wert = EEPLoadData(slot_nr)
```

und

```
hResult = EEPSaveData(slot_nr, wert)
```

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".

Code aus dem Beispiel:

```
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

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

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

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
```

Beschreibung der Anlage „Tutorial_44_LUA_1”

Bild 25

Dieses Tutorial demonstriert die Steuerung der Immobilien-Eigenschaften Rauch, Licht und Feuer (Brand)

`EEPStructureSetSmoke()`
`EEPStructureSetLight()`
`EEPStructureSetFire()`

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 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.

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

```
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
```

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_Abfertigung Lauscha" )
    hResult, IsLight = EEPStructureGetLight ( "#2_Betriebsdienstgebaeude" )
    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` zurück gesetzt, 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` 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_Betriebsdienstgebaeude", 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" jetzt 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.

Beschreibung der Anlage „Tutorial_45_LUA_2”

Bild 26

Dieses Tutorial zeigt den Umgang mit animierbaren Achsen bei Immobilien und Gleisobjekten.

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 fuer endlose Bewegung

hResult = 0          -- Variable um Ergebnisse aufzunehmen
Angle = 0            -- Variable fuer den Winkel der Drehscheibe
PreviousAngle = -1   -- Variable fuer den aktuellen Winkel der Drehscheibe
Speed = 0            -- Variable fuer die Geschwindigkeit
```

Die `EEPMain()` Funktion gibt bei jedem Durchlauf erneut den aktuellen Winkel der Drehscheibe aus und ruft die Funktion `CheckAnRunTrain()` 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 herum führt, ruft die Funtion `CallFromContactPoint1()` auf. Darin befindet sich ein Funktionsaufruf, der die Mühle anhält.

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

Die Funktion 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" jetzt 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.

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.

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

Der erste Parameter der Funtion `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 weiter bewegen soll. Beim verwendeten Modell entsprechen 9 Schritte einer halben Drehung.

Die Funktion `CheckAndRunTrain()`, welche bei jedem Durchlauf von der `EEPMain()` aufgerufen wird, ist wie folgt definiert:

```
function CheckAndRunTrain()
```

Zuerst liest sie die aktuelle Position der Drehscheibe aus und speichert sie in der Variablen `Angle`:

```
    hResult, Angle = EEPStructureGetAxis("#10_Drehscheibe", "Brücke")
```

Dann ermittelt sie die aktuelle Geschwindigkeit der Lok und speichert sie in der Variablen `Speed`:

```
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

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

prüft die Funktion, ob die Drehscheibe und die Lok zum Stillstand gekommen sind

```
if(Angle == PreviousAngle and Speed == 0) then
```

um dann die Lok wieder in Bewegung zu setzen:

```
    EEPSetTrainSpeed("#DB 120-119 or EpIV", 50)
```

Oder andernfalls gar nichts zu tun

```
end
```

Zuletzt wird der aktuelle Winkel der Drehscheibe an die Variable Previous Angle übergeben

```
PreviousAngle = Angle
```

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 etwas weiter oben zunutze um zu prüfen, ob die Drehscheibe still steht.

```
end
```

Das ist das Ende der Funktion und damit ist auch das Ende dieses Skripts erreicht.

Beschreibung der Anlage „Tutorial_46_LUA_3”

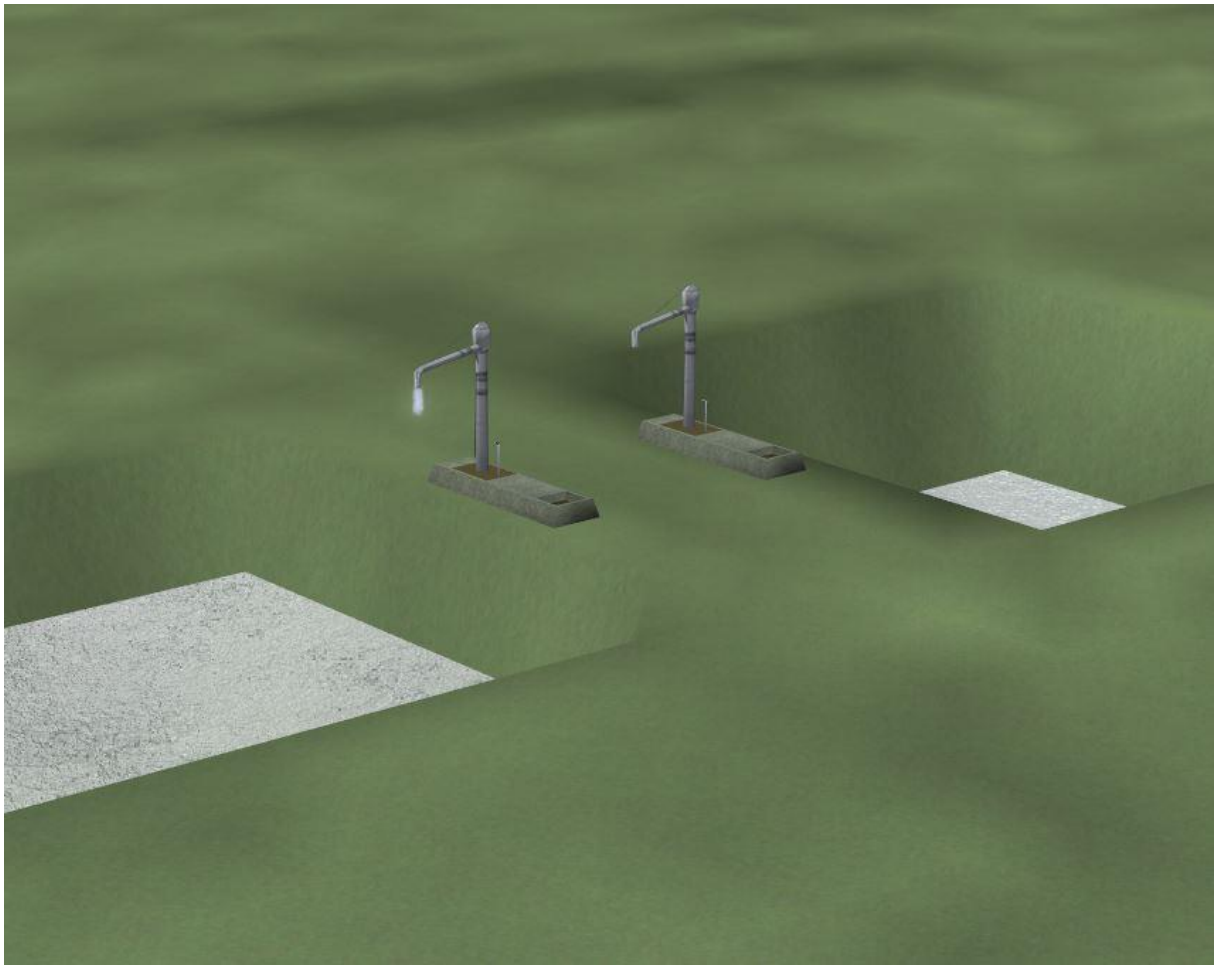

Bild 27

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 fuer den Wasserstand
FillTempo = 0.3             -- Variable fuer die Fuellgeschwindigkeit
Water = 1                   -- Variable fuer das Wasser selbst
FillNone = 0                -- Variable fuer die Fuellung
FillUp = 1                  -- Variable, ob gefuellt werden soll
FillOut = 2                 -- Variable, ob geleert werdden soll
WaterLevelCondition = -8    -- Variable zum Wasserzustand
AxisValueCondition = 50     -- Variable zum Achs-Wert
```

```
hResult = 0                  -- Variable um Ergebnisse aufzunehmen
```

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 haben wir 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
```

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 dem selben 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. Die Position wird ermittelt ...

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

... im Ereignisfenster ausgegeben ...

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

... und nachgeschaut, ob die Drehung mindestens den Wert 80 hat.

```
    if( AxisValueCondition > 80 ) then
```

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 ( 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 ...

```
    else
```

... wird beim Wasserkran der Wasserfluss gestoppt ...

```
        EEPStructureSetAxis("#3_Wasserkran", "Wasser", 0)
```

--- und geprüft, ob im Teich noch Wasser ist.

```
        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
```

Und hier endet die Definition dieser Funktion.

