LUA

DESCRIPTION ET MANUEL UTILISATEUR POUR EEP

Version 14.1

Table des matières

Informations générales

Le plug-in 2 de '***Railway X***' introduit une nouvelle fonctionnalité qui enrichit énormément les possibilités d'exploitation d'EEP. Il s'agit d'un langage de scripting, c'est-à-dire un langage de programmation qui est traduit en langage informatique lors de l'exécution du programme. Le nom de ce langage de scripting est '***Lua***' et il a été développé par la PUC Rio (Université Catholique Pontificale de Rio de Janeiro). Pour de plus amples informations, veuillez consulter le site http://de.wikipedia.org/wiki/Lua.

La version implémentée dans EEP est la version 5.2 et contient en outre un certain nombre de fonctions spécifiques qui peuvent établir une relation avec lui et l'influencer. Le nombre de fonctions spécifiques à EEP est sans cesse étendu. Dans EEP 10 avec le plugin 2, les aiguillages et les signaux peuvent être commandés par Lua. Avec EEP 11.0, la vitesse du train, la commande d'embrayage pour le matériel roulant et les emplacements de données pour le stockage permanent des valeurs ont été ajoutés. Le plugin 1 pour EEP 11 contenait des fonctions de contrôle pour les éléments immobiliers. Avec le Plugin 2, la gamme de fonctions s'est encore élargie. Les points de contact transmettent maintenant le nom du train déclencheur à chaque appel de fonction. De plus, il est désormais possible de gérer la fumée, la lumière et le klaxon d'avertissement d'un train ainsi que les couplages, les essieux et le cas échéant, le crochet du coupleur d'un train. Enfin, EEP 13 propose également l'édition de textes personnalisés.

Dans les pages suivantes, vous trouverez une description de l'éditeur de script et une description des fonctions spécifiques EEP implémentées avec Lua. Les exemples joints dans le tutoriel avec les scripts Lua sont également expliqués. L'ensemble des instructions du langage Lua est vaste et un développement complet dépasserait le cadre de ce manuel. Une documentation allemande, y compris la traduction du Manuel de référence original, est disponible sur le site Internet suivant : http://lua.coders-online.net

Nous vous encourageons à visiter ces sites ou d'autres sites Web et à vous familiariser avec les bases de ce langage de scripting.

1. Intégration dans EEP 14

Des interfaces dédiées ont été implémentées dans EEP, dans le but de créer un moyen d'échanger des données avec Lua. Cela signifie qu'EEP peut transmettre des informations à Lua et recevoir des commandes de Lua. Pour cela, nous avons créé des fonctions Lua spéciales. Elles commencent toutes par les lettres EEP.

EEP contient maintenant une fenêtre d'événements pour la sortie des messages (*Figure 1*) et une fenêtre d'édition pour la création des scripts (*Figure 2*).