```
end
```

Beschreibung der Anlage „Tutorial_48_LUA”

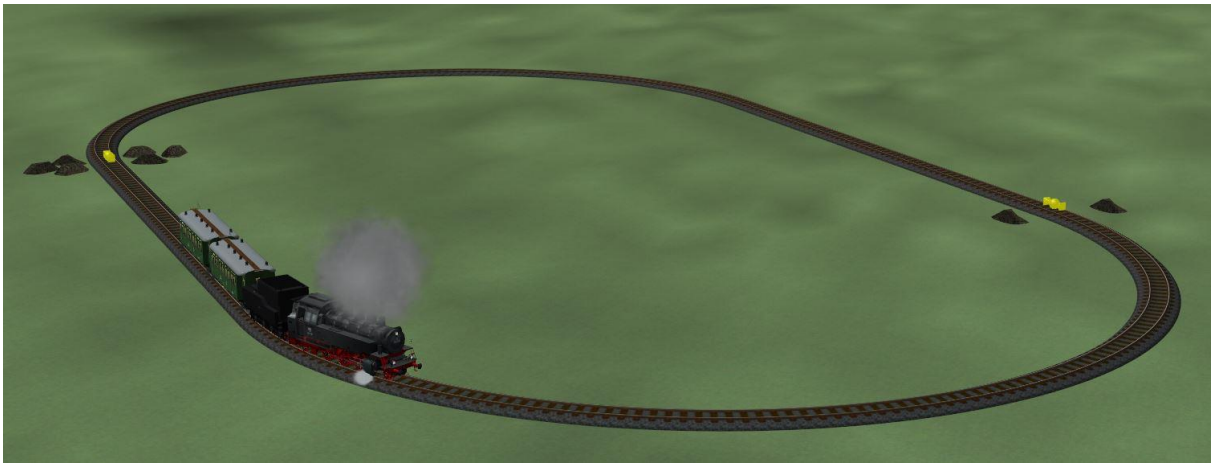

Bild 28

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 Plugin 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 `hResult` 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.

In der folgenden Funktion

```
function TurnOffEffects(name)
```

wird das selbe Prinzip verwendet, um Licht und Rauch auszuschalten. Der Warnton wird nicht ausgeschaltet, weil er zu diesem Zeitpunkt längst verklungen ist.

Beschreibung der Anlage „Tutorial_49_LUA”

Bild 29

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.

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 Plugin 2 zu EEP 11 immer den Namen des auslösenden Zuge 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
function TakeGoodsStart(name)
      TrainName = name

      hResult = EEPSetTrainSpeed(name, 0)
      print("-- Stop train : ", hResult)


      hResult = EEPSetTrainAxis(name, "Ausleger heben/senken", 100)
      print("-- Turn crane up : ", hResult)
      hResult = EEPSetTrainAxis(name, "Drehung nach rechts", 90)
      print("-- Turn crane right : ", hResult)


      hResult = EEPSetTrainHook(name, false)
      print("-- Temporary switch off the hook : ", 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 zurück gesetzt, wenn der Zug den Kontaktpunkt überfährt und nicht erst, wenn die Aktionen "Zug stoppen, Ausleger heben und Drehen, Haken auschalten" 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
                  TakeGoods(TrainName)
            end

            if TimeCounter == 60 then
                  print("-- Hook goods")
                  EEPSetTrainHook(TrainName, true)
                  print("-- Crane up")
                  EEPSetTrainAxis(TrainName, "Ausleger heben/senken", 100)
            end

            if TimeCounter == 80 then
                  print("-- Turn left")
                  EEPSetTrainAxis(TrainName, "Drehung nach rechts", 70)
            end

            if TimeCounter == 100 then
                  print("-- Unhook goods")
                  EEPSetTrainHook(TrainName, false)
            end

            if TimeCounter == 110 then
                  print("-- Turn crane arm to default position")
                  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 zurück gesetzt, wenn er den Kontaktpunkt überfährt.

Nach 30 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("-- Turn crane down : ", 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(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, das zum Beispiel nach 30 Durchläufen 6 Sekunden vergangen sind.

Beschreibung der Anlage „Tutorial_50_LUA”

Bild 30

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 werden 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 Plugin 2 zu EEP 11 immer den Namen des auslösenden Zuge 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 geprüft, ob der Name ermittelt werden konnte

```
if hResult then
```

`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.

Beschreibung der Anlage „Tutorial_51_LUA”

Bild 31

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

```
EEPSetTrainCouplingFront() -- Im Tutorial nicht enthalten
EEPSetTrainCouplingRear()
EEPTrainLooseCoupling()
```

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

Funktionsaufrufe aus Kontaktpunkten geben seit Einführung des Plugin 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 Wagons 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 Zugverbands. Daher fährt der Zug wieder mit 50 km/h weiter und der Zyklus beginnt von neuem.

Beschreibung der Anlage „Tutorial_55_Gleisbesetzt”

Dieses Tutorial demonstriert die Verwendung der Gleisbesetzt Funktionen für Gleise.

EEPRegisterRailTrack()
EEPIsRailTrackReserved()

Der Kühlwagen versperrt dem Güterzug den Weg. 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. zu dienen die beiden Zeilen:

```
Gleis1Reg = EEPRegisterRailTrack(20)
Gleis2Reg = EEPRegisterRailTrack(2)
```

Die Variablen Gleis1Reg und Gleis2Reg dienen der anschließenden Prüfung, ob Gleise, die registriert werden sollten, tatsächlich existieren. Sie sind true, falls die Gleise erfolgreich registriert wurden, andernfalls false.

Die eigentliche Besetzt-Prüfung findet in der Funktion Signal5Automatik() statt, welche bei jedem Durchlauf der EEPMain() Funktion aufgerufen wird, wenn die Registrierung der Gleise erfolgreich war.

```
function Signal5Automatik()
```

Da die Gleisbesetzt-Funktion zwei Werte zurück liefert, werden diese zunächst zwischengespeichert:

```
    hResult1,Besetzt1 = EEPIsRailTrackReserved(20)
    hResult2,Besetzt2 = EEPIsRailTrackReserved(2)
```

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

```
    if hResult1 and hResult2 then
```

Der zweite Rückgabewert besagt, ob Rollmaterial auf dem Gleis steht.

```
        if Besetzt1 or Besetzt2 then
```

Ist das Gleis besetzt, dann ist der Wert true. Das Signal bleibt dann auf "Halt.

```
            EEPSetSignal(5,2)
```

Nur, wenn beide Gleise false, also "nicht besetzt" zurück melden, wird die Fahrt frei gegeben..

```
        else
            EEPSetSignal(5,1)
        end
```

Der Vollständigkeit halber wird auch noch 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.

```
    else
        print("Prüfung fehlgeschlagen!")
end
```

Das Skript prüft außerdem, ob die EEPVersion überhaupt geeignet ist. Denn Mindestvoraussetzung für diese Tutorial-Anlage ist EEP Version 11.3

Ein installiertes Plugin 3 zu EEP 11 ist ebenfalls Voraussetzung, kann durch Lua aber nicht überprüft werden. Deshalb wird der wiederholte Aufruf der EEPMain() Funktion gestoppt, wenn die beiden Gleise nicht registriert werden konnten.

Zusammenfassung

Es war unser Ziel Ihnen mit dieser Dokumentation einen ersten Einblick in die große Welt der Programmierung zu geben. Einen Ratschlag möchten wir Ihnen an dieser Stelle noch mit auf den Weg geben: Versuchen Sie gerade zu Beginn Ihrer Tätigkeit als Lua-Programmierer nicht allzu schwierige Aufgaben umzusetzen. Arbeiten Sie nach dem Prinzip „Vom leichten zum schweren“. Wenn Sie eine einfache Aufgabe lösen konnten, dann werden Sie diese Erfahrungen bei einer nächsten, vielleicht schwierigeren Aufgabe einbringen können. Anders herum besteht die Gefahr, dass Sie an einer zu ambitionierten Aufgabe scheitern. Starten Sie also besser mit Erfolgserlebnissen und steigern Sie sich.

Probieren Sie am besten die beschriebenen Funktionen der Testanlagen aus. So können Sie sehr einfach die einzelnen Schritte Ihrer Tätigkeit nachvollziehen und verlieren sich nicht in einer zu komplizierten Steuerung.

Ausblick

Der Umfang an EEP-spezifischen Lua Funktionen wird in den kommenden Jahren immer weiter wachsen. Die bisher verfügbaren Befehle stellen also nur den Beginn eines Weges dar, auf den wir uns schon heute sehr freuen.

Wir wünschen Ihnen nun viel Erfolg und natürlich auch viel Spaß bei der Automatisierung Ihrer Anlagen mit Lua-Skripten!

Ihr EEP Team

Anhang I

Kommentare

Kommentare dienen in Skripten als Gedächtnisstützen und als Erklärungen für andere, die ein Skript lesen und verstehen möchten.

Die Syntax von Lua erfordert, dass einem Kommentar zwei Bindestriche vorangestellt werden.

```
-- Dies ist ein Kommentar
```

Alles, was in der Zeile nach den beiden Bindestrichen folgt, wird vom Interpreter ignoriert. Auch dann, wenn es Funktionen, Schlüsselworte oder andere Elemente aus der Sprache Lua enthält.

```
-- Die folgende Funktion wird nicht ausgeführt: print("Kommentar")
```

Die nächste Zeile wird wieder als Code betrachtet.

Nach den ersten beiden Bindestrichen darf alles stehen. Also auch weitere Bindestriche

```
---------- Ich bin auch ein Kommentar ----------
```

Kommentare dürfen auch hinter auszuführendem Code stehen

```
print("Hallo") -- gibt den Text aus, der in Anführungsstrichen steht
```

Es ist ratsam ein Skript mit vielen und aussagekräftigen Kommentaren zu versehen. Sie helfen einem den Überblick zu bewahren. Wenn Sie ein Skript erstellen, welches Sie anderen Usern zur Verfügung stellen möchten - sei es zur Verwendung oder auch, *damit die mal einen Blick drauf werfen* - dann sind erklärende Kommentare geradezu Pflicht. Sie ersparen dem Leser damit die Mühe sich in Ihr Skript *hinein denken* zu müssen.

Programmierer nutzen die Kommentare auch gerne, um Teile ihres Skripts zeitweilig zu deaktivieren. Denn so muss man die Programmzeilen nicht löschen und später neu schreiben.

```
-- EEPSetSignal(1,2)
-- EEPSetSwitch(3,1)
-- print("Signal 1 auf Fahrt und Weiche 3 auf Fahrt gestellt")
```

Benötigt man diese Zeilen im Skript wieder, dann entfernt man einfach die beiden Bindestriche davor.

Diesen Trick sollten Sie sich zunutze machen, wenn Sie Experimente mit den Tutorials anstellen. Deaktivieren Sie bestimmte Zeilen im Code um zu beobachten, was diese tun bzw. dann nicht mehr tun. Achten Sie aber bitte beim erneuten Aktivieren der Codezeilen darauf, dass Sie nicht versehentlich die Bindestriche vor einer wirklichen Kommentarzeile entfernen. Denn sonst wird der Interpreter den nachfolgenden Text als Code lesen und versuchen auszuführen, was mit Gewissheit Fehlermeldunen zur Folge hat.

Variablen

Als Variablen bezeichnet man Speicherplätze, in denen etwas zur späteren Verwendung abgelegt wird. Der Inhalt dieser Speicher kann vom Programm verändert werden.

Sie können eine Variable grob mit einem Notizzettel vergleichen, auf dem Sie zum Beispiel eine Strichliste führen. Die Anzahl der Striche auf diesem Zettel verändert sich jedesmal, wenn Sie einen Strich hinzu fügen oder ausradieren. Aber der Zettel bleibt stets der selbe. Sie wissen also immer, wo Sie nachschauen müssen um zu prüfen, wieviele Striche sie gerade gemacht haben.

Eine Variable bekommt einen Namen. Anschließend können Sie immer, wenn Sie es benötigen, den Wert erhalten, der unter diesem Namen gespeichert wurde. Und den Wert ändern, wenn die Situation das erfordert.

Zur Syntax von Lua gehört eine strenge Namensregel für Variablen und Funktionen.

Erlaubt sind ausschließlich

- Groß- und Kleinbuchstaben aus dem lateinischen Alphabet.
- Ziffern
- und der Unterstrich

Verboten sind

- Umlaute wie ä, ö, ü und andere exotische Buchstaben und Sonderzeichen wie ß, Ø, €, #
- sämtliche Satzzeichen und Sonderzeichen wie Punkt, Komma, Raute, Stern, Plus und Minus
- und das Leerzeichen

Außerdem darf ein Name nicht mit einer Ziffer beginnen!

Variablen kann man an jeder Stelle als Platzhalter für etwas einsetzen. Sie können Zahlen, Texte und anderes mehr enthalten.

Eine typische Anwendung ist die eingangs erwähnte Strichliste.

Zunächst erzeugen wir eine Variable, indem wir ihr einen Wert zuweisen:

```
StrichListe = 0
```

Dann weisen wir ihr einen neuen Wert zu, indem wir den aktuellen Wert auslesen, `1` hinzuzählen und das Ergebnis wieder in der selben Variable speichern:

```
StrichListe = StrichListe + 1      -- Jetzt hat die Variable StrichListe den Wert 1
```

Und diesen Vorgang können wir beliebig oft wiederholen

```
StrichListe = StrichListe + 1      -- Jetzt hat die Variable StrichListe den Wert 2
```

Diese Methode funktioniert deshalb, weil zuerst der Ausdruck rechts vom = Zeichen ausgerechnet und dann das Ergebnis der Variablen links vom = Zeichen zugewiesen wird.

Funktionen

Bei Funktionen muss man zwischen Definition und Aufruf unterscheiden. Um das besser nachvollziehen zu können empfehlen wir Ihnen an dieser Stelle die erste der Lua-Tutorial Anlage „Tutorial_33_LUA_1.anl3" zu öffnen. Das Skript zur Anlage enthält sowohl Funktionsaufrufe als auch Funktionsdefinitionen.

In der fünften Zeile des Skripts steht

```
clearlog()
```

Das ist der Aufruf einer Funktion namens `clearlog`. Sie wird ausgeführt, sobald der Interpreter diese Zeile des Skripts abarbeitet.

Ebenso ist

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

ein Funktionsaufruf.

Am Ende des Skripts finden sie eine Funktionsdefinition. Vorher steht auch schon eine, aber diese ist kürzer und soll deshalb als Beispiel dienen:

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

Die Definition einer Funktion wird durch das Schlüsselwort `function` eingeleitet. Es teilt dem Interpreter mit, dass er die nachfolgenden Anweisungen nicht ausführen, sondern sich unter dem Namen, der hinter dem Schlüsselwort steht, merken soll.

Weil die Liste der Anweisungen unterschiedlich lang sein kann, muss dem Interpreter auch mitgeteilt werden, wo sie endet. Dafür dient das Schlüsselwort `end`.

Im Beispiel merkt Lua sich also unter dem Namen `OpenAllSignals`, dass zuerst mit `print()` ein Text ausgegeben und dann die Signale 4 und 5 umgestellt werden sollen.

Der Zweck einer Funktion ist ein vereinfachter Umgang mit Aufgaben, die mehrfach ausgeführt werden müssen. Im vorliegenden Fall geht es darum, dass die Schranken der beiden Bahnübergänge in jeder Runde, welche die Lok dreht, geöffnet werden sollen, wenn der Zug durch ist.

Hat man diese Aufgabenliste einmal erstellt, dann kann man sie beliebig oft nutzen indem man die Funktion per Namen aufruft.

Bei genauem Studium des Skripts zum Tutorial 33 fällt auf, dass einerseits Funktionen genutzt werden, zu denen die Definition fehlt und andererseits Funktionen definiert sind, die nicht aufgerufen werden. Der Grund dafür liegt im Zusammenspiel von EEP und Lua. Die Funktionen `clearlog()`, `print()` und `EEPSetSignal()` sind in EEP definiert und können daher sofort genutzt werden. Umgekehrt werden die Funktionen `EEPMain()`, `SETROUTE1()` und `OpenAllSignals()` von EEP aufgerufen und müssen daher nur definiert werden.

Funktionen erkennt der Interpreter and den runden Klammern hinter dem Namen. Sie sind zugleich der Speicherplatz für die Übermittlung von Inhalten, welche die Funktion bei ihrer Ausführung verwenden soll. So steht beispielsweise in den Klammern der `print()` Funktion der Text, den `print()` ins Ereignisfenster schreiben soll.

Die runden Klammern sind sowohl beim Aufruf als auch bei der Definition einer Funktion zwingend erforderlich. Und zwar auch dann, wenn sie ohne Inhalt bleiben, also nichts übertragen sollen wie zum Beispiel bei `clearlog()`.

Der Speicherplatz, welcher durch die runden Klammern repräsentiert wird, macht Funktionen flexibel. So kann beispielsweise die Funktion `print()` ganz unterschiedliche Texte ausgeben, obwohl dafür stets ein und dieselbe Funktion benutzt wird.

Das gleiche gilt für die Funktion `EEPSetSignal()`, welche im Skript mehrfach verwendet wird. Sie kann jedes Signal in jede mögliche Stellung schalten, obwohl diese Funktion nur einmal in EEP definiert wurde. Beim Aufruf schreibt man zwei Zahlen in die runden Klammern: Zuerst die ID, also die Nummer des Signals und dann die gewünschte Signalstellung. Die beiden Zahlen müssen durch ein Komma getrennt werden, damit sie bei der Ausführung richtig erkannt und zugewiesen werden.

Die Signalstellung, welche die zweite Zahl vorgibt, entspricht der Position unter "Auswahl des Signalbegriffs bzw. Stellung" in den Eigenschaften des Signals. (Bild 32).

Bild 32

% - der Modulo Operator

Der Modulo Operator ist so nützlich, dass wir ihm ein eigenes Kapitel widmen möchten.

In der Syntax von Lua ist `%` das Zeichen für "Modulo"

Ein `%` Zeichen im Lua Code hat nichts mit klassischer Prozentrechnung zu tun. Die Entwickler von Lua haben dieses Zeichen einfach für einen anderen Zweck *missbraucht*.

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

`0 % 5 ist 0` Ist der Wert links kleiner als der rechts vom `%`, dann ist das Ergebnis die linke Zahl

```
1 % 5 ist 1
2 % 5 ist 2
3 % 5 ist 3
4 % 5 ist 4
```

`5 % 5 ist 0` Ist der Wert erreicht, der rechts vom `%` steht, dann ist das Ergebnis `0`

`6 % 5 ist 1` Die `5` wird von der linken Zahl subtrahiert, bis der Rest kleiner als `5` ist

```
7 % 5 ist 2
...
10 % 5 ist 0
...
```

`-1 % 5 ist 4` Natürlich funktioniert Modulo auch mit negativen Zahlen.

Vielleicht fällt es Ihnen leichter Modulo zu verstehen, wenn wir zwei Zahlenreihen übereinander stellen. Oben die Zahlen, welche umgerechnet werden und darunter die Werte für `% 5`

```
-9 -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
 1  2  3  4  0  1  2  3  4  0  1  2  3  4  0  1  2  3  4  0  1  2  3  4  0
```

Wenn man Modulo auf Zähler anwendet, die man zugleich erhöht, dann muss man die Reihenfolge beachten.

```
(Wert + 1) % 4 ergibt immer Zahlen von 0 bis 3
(Wert % 4) + 1 ergibt immer Zahlen von 1 bis 4
```

Die zweite Methode wird man im Zusammenhang mit EEP häufiger finden, weil Gleisnummern, Signal- und Weichenstellungen und vieles andere mehr nicht mit 0, sondern mit 1 beginnen.

Arrays und Tabellen

In Lua sind Arrays nichts anderes als eindimensionale Tabellen. Deshalb werden wir in diesem Kapitel ausschließlich von Tabellen sprechen. Eine Tabelle ist deshalb so nützlich, weil sie viele Werte unter einem Namen zusammenfasst.

In der Syntax von Lua wird eine Tabelle erstellt, indem man ihr etwas in geschweiften Klammern zuweist:

```
meineTabelle = {}
```

Die Klammern können anfangs leer bleiben oder gleich einen Inhalt bekommen.

Bei mehreren Einträgen müssen sie jeweils durch ein Komma getrennt werden.