Fig. 1

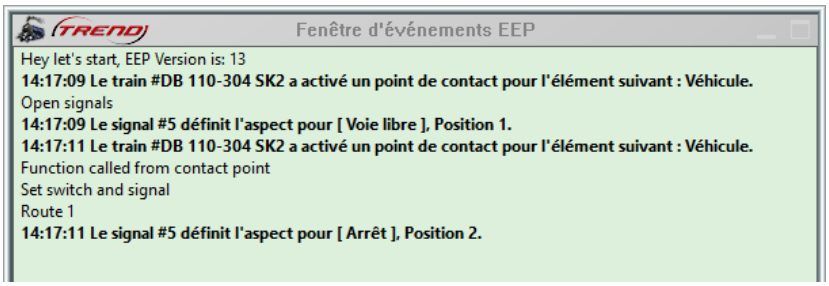
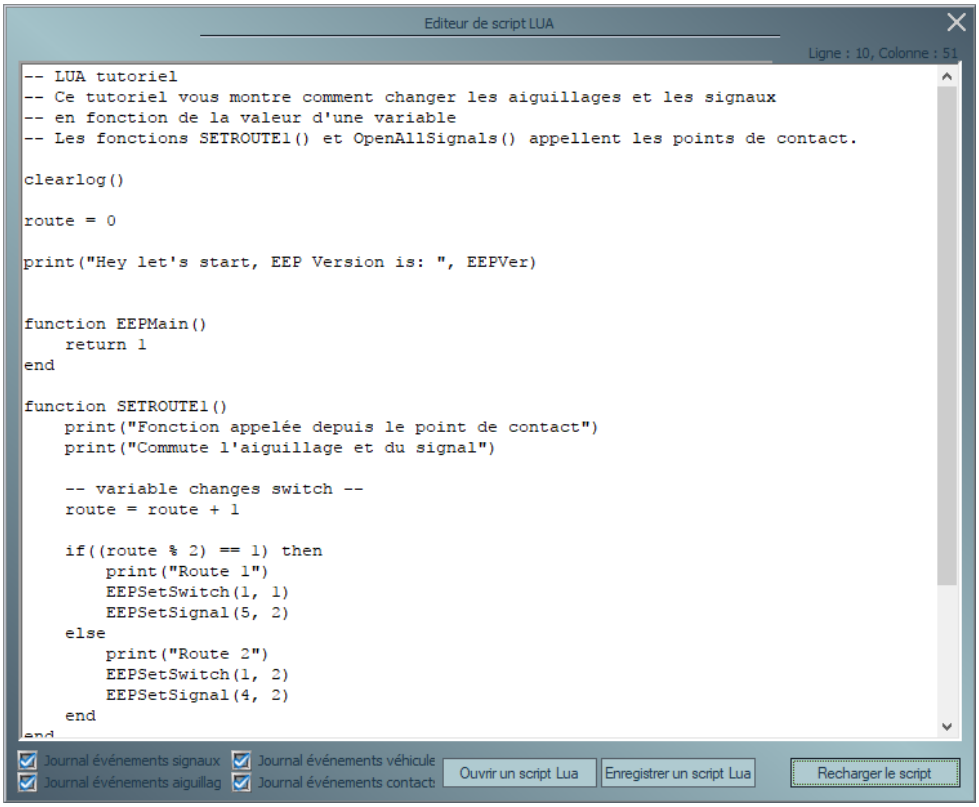
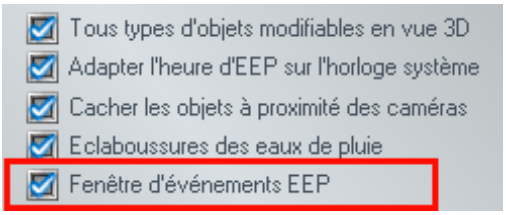


Fig. 2



Vous pouvez activer la fenêtre d'événement dans les paramètres du programme. (*Figure 3*)

Fig. 3



Ouvrez la fenêtre de script à l'aide du bouton dans la barre de menu. (*Figure 4*)

Fig. 4

La fenêtre d'événement peut afficher des messages système et des textes définis par l'utilisateur. Les messages système comprennent les messages d'état des aiguillages, signaux, trains et points de contact ainsi que les messages d'erreur de l'interpréteur Lua.

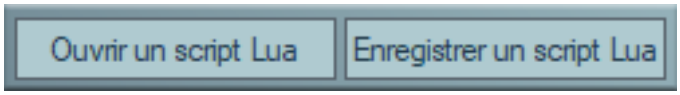
Pour imprimer votre propre texte, veuillez utiliser la fonction `print()`.

Les fichiers journaux des évènements peuvent être activés et désactivés dans la fenêtre de l'éditeur de script (*Figure 5*).

Fig. 5

L' éditeur est l'interface principale dans laquelle vous créez ou modifiez un script. Il contient également des boutons pour enregistrer, ouvrir et recharger le script courant. Un script est automatiquement sauvegardé avec votre projet. Lorsque vous l'ouvrez à nouveau, vous le retrouvez dans l'état où vous l'aviez laissé la dernière fois. Les boutons '***Ouvrir un script Lua***' et '***Enregistrer un script Lua***' (*Figure 6*) ne sont destinés uniquement que dans le cas où vous souhaitez ouvrir un script déjà existant ou l'enregistrer sur un support indépendant (par exemple, sauvegarde sur un disque dur externe ou une clé USB).

Fig. 6



Important : Un script modifié doit être rechargé.

Pour cela, vous devez toujours appuyer sur le bouton '***Recharger le script***' (*Figure 7*) après avoir créé un script ou apporté des modifications ! sinon vos informations seront perdues lorsque vous fermerez la fenêtre de script.

Fig. 7

Pour vos premiers pas dans le nouvel univers du scripting, vous trouverez quinze petits tutoriels de projets fonctionnels dans le dossier '***Resourcen -> Anlagen -> Tutorials***'. Chacun d'eux présente quelques fonctions dans un environnement simple. Cela vous donne la possibilité de comprendre exactement ce que fait chaque élément d'un script. Chacun de ces tutoriels est décrit plus en détail dans le chapitre `exemples inclus`.

2. Syntaxe du langage script Lua

Les langages de programmation sont des textes traduits en commandes informatiques. Pour les langages de script, un interpréteur exécute cette tâche. Celui-ci a besoin d'une certaine notation pour reconnaître ce que le programme doit faire. Cette notation est appelée syntaxe.

Comme tout langage de programmation, Lua requiert une précision syntaxique absolue. Un seul caractère manquant ou erroné entraîne une exécution erronée, voire un échec. Une distinction est faite entre les lettres majuscules et minuscules.

L'exemple suivant illustre à quel point vous devez être précis et rigoureux lors de la conception d'un script. Exemple : un commentaire est inséré dans l'éditeur Lua avec deux traits d'union consécutifs.

```
-- Ceci est un commentaire
```

Ce texte est ignoré par l'interpréteur. De tels commentaires sont des rappels utiles et peuvent également fournir des informations importantes aux autres utilisateurs qui lisent votre script. Si vous omettez accidentellement un des deux tirets, vous obtenez le message d'erreur suivant lors du rechargement du script (*Figure 8*) :

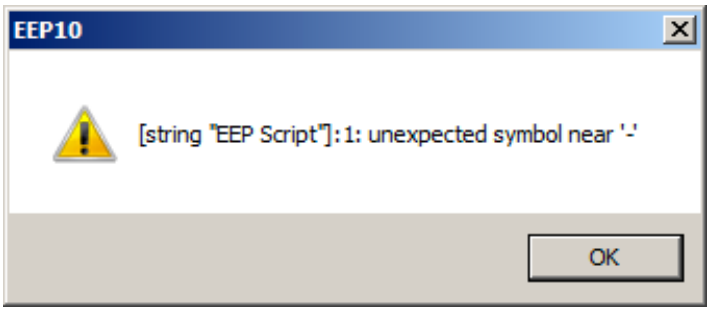


Fig. 8

Le message contient un certain nombre d'informations utiles :

- [string "EEP Script"] vous indique dans quel fichier l'erreur s'est produite,
- :1: est le numéro de ligne où l'erreur se situe,
- A côté du symbole '-' (Ce qui veut dire : caractère inattendu à côté de ' -'.) signifie que l'interpréteur a trouvé quelque chose qui n'est pas correct avant ou après le trait d'union.

Veuillez noter que l'interpréteur ne signale pas le deuxième trait d'union manquant. Un élément syntaxique de Lua est qu'un seul trait d'union est considéré comme le signe "moins". Lors de la traduction du script, l'interpréteur essaie de soustraire ce qui se trouve derrière la ligne. Par conséquent, il s'attend à ce qu'il y ait un nombre après le signe 'moins'. Mais ce qu'il trouve dans notre exemple est quelque chose de différent et donc inattendu. Un interpréteur n'est pas 'dans la tête' de l'auteur du script, mais ne peut que traduire rigoureusement ce qui est écrit dans le script.

Par conséquent, ce manuel, est la pour vous apprendre à ne pas commettre d'erreur de syn-

taxe. Nous allons vous montrer des exemples de création d'un script Lua, pas à pas, pour vous familiariser avec la syntaxe Lua.

L'exemple suivant vous montre l'importance des lettres majuscules et minuscules :

Lua dispose d'une fonction print(), qui affiche ce qui se trouve entre les parenthèses dans la fenêtre d'événement. La syntaxe correcte est :

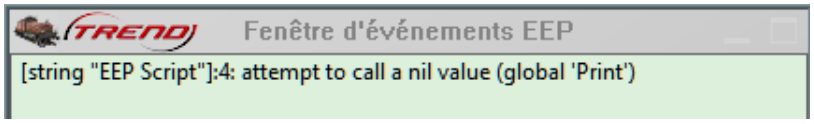
```
print("Bonjour, la version EEP est : ", EEPVer)
```

Ici, la syntaxe est incorrecte :

```
Print("Bonjour, la version EEP est : ", EEPVer)
```

La fonction `Print()` est considérée comme une fonction utilisateur et se distingue de la fonction `print()` propre à Lua. Par conséquent, le message d'erreur dans ce cas est le suivant :

Fig. 9



L'interpréteur a recherché dans le script une fonction appelée `Print()` et n'a rien trouvé. Dans la langage Lua, 'nil' veut dire 'rien'. Le message d'erreur indique donc que l'interpréteur n'a rien trouvé lors de la recherche.

Nous aimerions souligner une différence importante entre les deux messages d'erreur : le premier message d'erreur est affiché dans une petite fenêtre séparée, le deuxième message d'erreur a été écrit directement dans la fenêtre d'événement. La raison est liée au moment où l'erreur a été constatée par l'interpréteur.

Lorsqu'un script est analysé par l'interpréteur, celui-ci vérifie qu'il n'y a pas d'erreurs de syntaxe. Dans le cas contraire, l'interpréteur renvoie un message d'erreur dans une fenêtre séparée. La ligne avec un seul signe 'moins' au lieu de deux pour un commentaire était incorrecte, car il n'y avait rien avant le signe moins et donc une soustraction était également impossible. Les erreurs détectées lors de l'analyse du script par l'interpréteur sont signalées par EEP dans une fenêtre séparée.