```
meineTabelle = {4, 8, 15, 16, 23, 42}
```

Die Plätze einer Tabelle werden als Index bezeichnet. In der Syntax von Lua schreibt man den Index in eckige Klammern. Ein Index kann wahlweise eine Nummern oder ein Name sein:

`meineTabelle[1]` steht für den ersten Eintrag in der Tabelle

Wenn wir die Tabelle also mit der vorherigen Zeile befüllt haben, dann würde

`print(meineTabelle[1])` anschließend eine `4` ausgeben.

Mit

```
meineTabelle[1] = 123
```

würde der erste Eintrag in der Tabelle durch die Zahl 123 ersetzt.

Namen für Tabellenplätze kann man entweder so

```
meineTabelle["Signal"]
```

oder so

```
meineTabelle.Signal
```

schreiben.

Die zweite Variante ist nur lesbarer und bequemer als die erste, aber in ihrer Bedeutung identisch.

Bei der Initialisierung benutzt man das = Zeichen für Zuweisungen, um Tabellenplätzen einen Namen zu geben

```
meineWeiche = { ID = 2 , Stellung = 1, Fahrstraße = 4 }
```

`meineWeiche.ID` ist anschließend 2

`meineWeiche.Stellung` ist anschließend 1 und

`meineWeiche.Fahrtsraße` ist anschließend 4

Da ein Tabellenplatz selbst eine komplette Tabelle enthalten kann, sind auch mehrdimensionale Tabellen möglich:

```
meineWeiche = {}
meineWeiche[1] = { ID = 2 , Stellung = 1, Fahrstraße = 4 }
meineWeiche[2] = { ID = 7 , Stellung = 2, Fahrstraße = 4 }
```

`meineWeiche[2].Stellung` ist anschließend 2

Tabellen sind eine kompakte und bequeme Form um größere Datenmengen zu verwalten.

Anhang II

EEP-spezifische Lua-Variablen und -Funktionen

bis einschließlich EEP 13

System-Variablen

EEPVer		EEPVer
Typ	Variable	
Verwendung in	Skript	
Zuweisung durch	EEP	
Voraussetzung	EEP 10.2 Plugin 2	
Zweck	EEP schreibt in diese Variable die Versionsnummer von EEP.	

```
if EEPVer < 11 then
    print("Zugbeeinflussung per Lua nicht möglich!")
end
```

EEPTime		EEPTime
Typ	Variable	
Verwendung in	Skript	
Zuweisung durch	EEP	
Voraussetzung	EEP 10.2 Plugin 2	
Zweck	EEP schreibt in diese Variable die aktuelle Zeit. Der Wert entspricht den seit Mitternacht (EEP-Zeit) vergangenen Sekunden.	

```
if EEPTime == alteZeit + 50 then
    print("Es sind genau 50 Sekunden vergangen")
    alteZeit = EEPTime
elseif EEPTime > alte Zeit + 50 then
    print("Es sind mehr als 50 Sekunden vergangen")
    alteZeit = EEPTime
else
    print("Es sind noch keine 50 Sekunden vergangen")
end
```

EEPTimeH		EEPTimeH
Typ	Variable	`print("Es ist jetzt: ", EEPTimeH, ":", EEPTimeM, " Uhr")`
Verwendung in	Skript	
Zuweisung durch	EEP	
Voraussetzung	EEP 10.2 Plugin 2	
Zweck	EEP schreibt in diese Variable die aktuelle EEP-Stunde.	

EEPTimeM		EEPTimeM
Typ	Variable	`print("Es ist jetzt: ", EEPTimeH, ":", EEPTimeM, " Uhr")`
Verwendung in	Skript	
Zuweisung durch	EEP	
Voraussetzung	EEP 10.2 Plugin 2	
Zweck	EEP schreibt in diese Variable die aktuelle EEP-Minute.	

EEPTimeS		EEPTimeS

Typ	Variable	`if EEPTimeS == 15 then`
		`  EEPSetSignal(1, 1)  -- schalte Ampel 1 auf Grün`
		`elseif EEPTimeS == 45 then`
		`  EEPSetSignal(1, 2)  -- schalte Ampel 1 auf Rot`
		`end`

Verwendung in	Skript	
Zuweisung durch	EEP	
Voraussetzung	EEP 10.2 Plugin 2	
Zweck	EEP schreibt in diese Variable die aktuelle EEP-Sekunde.	

System-Funktionen

clearlog()	clearlog()
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	keine
Rückgabewerte	keine
Voraussetzung	EEP 10.2 Plugin 2
Zweck	Löscht den Inhalt des Ereignisfensters

```
clearlog()
```

print()	print("Text1" , "Text2" , ... , TextN)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	mehrere
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2
Zweck	Gibt das, was in den Klammern steht, im Ereignisfenster als Text aus.
Bemerkungen	• Alle Typen werden automatisch in Text umgewandelt. • Es können mehrere Parameter mitgegeben werden. Sie müssen durch ein Komma getrennt sein. • Rückgabewert ist der komplette, ausgegebene String.

```
print("Es ist jetzt: ", EEPTimeH, ":", EEPTimeM, " Uhr")
```

EEPMain()	EEPMain()
Typ	Funktion
Aufruf durch	EEP
Definiert in	Skript
Parameter	keine
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2
Zweck	Wird zyklisch alle 200 Millisekunden, also fünf Mal je Sekunde, von EEP aufgerufen. Geeignet für alle Aktionen, die regelmäßig ausgeführt werden sollen.
Bemerkungen	• Muss im Skript deklariert sein, sonst stellt EEP die Verbindung zu Lua nicht her. • Der Funktionsaufruf durch EEP erfolgt ohne Parameter. • Die Funktion muss eine Zahl ungleich Null zurück liefern. • Liefert die Funktion den Wert 0 zurück, dann wird die Funktion nicht erneut aufgerufen. Alle anderen Funktionsaufrufe funktionieren weiterhin. • Fehlt der Rückgabewert oder ist er keine Zahl, dann erfolgt eine Fehlermeldung und die Verbindung zu Lua wird beendet.

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

Signal-Funktionen

EEPSetSignal()	EEPSetSignal(ID , Stellung , Callback)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei oder drei
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2
Zweck	Schaltet ein Signal
Bemerkungen	• Das erste Argument ist die Signal-ID. • Das zweite Argument ist die gewünschte Signalstellung. • Eine 1 als drittes (optionales) Argument bewirkt, dass die für dieses Signal definierte Funktion EEPOnSignal_x() aufgerufen wird. **Bitte mit Bedacht einsetzen!** Das Signal muss für EEPOnSignal_x() registriert und die Funktion definiert sein. Außerdem besteht die Gefahr, dass man sich bei unbedachtem Einsatz Programmschleifen einhandelt, die EEP und Lua lahm legen. • Rückgabewert ist 1 wenn das Signal und die gewünschte Signalstellung existieren oder 0, wenn eins von beidem nicht existiert.

```
-- stell Signal 0023 auf 1 (kann Fahrt oder Halt sein)
EEPSetSignal(23, 1)

-- stell Signal 0045 auf 1 und ruf EEPOnSignal_45() auf
EEPSetSignal(45, 1, 1)
```

EEPGetSignal()	EEPGetSignal(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2
Zweck	Ermittelt die Stellung eines Signals
Bemerkungen	• Das Argument ist die ID des Signals, dessen Stellung man ermitteln möchte. • Rückgabewert ist die Signalstellung. Die Nummer entspricht der Position dieser Signalstellung in der Auswahlliste unter den Signaleigenschaften. • Wenn das abgefragte Signal nicht existiert, ist der Rückgabewert 0.

```
Signalbild = EEPGetSignal(1)
if Signalbild == 0 then
    print("Signal 1 existiert nicht")
elseif Signalbild == 1 then
    print("Signal 1 steht auf Halt")
elseif Signalbild == 2 then
    print("Signal 1 steht auf Fahrt")
end
```

EEPRegisterSignal()	EEPRegisterSignal(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2

```
EEPRegisterSignal(1)

function EEPOnSignal_1(Signalbild)
    print("Signal 1 auf ", Signalbild, " gestellt")
end
```

Zweck	Registriert ein Signal für die Callback-Funktion EEPOnSignal_x() Diese notwendige Registrierung soll verhindern, dass Signale die Callback-Funktion aufrufen, für die keine entsprechende Funktion im Skript definiert wurde.
Bemerkungen	• Die Registrierung eines Signals ist zwingend erforderlich, damit es bei Schaltvorgängen selbständig die Funktion EEPOnSignal_x() aufruft. • Das Argument ist die Signal-ID. • Rückgabewert ist 1, wenn das zu registrierende Signal existiert oder 0, wenn es nicht existiert.

EEPOnSignal_x()	EEPOnSignal_x(Stellung)
Typ	Funktion
Aufruf durch	EEP
Definiert in	Skript
Parameter	einer
Rückgabewerte	keiner
Voraussetzung	EEP 10.2 Plugin 2

```
EEPRegisterSignal(1)

function EEPOnSignal_1(Signalbild)
    print("Signal 1 auf ", Signalbild, " gestellt")
end
```

Zweck	Registrierte Signale rufen selbständig diese Funktion auf, wenn sich ihre Stellung ändert. Im Skript definiert man die zugehörige Funktion und legt so fest, was bei Änderung der Signalstellung zu tun ist.
Bemerkungen	• Der Name der Funktion darf nicht mit _x enden, wie oben geschrieben, sondern muss mit der Signal-ID enden. Für Signal 0012 muss die Funktion also EEPOnSignal_12() heißen! **Bitte beachten:** Die führenden Nullen dürfen nicht im Funktionsnamen stehen! • Das Argument ist die neue Signalstellung als Zahl, entsprechend der Position dieser Signalstellung in der Auswahlliste der Signal-Eigenschaften. Eine selbst definierte Variable in den Funktionsklammern nimmt diesen Wert für die weitere Verwendung auf. • EEP erwartet bei Aufruf dieser Funktion keinen Rückgabewert.

Weichen-Funktionen

EEPSetSwitch()	EEPSetSwitch(ID , Stellung , Callback)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei oder drei
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2

```
-- stell Weiche 0067 auf 1 (Fahrt)
EEPSetSwitch(67, 1)

-- stell Weiche 0089 auf 1 und ruf EEPOnSwitch_89() auf
EEPSetSwitch(89, 1, 1)
```

Zweck	Schaltet eine Weiche
Bemerkungen	• Das erste Argument ist die Weichen-ID. • Das zweite Argument ist die gewünschte Weichenstellung. • Eine 1 als drittes (optionales) Argument bewirkt, dass die für diese Weiche definierte Funktion EEPOnSwitch_x() aufgerufen wird. **Bitte mit Bedacht einsetzen!** Die Weiche muss für EEPOnSwitch_x() registriert und die Funktion definiert sein. Außerdem besteht die Gefahr, dass man sich bei unbedachtem Einsatz Programmschleifen einhandelt, die EEP und Lua lahm legen. • Rückgabewert ist 1 wenn die Weiche und die gewünschte Weichenstellung existieren oder 0, wenn eins von beidem nicht existiert.

EEPGetSwitch()	EEPGetSwitch(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2

```
Weichenstellung = EEPGetSwitch(1)
if Weichenstellung == 0 then
    print("Weiche 1 existiert nicht")
elseif Weichenstellung == 1 then
    print("Weiche 1 steht auf Fahrt")
elseif Weichenstellung == 2 then
    print("Weiche 1 steht auf Abzweig")
end
```

Zweck	Ermittelt die Stellung einer Weiche
Bemerkungen	• Das Argument ist die ID der Weiche, deren Stellung man ermitteln möchte. • Rückgabewert ist die Weichenstellung. Die Nummer entspricht der Position dieser Weichenstellung in der Auswahlliste unter den Eigenschaften. • Wenn die abgefragte Weiche nicht existiert, ist der Rückgabewert 0.

EEPRegisterSwitch()	EEPRegisterSwitch(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	einer
Voraussetzung	EEP 10.2 Plugin 2
Zweck	Registriert eine Weiche für die Callback-Funktion EEPOnSwitch_x() Diese notwendige Registrierung soll verhindern, dass Weichen die Callback-Funktion aufrufen, für die keine entsprechende Funktion im Skript definiert wurde.
Bemerkungen	• Die Registrierung einer Weiche ist zwingend erforderlich, damit sie bei Schaltvorgängen selbständig die Funktion EEPOnSwitch_x() aufruft. • Das Argument ist die ID der Weiche. • Rückgabewert ist 1, wenn die zu registrierende Weiche existiert oder 0, wenn sie nicht existiert.

```
EEPRegisterSwitch(1)

function EEPOnSwitch_1(Stellung)
    print("Weiche 1 auf ", Stellung, " gestellt")
end
```

EEPOnSwitch_x()	EEPOnSwitch_x(Stellung)
Typ	Funktion
Aufruf durch	EEP
Definiert in	Skript
Parameter	einer
Rückgabewerte	keiner
Voraussetzung	EEP 10.2 Plugin 2
Zweck	Registrierte Weichen rufen selbständig diese Funktion auf, wenn sich ihre Stellung ändert. Im Skript definiert man die zugehörige Funktion und legt so fest, was bei Änderung der Weichenstellung zu tun ist.
Bemerkungen	• Der Name der Funktion darf nicht mit _x enden, wie oben geschrieben, sondern muss mit der Weichen-ID enden. Für Weiche 0034 muss die Funktion also EEPOnSwitch_34() heißen! **Bitte beachten:** Die führenden Nullen dürfen nicht im Funktionsnamen stehen! • Das Argument ist die neue Weichenstellung als Zahl, entsprechend der Position dieser Weichenstellung in der Auswahlliste der Eigenschaften. Eine selbst definierte Variable in den Funktionsklammern nimmt diesen Wert für die weitere Verwendung auf. • EEP erwartet bei Aufruf dieser Funktion keinen Rückgabewert.

```
EEPRegisterSwitch(1)

function EEPOnSwitch_1(Stellung)
    print("Weiche 1 auf ", Stellung, " gestellt")
end
```

Speicher-Funktionen

EEPSaveData()		EEPSaveData(Slot , Boolean|Zahl|"String"| nil)

Typ	Funktion	`-- speichert "wahr" in Slot 1`
		`EEPSaveData(1, true)`
		
		`-- speichert die Zahl 42 in Slot 2`
		`EEPSaveData(2, 42)`
		
		`-- speichert den String "Ich bin Slot 3" in Slot 3`
		`EEPSaveData(3, "Ich bin Slot 3")`
		
		`-- löscht den Inhalt von Slot 4`
		`EEPSaveData(4, nil)`

Aufruf durch	Skript	
Definiert in	EEP	
Parameter	zwei	
Rückgabewerte	einer	
Voraussetzung	EEP 11.0	
Zweck	Speichert etwas in einem speziellen Speicherbereich ab. Wird automatisch zusammen mit der Anlage gespeichert und geladen. Ersatz für die klassischen Hilfssignale.	
Bemerkungen	• Es gibt 1000 Speicherplätze, durchnummeriert von 1 bis 1000. • Man kann entweder Booleans, Zahlen oder Zeichenketten ("Strings") speichern, wobei letztere keine Formatierungszeichen enthalten dürfen. • Das erste Argument ist die Nummer des Speicherplatzes • Das zweite Argument ist der zu speichernde Inhalt. Mit nil kann der Speicher gelöscht werden. • Rückgabewert ist true bei erfolgreicher Speicherung oder false bei Misserfolg. • Wenn die Anlage gespeichert wird, dann hängt EEP selbständig die Inhalte dieser Speicherplätze an das zugehörige Skript an. Dieser Teil des Skripts hat keinen Einfluss auf die Prüfsumme, welche in der Anlagendatei gespeichert wird. Sie können daher auch mit einem externen Editor bearbeitet werden.	

EEPLoadData()		EEPLoadData(Slot)
Typ	Funktion	

Aufruf durch	Skript	`hResult, hData = EEPLoadData(1)`
		
		`if hResult then`
		`  print("Slot 1 enthält: "..hData)`
		`else`
		`  print("Slot 1 ist leer")`
		`end`

Definiert in	EEP	
Parameter	einer	
Rückgabewerte	zwei	
Voraussetzung	EEP 11.0	
Zweck	Lädt etwas aus einem speziellen Speicherbereich. Wird automatisch zusammen mit der Anlage gespeichert und geladen. Ersatz für die klassischen Hilfssignale.	
Bemerkungen	• Es gibt 1000 Speicherplätze, durchnummeriert von 1 bis 1000. • Man kann entweder Zahlen oder Zeichenketten ("Strings") speichern, wobei letztere keine Formatierungszeichen enthalten dürfen. • Das Argument ist die Nummer des Speicherplatzes. • Der erste Rückgabewert ist true, wenn der betroffene Speicher einen Inhalt hat oder false, wenn er leer ist. • Der zweite Rückgabewert ist der Inhalt des Speichers. • Wenn die Anlage geladen wird, dann holt EEP selbständig alle Inhalte dieser Speicherplätze aus dem Anhang des Skripts. Damit können sie bei Bedarf durch Aufruf von EEPLoadData() abgefragt und Variablen zugewiesen werden.	

Zug-Funktionen

EEPSetTrainSpeed()	EEPSetTrainSpeed("#Name" , Geschwindigkeit)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.0
Zweck	Weist einem Zugverband eine Soll-Geschwindigkeit zu.
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist die Geschwindigkeit. Ein negativer Wert bewirkt Rückwärts-Fahrt. • Die aktuelle Signalbeeinflussung wird aufgehoben. • Rückgabewert ist entweder true, wenn der angesprochene Zug existiert oder false, wenn er nicht existiert.

```
EEPSetTrainSpeed("#Personenzug", 80)
```

EEPGetTrainSpeed()	EEPGetTrainSpeed("#Name")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.0
Zweck	Ermittelt die Ist-Geschwindigkeit eines Zugverbandes.
Bemerkungen	• Das Argument ist der komplette Zugname als String. • Der erste Rückgabewert ist entweder true, wenn der angesprochene Zug existiert oder false, wenn er nicht existiert. • Der zweite Rückgabewert ist die ermittelte Geschwindigkeit.

```
hResult, hData = EEPGetTrainSpeed("#VT98;001")
```

EEPSetTrainRoute()		EEPSetTrainRoute("#Name" , "Route")
Typ	Funktion	EEPSetTrainRoute("#Personenzug", "Route")
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	zwei	
Rückgabewerte	einer	
Voraussetzung	EEP 11.2 Plugin 2	
Zweck	Weist einem Zugverband eine Route zu.	
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist die Route als String. • Rückgabewert ist entweder true, wenn der angesprochene Zug und die gewünschte Route existieren oder false, wenn eins von beidem nicht existiert.	

EEPGetTrainRoute()		EEPGetTrainRoute("#Name")
Typ	Funktion	hResult, hData = EEPGetTrainRoute("#Personenzug")
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	einer	
Rückgabewerte	zwei	
Voraussetzung	EEP 11.2 Plugin 2	
Zweck	Ermittelt die Route eines Zugverbandes.	
Bemerkungen	• Das Argument ist der komplette Zugname als String. • Der erste Rückgabewert ist true, wenn der angesprochene Zug existiert oder false, wenn er nicht existiert. • Der zweite Rückgabewert ist die ermittelte Route.	

EEPSetTrainLight()	EEPSetTrainLight("#Name" , true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.2 Plugin 2
Zweck	Schaltet an einem Zugverband das Licht ein oder aus.
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist entweder true (= Licht ein) oder false (= Licht aus). • Rückgabewert ist true, wenn der angesprochene Zug existiert. Sonst false.