Cependant, l'analyse de la fonction Print("Bonjour, la version EEP est : ", EEPVer) ne produit pas d'erreur car, le fait que cette fonction n'était pas encore définie au moment du rechargement du script n'est pas significatif dans ce contexte. Cette fonction pourrait être définie ailleurs ou ultérieurement. Par conséquent, l'erreur ne peut pas être détectée tant que le script ne s'exécute pas. Les erreurs détectées lors de l'exécution du script sont signalées par EEP dans la fenêtre d'événement.

Veuillez garder à l'esprit que la fenêtre d'événement doit rester ouverte pour que Lua vous informe des erreurs qui ne sont pas détectables jusqu'à ce que le script soit exécuté.

Plus un script est important, plus il sera difficile de trouver des erreurs de syntaxe similaires. Par conséquent, nous vous recommandons de développer des scripts en petites parties et de procéder à des tests. Cela permet d'identifier plus facilement la cause d'une erreur.

3. Les commandes disponibles

Un langage de programmation se compose de mots-clés, d'opérateurs, de variables et de fonctions :

- Les mots-clés sont des mots réservés qui obligent l'interpréteur à faire quelque chose,
- Les opérateurs sont les caractères qui obligent l'interpréteur à calculer ou associer logiquement quelque chose,
- Les variables sont des espaces de stockage dont le contenu peut être modifié par le script,
- Les fonctions sont des ensembles d'instructions regroupées sous un seul nom.

Exemples de mots clés : `if, then, return, do, function, end`

Exemples d'opérateurs : `+, -, *, /, <, >`

Exemples de variables : `EEPVer, EEPTime`

Exemples de fonctions : `clearlog(), print(), EEPSetSignal()`

Les mots-clés et les opérateurs font tous partie du langage Lua. Les variables et les fonctions existent aussi au sein de Lua, mais également celles que nous avons créées pour que Lua puisse communiquer avec EEP. C'est la raison pour laquelle vous pouvez définir vos propres variables et fonctions.

3.1 Commandes générales du langage Lua

Une description complète de tous les éléments que comporte Lua dépasserait le cadre de ce manuel. C'est pourquoi nous vous renvoyons à des sites Internet qui contiennent des informations détaillées, des explications et des exemples d'application :

http://lua.coders-online.net
http://www.lua.org/docs.html

Cependant, nous avons inclus quelques informations importantes de Lua dans ce manuel. Elles se trouvent à l'annexe I et les descriptions des tutoriels joints contiennent des liens vers ces explications.

3.2 Variables et fonctions spécifiques à EEP

Une liste de toutes les variables et fonctions spécifiques à EEP figure à l'annexe II.

4. Exemples inclus dans ce manuel

Les exemples ne sont pas forcément spectaculaires et ne disposent que de quelques fonctions. Pour cette raison, ils sont particulièrement indiqués pour l'étude du langage de script Lua ainsi que la possibilité de les expérimenter. Nous vous recommandons d'étudier attentivement chacun des exemples.

Les événements sur les tutoriels joints sont simples à comprendre. Habituellement, une seule locomotive tourne en rond et ne fait guère plus de commuter un aiguillage ou un signal. Ce qui est passionnant au sujet des tutoriels, ne sont pas les évènements qui se produisent, mais les lignes du script Lua qui permettent cela.

Dans de nombreux cas, vous penserez que tout cela serait possible sans Lua. Mais n'allez pas considérer à tort l'inutilité de ces tutoriels. Il ne s'agit pas de démontrer que tout est plus facile avec Lua, mais de vous aider à comprendre son fonctionnement. Une fois que vous aurez acquis un certain niveau d'expérience pour ce langage, vous serez étonné de constater tout ce qu'il est possible de faire avec lui. Avec Lua, vous pouvez apporter de la puissance et de l'intelligence à vos projets et concevoir des commandes en quelques lignes qui, autrefois, nécessitaient des développements bien plus longs. De plus, avec un script vous pouvez centraliser toutes les opérations de contrôle pour votre système ferroviaire.

Ne craignez pas d'apporter des changements et d'observer les conséquences de ces changements. Vos propres expériences sont le moyen le plus formateur et le plus productif d'apprendre quelque chose. Et tant qu'ils n'écrasent pas les tutoriels livrés avec EEP, vous ne risquez rien.

Les tutoriels sont simples pour que les scripts associés soient petits et compréhensibles. Les modifications apportées dans le code ont des conséquences évidentes. N'ayez pas peur de faire des erreurs en modifiant les scripts. Bien au contraire, provoquer des erreurs, c'est le but premier des didacticiels. Vous pouvez apprendre beaucoup des erreurs. Les explications pour les tutoriels sont classées chronologiquement les unes après les autres. Par exemple, les explications comprises dans le chapitre '*Tutoriel_33_Lua_1*', sont reprises dans le chapitre '*Tutoriel_34_Lua_2*'.