```
EEPSetTrainLight("#Personenzug", true)
```

EEPSetTrainSmoke()	EEPSetTrainSmoke("#Name" , true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.2 Plugin 2
Zweck	Schaltet an einem Zugverband den Rauch an oder aus.
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist entweder true (= Rauch an) oder false (= Rauch aus). • Rückgabewert ist true, wenn der angesprochene Zug existiert. Sonst false.

```
EEPSetTrainSmoke("#Personenzug", true)
```

EEPSetTrainHorn()	EEPSetTrainHorn("#Name" , true)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.2 Plugin 2
Zweck	Lässt bei einem Zugverband den Warnton (Pfeife, Hupe) ertönen.
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist true um den Warnton ertönen zu lassen oder false um ihn abzustellen bevor er verklungen ist. • Rückgabewert ist true, wenn der angesprochene Zug existiert. Sonst false.

```
EEPSetTrainHorn("#Personenzug", true)
```

EEPSetTrainCouplingFront()	EEPSetTrainCouplingFront("#Name" , true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.2 Plugin 2
Zweck	Schaltet bei einem Zugverband die vordere Kupplung auf Kuppeln oder Abstoßen.
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist true (= Kuppeln) oder false (= Abstoßen). • Rückgabewert ist true, wenn der angesprochene Zug existiert. Sonst false.

```
EEPSetTrainCouplingFront("#Gueterzug", true)
```

EEPSetTrainCouplingRear()	EEPSetTrainCouplingRear("#Name" , true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.2 Plugin 2
Zweck	Schaltet bei einem Zugverband die hintere Kupplung auf Kuppeln oder Abstoßen.
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist true (= Kuppeln) oder false (= Abstoßen). • Rückgabewert ist true, wenn der angesprochene Zug existiert. Sonst false.

```
EEPSetTrainCouplingRear("#Gueterzug", true)
```

EEPTrainLooseCoupling()	EEPTrainLooseCoupling("#Name",true|false,Stelle)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	drei
Rückgabewerte	einer
Voraussetzung	EEP 11.2 Plugin 2
Zweck	Trennt einen Zugverband an der angegebenen Stelle.
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument bestimmt, ob von vorne oder hinten gezählt wird. (true = vorne, false = hinten) • Das dritte Argument bestimmt die Position, an der abgekuppelt wird. • Rückgabewert ist true, wenn der angesprochene Zug existiert. Sonst false.

```
EEPTrainLooseCoupling("#Gueterzug", true, 3)
```

EEPSetTrainHook()		EEPSetTrainHook("#Name" , true|false)
Typ	Funktion	EEPSetTrainHook("#Kranzug", true)
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	zwei	
Rückgabewerte	einer	
Voraussetzung	EEP 11.2 Plugin 2	
Zweck	Schaltet an einem Zugverband den Haken für Güter an oder aus.	
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist entweder true (= an) oder false (= aus). • Rückgabewert ist true, wenn der angesprochene Zug existiert. Sonst false.	

EEPSetTrainAxis()		EEPSetTrainAxis("#Name" , "Achse" , Stellung)
Typ	Funktion	EEPSetTrainAxis("#Kranzug", "Ausleger heben/senken", 100)
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	drei	
Rückgabewerte	einer	
Voraussetzung	EEP 11.2 Plugin 2	
Zweck	Animiert bei einem Zugverband eine ausgewählte Achse.	
Bemerkungen	• Das erste Argument ist der komplette Zugname als String. • Das zweite Argument ist der Name der Achse als String. • Das dritte Argument ist die Position, zu der sich die Achse bewegen soll. • Rückgabewert ist true, wenn der angesprochene Zug und die angesprochene Achse existieren. Sonst false.	

Rollmaterial-Funktionen

EEPRollingstockSetCouplingFront()	EEPRollingstockSetCouplingFront("Name" , Kupplung)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.0
Zweck	Stellt die vordere Kupplung eines Rollmaterials um.
Bemerkungen	• Das erste Argument ist der komplette Name des Rollmaterials als String. • Das zweite Argument ist der gewünschte Kupplungszustand. 1 bedeutet Kuppeln 2 bedeutet Abstoßen • Der Rückgabewert ist entweder true, wenn das angesprochene Rollmaterial existiert oder false, wenn es nicht existiert.

```
EEPRollingstockSetCouplingFront("Castor 1;001", 1)
```

EEPRollingstockGetCouplingFront()	EEPRollingstockGetCouplingFront("Name")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.0
Zweck	Ermittelt die Stellung der vorderen Kupplung eines Rollmaterials.
Bemerkungen	• Das Argument ist der komplette Name des Rollmaterials als String. • Der erste Rückgabewert ist entweder true, wenn das angesprochene Rollmaterial existiert oder false, wenn es nicht existiert. • Der zweite Rückgabewert ist die Stellung der Kupplung. 1 bedeutet Kupplung scharf 2 bedeutet Abstoßen 3 bedeutet Gekuppelt

```
hResult, hData =
EEPRollingstockGetCouplingFront("Castor 1;001")
```

EEPRollingstockSetCouplingRear()	EEPRollingstockSetCouplingRear("Name" , Kupplung)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.0
Zweck	Stellt die hintere Kupplung eines Rollmaterials um.
Bemerkungen	• Das erste Argument ist der komplette Name des Rollmaterials als String. • Das zweite Argument ist der gewünschte Kupplungszustand. 1 bedeutet Kuppeln 2 bedeutet Abstoßen • Der Rückgabewert ist entweder true, wenn das angesprochene Rollmaterial existiert oder false, wenn es nicht existiert.

```
EEPRollingstockSetCouplingRear("fals 175 Kalk", 1)
```

EEPRollingstockGetCouplingRear()	EEPRollingstockGetCouplingRear("Name")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.0
Zweck	Ermittelt die Stellung der hinteren Kupplung eines Rollmaterials.
Bemerkungen	• Das Argument ist der komplette Name des Rollmaterials als String. • Der erste Rückgabewert ist entweder true, wenn das angesprochene Rollmaterial existiert oder false, wenn es nicht existiert. • Der zweite Rückgabewert ist die Stellung der Kupplung. 1 bedeutet Kupplung scharf 2 bedeutet Abstoßen 3 bedeutet Gekuppelt

```
Name = "fals 175 Kalk"
hResult, hData = EEPRollingstockGetCouplingRear(Name)
```

EEPRollingstockSetAxis()		EEPRollingstockSetAxis("Name" , "Achse" , Stellung)
Typ	Funktion	
Aufruf durch	Skript	
Definiert in	EEP	Name = "Bekohlungskranbrücke 1" Achse = "Drehung links"
Parameter	drei	EEPRollingstockSetAxis(Name, Achse, 50)
Rückgabewerte	einer	
Voraussetzung	EEP 11.0	
Zweck	Bewegt die benannte Achse des benannten Rollmaterials in eine gewünschte Position.	
Bemerkungen	• Das erste Argument ist der komplette Name des Rollmaterials als String. • Das zweite Argument ist der komplette Name der zu bewegenden Achse als String. • Das dritte Argument ist die Position, zu der sich die Achse bewegen soll. • Rückgabewert ist true, wenn Rollmaterial und Achse existieren oder false, falls mindestens eins von beidem nicht existiert.	

EEPRollingstockGetAxis()		EEPRollingstockGetAxis("Name" , "Achse")
Typ	Funktion	
Aufruf durch	Skript	
Definiert in	EEP	Name = "Bekohlungskranbrücke 1" Achse = "Drehung links"
Parameter	zwei	EEPRollingstockGetAxis(Name, Achse)
Rückgabewerte	zwei	
Voraussetzung	EEP 11.0	
Zweck	Ermittelt die aktuelle Position einer benannten Achse des benannten Rollmaterials.	
Bemerkungen	• Das erste Argument ist der komplette Name des Rollmaterials als String. • Das zweite Argument ist der komplette Name der Achse als String. • Der erste Rückgabewert ist true, wenn Rollmaterial und Achse existieren oder false, falls mindestens eins von beidem nicht existiert. • Der zweite Rückgabewert ist die momentane Position der Achse als Zahl.	

EEPRollingstockSetSlot()	EEPRollingstockSetSlot("Name" , Slot)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.0
Zweck	Bewegt alle Achsen des genannten Rollmaterials zu der im Slot gespeicherten Position
Bemerkungen	• Für das Rollmaterial müssen zuvor Kombinationen von Achsstellungen in einem Slot gespeichert werden. Mit Aufruf dieser Funktion bewegen sich dann alle Achsen von ihrer augenblicklichen zur im Slot gespeicherten Position. • Das erste Argument ist der komplette Name des Rollmaterials als String. • Das zweite Argument ist die Nummer des Slots, in dem die gewünschten Achsstellungen gespeichert sind. • Rückgabewert ist true, wenn das Rollmaterial und die Slot-ID existieren oder false, wenn mindestens eins von beidem nicht existiert. Es wird nicht geprüft, ob im Slot tatsächlich etwas gespeichert ist.