A partir de la page suivante, rentrons maintenant dans le vif du sujet.

5. Tutoriels

5.1 Description du projet "Tutorial_33_LUA_1"

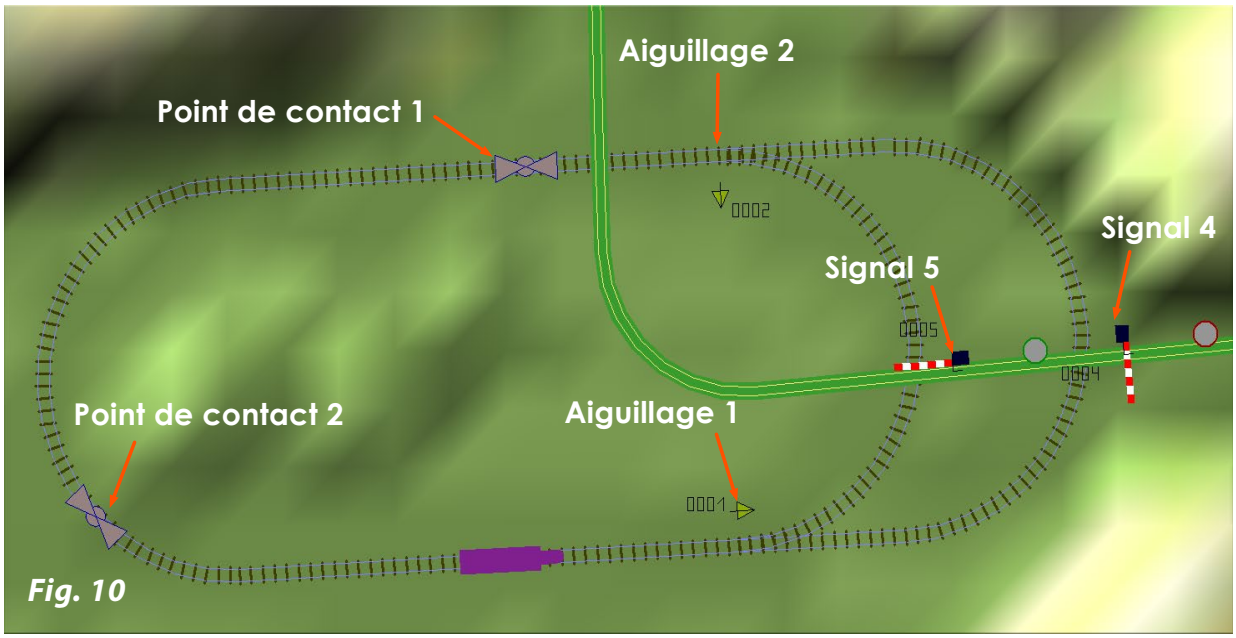


Fig. 10

Ce tutoriel vous initie au fonctionnement de base des passages à niveau via Lua. Un train circule en alternance sur l'ovale intérieur et extérieur, commutant les aiguillages et les barrières à l'aide des fonctions Lua suivantes :

EEPVer
EEPMain()
EEPSetSwitch()
EEPSetSignal()

Les explications de ce tutoriel seront beaucoup plus détaillées que les explications des tutoriels suivants, car il contient de nombreuses explications de base communes à tous les autres. Nous vous recommandons d'étudier ce premier tutoriel de manière approfondie. Il fournit des connaissances de base pour le contrôle d'un circuit ferroviaire au moyen de Lua. Il contient également de nombreuses suggestions pour vos propres expériences, qui devraient vous donner un aperçu de travailler avec Lua.

La vue 2D de cette installation (*Figure 10*) montre deux barrières, deux aiguillages et deux points de contact pour les véhicules. Lors de la mise en service du système, une seule locomotive roule en continu dans un des ovales. A chaque tour, elle change de parcours et ferme la barrière correspondante. Les barrières et les aiguillages sont commandés par les deux points de contact, mais pas de la manière traditionnelle. Uniquement par le fait que ces points de contact appellent des fonctions qui sont définies dans le script Lua.

Ouvrez l'éditeur de script (*Figure 4*) et regardez les quatre premières lignes du script. Il contient des commentaires qui sont ignorés par l'interpréteur :

```
-- Tutoriel LUA
-- Ce tutoriel explique comment commuter les aiguillages et les signaux
-- selon la valeur de la variable
-- Les fonctions SETROUTE1() et OpenAllSignals() appellent les points de
contact.
```

Vous pouvez utiliser les lignes de commentaire pour votre premier essai. Supprimez un des deux signes moins au début d'une ligne dans une ligne. Cliquez sur le bouton 'Recharger le script' (*Figure 7*) Une fenêtre s'ouvre et vous indique que votre script contient une erreur.

L'éditeur de script reste ouvert parce que l'interpréteur a refusé le script erroné. Corrigez l'erreur, recharger le script corrigé et passez en mode 3D pour observer les événements dans l'installation ferroviaire.

Vous pouvez également ouvrir l'éditeur de script en mode 3D. Ceci n'est pas conseillé, car la locomotive continuera à rouler, mais le script fera une pause. Essayez par vous-même. Ouvrez l'éditeur de script et observez le mouvement. La locomotive continuera de tourner comme avant, mais elle ne changera plus de direction et ne fera plus fonctionner les barrières.

Nous vous recommandons de passer à l'éditeur 2D si vous souhaitez apporter des modifications au script afin que le mouvement soit stoppé durant l'édition du script.

Regardez la ligne suivante du script :

```
clearlog()
```

les parenthèses après le mot clearlog() indiquent une `Fonction`. Elle est définie dans EEP et peut donc être exécutée immédiatement. La fonction `clearlog()` efface le contenu de la fenêtre d'événements.

Si vous le souhaitez, vous pouvez ajouter un commentaire à la ligne qui vous rappelle le but de cette fonction :

```
clearlog() -- Supprime le contenu de la fenêtre d'événement
```

Comme vous pouvez le voir dans cet exemple, les trémas et les caractères spéciaux sont aussi autorisés dans les commentaires. Tout ce qui se trouve derrière les deux signes moins est ignoré par l'interpréteur.

Regardez la ligne suivante :

```
route = 0
```

Cette ligne crée une `Variable` nommée `route` et lui assigne la valeur `0`. Cette variable peut être utilisée à plusieurs reprises dans le script. Si vous changez le nom de cette variable à un endroit quelconque du script et que vous appuyez sur le bouton 'Recharger le script', le script ne fonctionnera plus correctement sauf que l'interpréteur ne peut pas détecter immédiatement cette erreur lors du rechargement du script. Cette erreur se produira lorsque vous mettrez le réseau en service et que Lua arrivera à l'endroit où la variable est utilisée.

La ligne suivante du script s'interprète comme suit :

```
print("Bonjour, la version EEP est : ", EEPVer)
```

Les parenthèses après le mot print indiquent qu'il s'agit à nouveau d'une fonction. L'exemple print() montre le but des parenthèses à la fin d'une fonction. Elles sont utilisées pour 'passer' quelque chose (En informatique, le terme utilisé est : argument) à une fonction quand elle est appelée. Ici, il s'agit du texte à afficher par `print()` .

Pour voir le résultat de la fonction `print()`, la fenêtre d'événement (*Figure 1*) doit être ouverte (Figure 3). Le texte imprimé par `print()` n'apparaît pas dans la fenêtre de sortie tant que vous n'avez pas rechargé le script et que vous n'êtes pas passé en mode 3D.

Si vous regardez attentivement le texte de sortie dans la fenêtre d'événement, vous remarquerez que les guillemets n'ont pas été affichés. Une des syntaxes de Lua est que les guillemets sont utilisés comme délimiteurs pour les chaines de caractères. Vous écrivez votre texte entre guillemets afin que l'interpréteur puisse le distinguer des mots-clés, des variables et des noms de fonctions. Vous pouvez modifier le texte entre parenthèses comme bon vous semble et le remplacer par un texte en langue allemande, par exemple :

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

Veuillez noter que seule la première partie est comprise entre guillemets. Ensuite, se trouvent une virgule puis un mot qui n'est pas entre guillemets. La virgule est aussi un caractère de contrôle. Il informe l'interpréteur que la fonction `print()` reçoit un deuxième argument après le premier. Le deuxième argument est une variable système faisant partie de l'ensemble des variables et des fonctions internes à EEP afin que le programme puisse échanger des données avec Lua. Cette fonction est décrite à `l'annexe II`. EEPVer contient le numéro de version actuelle d'EEP.

Les trois lignes suivantes présentent la définition d'une fonction :

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

`EEPMain()` est une fonction spéciale appelée par EEP 5 fois par seconde (c'est-à-dire toutes les 200 millisecondes) utilisée pour toutes les actions nécessitant des répétitions constantes pendant un processus en cours d'exécution. La comparaison est équivalente au train qui tourne constamment en rond. Tout ce qui est à l'intérieur de la fonction, correspond aux points de contact dans le circuit.