```
EEPRollingstockSetSlot("Ladekran2 Greifer", 1)
```

Immobilien-Funktionen

EEPStructureSetSmoke()	EEPStructureSetSmoke("Lua-Name",true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Schaltet den Rauch der benannten Immobilie an oder aus.
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist entweder true, wenn man den Rauch anschalten möchte oder false, wenn man den Rauch ausschalten möchte. • Rückgabewert ist entweder true, wenn die Immobilie existiert und eine Rauchfunktion hat oder false, falls mindestens eins von beidem nicht zutrifft.

```
EEPStructureSetSmoke("#1_Abfertigung Lauscha", true)
```

EEPStructureGetSmoke()	EEPStructureGetSmoke("Lua-Name")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Ermittelt, ob der Rauch der benannten Immobilie an- oder ausgeschaltet ist.
Bemerkungen	• Das Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Der erste Rückgabewert ist entweder true, wenn die Immobilie existiert und eine Rauchfunktion hat oder false, falls mindestens eins von beidem nicht zutrifft. • Der zweite Rückgabewert ist entweder true, wenn der Rauch an- oder false, wenn der Rauch ausgeschaltet ist.

```
Name = "#1 Abfertigung Lauscha"

hResult, hData = EEPStructureGetSmoke(Name)
```

EEPStructureSetLight()	EEPStructureSetLight("Lua-Name",true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Schaltet das Licht der benannten Immobilie an oder aus.
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist entweder true, wenn man das Licht anschalten möchte oder false, wenn man das Licht ausschalten möchte. • Rückgabewert ist entweder true, wenn die Immobilie existiert und eine Lichtfunktion hat oder false, falls mindestens eins von beidem nicht zutrifft.

```
EEPStructureSetLight("#1_Betriebsdienstgebaeude", true)
```

EEPStructureGetLight()	EEPStructureGetLight("Lua-Name")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Ermittelt, ob das Licht der benannten Immobilie an- oder ausgeschaltet ist.
Bemerkungen	• Das Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Der erste Rückgabewert ist entweder true, wenn die Immobilie existiert und eine Lichtfunktion hat oder false, falls mindestens eins von beidem nicht zutrifft. • Der zweite Rückgabewert ist entweder true, wenn das Licht an- oder false, wenn das Licht ausgeschaltet oder im "Automatik"-Modus ist.

```
Name = "#1_Betriebsdienstgebaeude"

hResult, hData = EEPStructureGetLight(Name)
```

EEPStructureSetFire()	EEPStructureSetFire("Lua-Name",true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Schaltet das Feuer der benannten Immobilie an oder aus.
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist entweder true, wenn man das Feuer anschalten möchte oder false, wenn man das Feuer ausschalten möchte. • Rückgabewert ist entweder true, wenn die Immobilie existiert und eine Brandfunktion hat oder false, falls mindestens eins von beidem nicht zutrifft.

```
EEPStructureSetFire("#1_Brandhaus_01_SB1", true)
```

EEPStructureGetFire()	EEPStructureGetFire("Lua-Name")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Ermittelt, ob das Feuer der benannten Immobilie an- oder ausgeschaltet ist.
Bemerkungen	• Das Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Der erste Rückgabewert ist entweder true, wenn die Immobilie existiert und eine Brandfunktion hat oder false, falls mindestens eins von beidem nicht zutrifft. • Der zweite Rückgabewert ist entweder true, wenn das Feuer an- oder false, wenn das Feuer ausgeschaltet ist.

```
Name = "#1_Brandhaus_01_SB1"

hResult, hData = EEPStructureGetFire(Name)
```

EEPStructureAnimateAxis()	EEPStructureAnimateAxis("Lua-Name","Achse",Stellung)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	drei
Rückgabewerte	einer
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Bewegt die Achse einer Immobilie oder eines Gleisobjekts.
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist der Name der Achse als String • Das dritte Argument ist die (positive oder negative) Schrittzahl, um welche die Achse weiter bewegt werden soll. Der Wert 1000 bzw. -1000 bewirkt eine endlose Bewegung. Der Wert 0 stoppt die Bewegung. • Rückgabewert ist true, wenn Immobilie und Achse existieren oder false, falls mindestens eins von beidem nicht existiert.

```
EEPStructureAnimateAxis("#1_Windmühle", "Muehlrad", 1000)
```

EEPStructureSetAxis()	EEPStructureSetAxis("Lua-Name","Achse",Stellung)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	drei
Rückgabewerte	einer
Voraussetzung	EEP 11.1 Plugin 1
Zweck	Setzt die Achse einer Immobilie oder eines Gleisobjekts (ohne Animation).
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist der Name der Achse als String • Das dritte Argument ist die Position, zu der die Achse springen soll. • Rückgabewert ist true, wenn Immobilie und Achse existieren oder false, falls mindestens eins von beidem nicht existiert.

```
EEPStructureSetAxis("#1_Drehscheibe", "Brücke", 50)
```

EEPStructureGetAxis()		EEPStructureGetAxis("Lua-Name","Achse")
Typ	Funktion	hResult, hData = EEPStructureGetAxis("#1_Drehscheibe", "Brücke")
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	zwei	
Rückgabewerte	zwei	
Voraussetzung	EEP 11.1 Plugin 1	
Zweck	Ermittelt die Position einer Achse der benannten Immobilie oder des Gleisobjekts	
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist der Name der Achse als String • Der erste Rückgabewert ist true, wenn Immobilie und Achse existieren oder false, falls mindestens eins von beidem nicht existiert. • Der zweite Rückgabewert ist die momentane Position der Achse als Zahl.	

EEPStructureSetPosition()		EEPStructureSetPosition("Lua-Name",PosX,PosY,PosZ)
Typ	Funktion	EEPStructureSetPosition("#1_Strohballen", 1, 2, 3)
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	vier	
Rückgabewerte	einer	
Voraussetzung	EEP 11.1 Plugin 1	
Zweck	Versetzt die benannte Immobilie an eine neue Position.	
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist die Position in X-Richtung. • Das dritte Argument ist die Position in Y-Richtung. • Das vierte Argument ist die Position in Z-Richtung. • Eine Positionierung außerhalb des Anlagenbereichs ist nicht möglich • Rückgabewert ist true, wenn die Immobilie existiert oder false, wenn sie nicht existiert oder außerhalb des Anlagenbereichs positioniert werden sollte.	

EEPStructureSetRotation()		EEPStructureSetRotation("Lua-Name",RotX,RotY,RotZ)
Typ	Funktion	EEPStructureSetRotation("#1_Strohballen", 0, 0, 25)
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	vier	
Rückgabewerte	einer	
Voraussetzung	EEP 11.1 Plugin 1	
Zweck	Dreht die benannte Immobilie in eine neue Position.	
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie als String. Dieser unterscheidet sich durch die vorangestellte ID der Immobilie vom Modellnamen. • Das zweite Argument ist die Drehung in X-Richtung, • Das dritte Argument ist die Drehung in Y-Richtung, • Das vierte Argument ist die Drehung in Z-Richtung, • Rückgabewert ist true, wenn die Immobilie existiert oder false, wenn sie nicht existiert.	

Fahrweg-Funktionen

Eisenbahngleise

EEPRegisterRailTrack()		EEPRegisterRailTrack(ID)
Typ	Funktion	`EEPRegisterRailTrack(1)`
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	einer	
Rückgabewerte	einer	
Voraussetzung	EEP 11.3 Plugin 3	
Zweck	Registriert ein Gleiselement für Besetztabfragen.	
Bemerkungen	• Das Argument ist die ID des zu registrierenden Gleises. • Rückgabewert ist true, wenn das Gleis existiert, andernfalls false • Registrierung ist zwingende Voraussetzung für Besetztabfragen.	

EEPIsRailTrackReserved()		EEPIsRailTrackReserved(ID)
Typ	Funktion	`hResult, hData = EEPIsRailTrackReserved(1)`
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	einer	
Rückgabewerte	zwei	
Voraussetzung	EEP 11.3 Plugin 3	
Zweck	Ermittelt, ob ein Gleiselement besetzt ist.	
Bemerkungen	• Das Argument ist die ID des Gleises, welches auf "besetzt" geprüft werden soll. • Der erste Rückgabewert ist true, wenn das zu prüfende Gleis existiert und registriert ist, andernfalls false. • Der zweite Rückgabewert ist true, wenn das Gleis besetzt ist, andernfalls false. • Das Gleis muss zuvor für Besetztabfragen registriert sein!	

Straßen

EEPRegisterRoadTrack()	EEPRegisterRoadTrack(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	einer
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Registriert ein Straßenelement für Besetztabfragen.
Bemerkungen	• Das Argument ist die ID der zu registrierenden Straße. • Rückgabewert ist true, wenn die Straße existiert, andernfalls false • Registrierung ist zwingende Voraussetzung für Besetztabfragen.

```
EEPRegisterRoadTrack(1)
```

EEPIsRoadTrackReserved()	EEPIsRoadTrackReserved(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Ermittelt, ob ein Straßenelement besetzt ist.
Bemerkungen	• Das Argument ist die ID der Straße, welche auf "besetzt" geprüft werden soll. • Der erste Rückgabewert ist true, wenn die zu prüfende Straße existiert und registriert ist, andernfalls false. • Der zweite Rückgabewert ist true, wenn die Straße besetzt ist, andernfalls false. • Die Straße muss zuvor für Besetztabfragen registriert sein!

```
hResult, hData = EEPIsRoadTrackReserved(1)
```

Straßenbahngleise

EEPRegisterTramTrack()		EEPRegisterTramTrack(ID)
Typ	Funktion	`EEPRegisterTramTrack(1)`
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	einer	
Rückgabewerte	einer	
Voraussetzung	EEP 11.3 Plugin 3	
Zweck	Registriert ein Straßenbahngleis-Element für Besetztabfragen.	
Bemerkungen	• Das Argument ist die ID des zu registrierenden Gleises. • Rückgabewert ist true, wenn das Gleis existiert, andernfalls false • Registrierung ist zwingende Voraussetzung für Besetztabfragen.	

EEPIsTramTrackReserved()		EEPIsTramTrackReserved(ID)
Typ	Funktion	`hResult, hData = EEPIsTramTrackReserved(1)`
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	einer	
Rückgabewerte	zwei	
Voraussetzung	EEP 11.3 Plugin 3	
Zweck	Ermittelt, ob ein Straßenbahngleis-Element besetzt ist.	
Bemerkungen	• Das Argument ist die ID des Gleises, welche auf "besetzt" geprüft werden soll. • Der erste Rückgabewert ist true, wenn das zu prüfende Gleis existiert und registriert ist, andernfalls false. • Der zweite Rückgabewert ist true, wenn das Gleis besetzt ist, andernfalls false. • Die Straße muss zuvor für Besetztabfragen registriert sein!	

Wasserwege, sonstige Fahrwege

EEPRegisterAuxiliaryTrack()	EEPRegisterAuxiliaryTrack(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	einer
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Registriert einen Weg-Element der Kategorie "Sonstige" für Besetztabfragen.
Bemerkungen	• Das Argument ist die ID des zu registrierenden Weges. • Rückgabewert ist true, wenn der Weg existiert, andernfalls false • Registrierung ist zwingende Voraussetzung für Besetztabfragen.

```
EEPRegisterAuxiliaryTrack(1)
```

EEPIsAuxiliaryTrackReserved()	EEPIsAuxiliaryTrackReserved(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Ermittelt, ob ein Weg-Element der Kategorie "Sonstige" besetzt ist.
Bemerkungen	• Das Argument ist die ID des Weges, welcher auf "besetzt" geprüft werden soll. • Der erste Rückgabewert ist true, wenn der zu prüfende Weg existiert und registriert ist, andernfalls false. • Der zweite Rückgabewert ist true, wenn der Weg besetzt ist, andernfalls false. • Der Weg muss zuvor für Besetztabfragen registriert sein!

```
hResult, hData = EEPIsAuxiliaryTrackReserved(1)
```

Steuerstrecken

EEPRegisterControlTrack()	EEPRegisterControlTrack(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	einer
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Registriert ein Steuerstrecken-Element für Besetztabfragen.
Bemerkungen	• Das Argument ist die ID der zu registrierenden Steuerstrecke. • Rückgabewert ist true, wenn die Steuerstrecke existiert, andernfalls false • Registrierung ist zwingende Voraussetzung für Besetztabfragen.

```
EEPRegisterControlTrack(1)
```

EEPIsControlTrackReserved()	EEPIsControlTrackReserved(ID)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	einer
Rückgabewerte	zwei
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Ermittelt, ob ein Steuerstrecken-Element besetzt ist.
Bemerkungen	• Das Argument ist die ID der Steuerstrecke, welche auf "besetzt" geprüft werden soll. • Der erste Rückgabewert ist true, wenn die zu prüfende Steuerstrecke existiert und registriert ist, andernfalls false. • Der zweite Rückgabewert ist true, wenn die Steuerstrecke besetzt ist, andernfalls false. • Die Steuerstrecke muss zuvor für Besetztabfragen registriert sein!

```
hResult, hData = EEPIsControlTrackReserved(1)
```

Kamera-Funktionen

EEPSetCamera()	EEPSetCamera(Typ , "Name")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Wählt eine der gespeicherten Kameras aus der Liste.
Bemerkungen	• Das erste Argument ist der Kameratyp: 0=statisch, 1=dynamisch, 2= mobile Kamera • Das zweite Argument ist der Name der Kamera als String • Rückgabewert ist true, wenn die Kamera existiert, andernfalls false

```
EEPSetCamera(0,"Bahnhof")
```

EEPSetPerspectiveCamera()	EEPSetPerspectiveCamera(ID , "Zugname")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 11.3 Plugin 3
Zweck	Wählt eine der Verfolger-Kameras für den angegebenen Zug.
Bemerkungen	• Das erste Argument ist die Kameraposition - entspricht den Tasten 1 bis 9 für Kameraauswahl. • Das zweite Argument ist der Zugname als String • Der Rückgabewert ist true, wenn die gewünschte Kamera existiert, andernfalls false.

```
EEPSetPerspectiveCamera(1,"#Personenzug")
```

Anlagen-Funktionen

EEPLoadProject()		EEPLoadProject("Dateiname")
Typ	Funktion	EEPLoadProject("Tutorials\\Tutorial_54_LUA.anl3")
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	einer	
Rückgabewerte	einer	
Voraussetzung	EEP 11.3 Plugin 3	
Zweck	Lädt eine Anlage aus dem Ordner "Resourcen\Anlagen".	
Bemerkungen	• Das Argument ist der Unterordner (wenn erforderlich) und der Dateiname einschließlich .anl3 Suffix. Trennzeichen zwischen Ordner- und Dateiname ist ein doppelter Backslash. • Rückgabewert ist true, wenn die Anlage existiert, andernfalls false	

zurück zum Inhaltsverzeichnis

Zugdepot-Funktionen

EEPGetTrainFromTrainyard()	`EEPGetTrainFromTrainyard(Depot , "Zugname" , Nr)`
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	drei
Rückgabewerte	einer
Voraussetzung	EEP 11.3 Plugin 2
Zweck	Schickt einen ausgewählten Zug aus einem ausgewählten virtuellen Zugdepot
Bemerkungen	• Das erste Argument ist die ID des Zugdepots. Sie steht in der Kopfzeile des Eigenschaftenfensters. • Das zweite Argument ist der Name des Zuges als String. Wird der Name weggelassen, dann bestimmt das dritte Argument den Zug • Das dritte Argument ist der Listenplatz des Zuges im Depot. Dieses Argument gilt nur, wenn kein Zugname angegeben ist. Bei vorgegebenem Zugnamen ist diese Zahl beliebig, aber dennoch erforderlich. • Rückgabewert ist true, wenn das Depot und der angeforderte Zug existieren, andernfalls false.

```
EEPGetTrainFromTrainyard(1,"#Rheingold",1)
```

Tipp-Text Funktionen

EEPChangeInfoStructure()	EEPChangeInfoStructure("Lua-Name","Text")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 13
Zweck	Weist einem Tipp-Text einen neuen Text zu
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie. Er steht in den Objekteigenschaften. • Das zweite Argument ist der gewünschte Text. Zeilenumbruch mit \n • Rückgabewert ist true, wenn das Ziel der Funktion gefunden wurde.

```
EEPChangeInfoStructure("#1", "Hallo")
```

EEPShowInfoStructure()	EEPShowInfoStructure("Lua-Name",true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 13
Zweck	Schaltet einen Tipp-Text ein und aus
Bemerkungen	• Das erste Argument ist der Lua-Name der Immobilie. Er steht in den Objekteigenschaften. • Das zweite Argument ist entweder true (Tipp-Text ein) oder false (Tipp-Text aus). • Rückgabewert ist true, wenn das Ziel der Funktion gefunden wurde.

```
EEPShowInfoStructure("#1", true)
```

EEPChangeInfoSignal()		EEPChangeInfoSignal(ID,"Text")
Typ	Funktion	EEPChangeInfoSignal(1, "Hallo")
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	zwei	
Rückgabewerte	einer	
Voraussetzung	EEP 13	
Zweck	Weist einem Tipp-Text einen neuen Text zu	
Bemerkungen	• Das erste Argument ist die Signal-ID. • Das zweite Argument ist der gewünschte Text. Zeilenumbruch mit \n • Rückgabewert ist true, wenn das Ziel der Funktion gefunden wurde.	

EEPShowInfoSignal()		EEPShowInfoSignal(ID,true|false)
Typ	Funktion	EEPShowInfoSignal(1, true)
Aufruf durch	Skript	
Definiert in	EEP	
Parameter	zwei	
Rückgabewerte	einer	
Voraussetzung	EEP 13	
Zweck	Schaltet einen Tipp-Text ein und aus	
Bemerkungen	• Das erste Argument ist die Signal-ID • Das zweite Argument ist entweder true (Tipp-Text ein) oder false (Tipp-Text aus). • Rückgabewert ist true, wenn das Ziel der Funktion gefunden wurde.	

EEPChangeInfoSwitch()	EEPChangeInfoSwitch(ID,"Text")
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 13
Zweck	Weist einem Tipp-Text einen neuen Text zu
Bemerkungen	• Das erste Argument ist die Weichen-ID. • Das zweite Argument ist der gewünschte Text. Zeilenumbruch mit \n • Rückgabewert ist true, wenn das Ziel der Funktion gefunden wurde.

```
EEPChangeInfoSwitch(1, "Hallo")
```

EEPShowInfoSwitch()	EEPShowInfoSwitch(ID,true|false)
Typ	Funktion
Aufruf durch	Skript
Definiert in	EEP
Parameter	zwei
Rückgabewerte	einer
Voraussetzung	EEP 13
Zweck	Schaltet einen Tipp-Text ein und aus
Bemerkungen	• Das erste Argument ist die Weichen-ID • Das zweite Argument ist entweder true (Tipp-Text ein) oder false (Tipp-Text aus). • Rückgabewert ist true, wenn das Ziel der Funktion gefunden wurde.

```
EEPShowInfoSwitch(1, true)
```