Dans ce tutoriel, La fonction `EEPMain()` n'a pas de tâche particulière. Néanmoins, elle doit être définie, car EEP appelle cette fonction cinq fois par seconde. La fonction doit renvoyer à chaque fois un numéro. Si ce n'est pas le cas, alors EEP suppose qu'il s'agit d'un projet sans script Lua et ignore totalement le script.

La fonction `EEPMain()` convient à toute une série de tests.

Essayez pour constater ce qui se passe si vous insérez la fonction `print()` avant `return 1` :

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

Lorsque vous démarrez le projet et ouvrez la fenêtre d'événement, vous verrez que Lua a écrit le texte '*Bonjour !*' en boucle encore et toujours. Comme Bart Simpson dans la séquence d'introduction des Simpsons.

Vu que la fonction `EEPMain()` est appelée indéfiniment par EEP, cinq fois par seconde, c'est-à-dire toutes les 200 millisecondes, la répétition est exécutée rapidement.

C'est le moment d'un autre test. Ajoutez deux signes moins devant la fonction `print()` :

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

Si vous redémarrez le projet, le texte 'Bonjour !' cette fois-ci n'est pas affiché. Avec les deux signes 'moins', l'interpréteur le considère comme un commentaire et l'ignore purement et simplement. Vous venez d'apprendre une astuce très prisée des programmeurs pour désactiver temporairement des lignes de code dans un script.

D'autres explications sont données en annexe I. Veuillez étudier attentivement et utiliser les tutoriels pour apprendre de manière ludique quelles sont les conséquences des modifications dans l'écriture d'une fonction.

La ligne suivante du script définit une autre fonction. Son nom est `SETROUTE1` :

```
function SETROUTE1()
```

Le fait qu'il s'agit de la définition d'une fonction indique à l'interpréteur de ne pas l'exécuter à ce moment précis, mais de l'enregistrer pour une utilisation ultérieure. Cette fonction, encadrée en rouge (*Figure 11*), est appelée par le point de contact 2. Veuillez ouvrir la fenêtre des propriétés du point de contact :

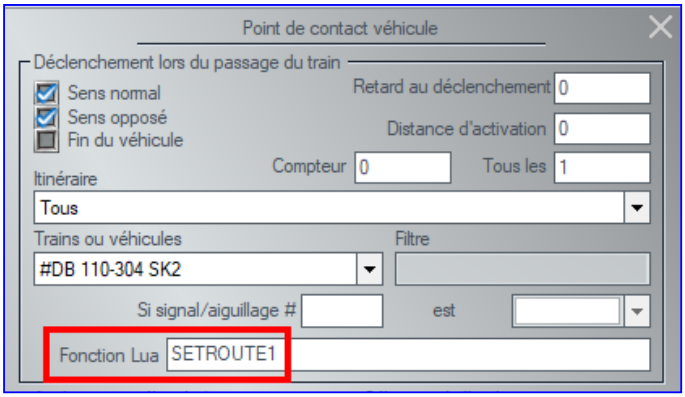


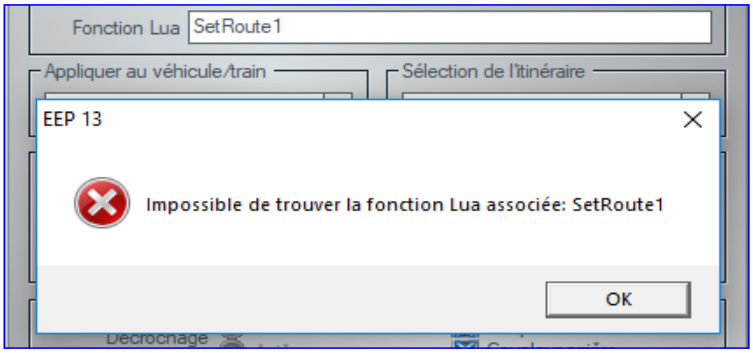
Fig. 11

Dans le cadre supérieur intitulé '***Déclenchement lors du passage du train***', se trouvent une ligne '***Fonction Lua***' et dans le champ de saisie situé après, le nom de la fonction qui est appelée lorsque le point de contact est déclenché. Veuillez noter que ce champ contient uniquement le nom de la fonction, mais pas les parenthèses ! Si vous saisissez les parenthèses à ce stade, cela entraînerait une faute lors de l'appel de la fonction. Seul le nom de la fonction doit être saisi dans le champ prévu à cet effet. Les parenthèses sont utilisées uniquement dans l'éditeur de script Lua.

Si le nom saisi dans la fenêtre du point de contact ne correspond pas exactement au nom donné à la fonction lors de sa définition dans le script, EEP ne trouve pas la fonction et émet un message d'erreur correspondant. Les lettres majuscules et minuscules sont différenciées et doivent donc être identiques dans la fenêtre du point de contact et dans la définition du nom de la fonction.

Essayez de remplacer `SETROUTE1` par `SetRoute1` dans le point de contact du véhicule et vous recevrez le message d'erreur suivant lorsque vous fermerez la fenêtre des propriétés :

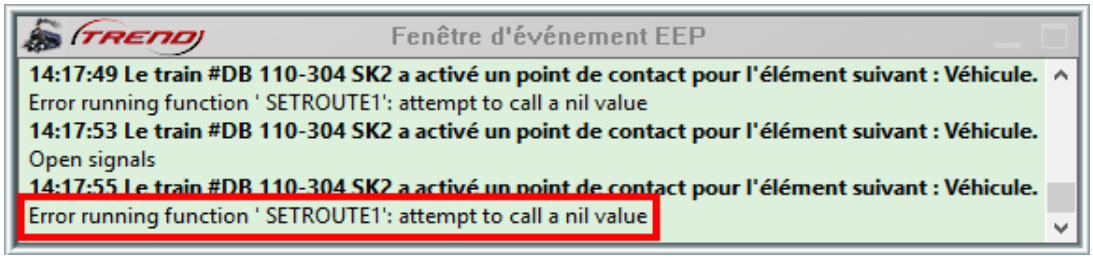
Fig. 12



Cette erreur se produit assez facilement, car nous ne faisons pas toujours attention à la casse des caractères entre les majuscules et les minuscules aussi précisément que l'interpréteur Lua.

L'erreur la plus commune est sans doute l'ajout d'un espace avant le nom de la fonction dans la fenêtre du point de contact. Elle est très difficile à détecter et ne produira pas de message d'erreur même lorsque le menu Propriétés est fermé. L'erreur se produit seulement lorsque le train franchit le point de contact. Par conséquent, le message d'erreur apparaît cette fois-ci dans la fenêtre d'événements :

Fig. 13



Vous devez regarder très attentivement pour voir qu'il y a un espace avant `SETROUTE1` dans le message d'erreur. C'est pourquoi nous vous recommandons de la tester par vous-même. Plus loin dans ce manuel, un tutoriel provoque délibérément cette erreur pour vous apprendre à avoir le réflexe d'y penser. Si cela devait vous arriver plus tard, vous comprendrez immédiatement ce qui s'est passé et où chercher pour trouver la ligne provoquant cette erreur.

La définition de la fonction `SETROUTE1` comporte de nombreuses instructions différentes :

```
function SETROUTE1()
  print("Fonction appelée par le point de contact")
  print("Commute l'aiguillage et le signal")
  -- variable changes switch --
  route = route + 1
  if ((route % 2) == 1) then
      print("Itinéraire 1")
      EEPSetSwitch(1, 1)
      EEPSetSignal(5, 2)
    else
      print("Itinéraire 2")
      EEPSetSwitch(1, 2)
      EEPSetSignal(4, 2)
  end
end
```

Tout d'abord, deux lignes de texte sont écrites dans la fenêtre d'événements suivi d'un commentaire. Les deux signes moins à la fin du commentaire ne servent qu'à des fins visuelles et n'ont aucune fonction. Tout ce qui suit après les deux premiers signes moins est considéré comme un commentaire. Il en est de même pour les deux signes moins à la fin.

La ligne suivante se lit ainsi : `route = route + 1,`
Vous aviez vu quelque chose de semblable plus haut dans le script : `route = 0,`
Une variable suivie d'un signe signifie qu'une valeur lui est affectée.

La différence ici est que cette valeur est incrémentée d'une unité. L'expression à droite du signe = est d'abord calculée et ensuite affectée à la variable à gauche du signe =. Cette commande est extrêmement pratique. Cela signifie que vous pouvez récupérer l'ancienne valeur d'une variable, modifier le résultat par le biais d'un calcul et sauvegarder à nouveau le résultat dans la même variable. Ainsi, dans cet exemple, la valeur 1 est ajoutée à la variable `route` chaque fois que le point de contact n° 2 est franchi. Avec cette méthode, en fonction de l'itinéraire emprunté par la locomotive, ce calcul permet de compter le nombre de tours effectués et de transférer la valeur dans la variable `route`.

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

Les mots `if` et `then` sont des mots-clés anglais. Le premier se traduit par '***si***' et le second par '***alors***'. On appelle cela un test conditionnel.

Les mots-clés indiquent à l'interpréteur du langage Lua ce qu'il faut faire. Ici, le mot clé '`if`' veut dire : si la condition est remplie '`then`' le programme continue. En langage informatique, on dit que le résultat retourné est vrai (`true`) si la condition est remplie.

Traduire cette ligne en langage courant donne ceci :
si `(route % 2) == 1`, alors ...

Lua vérifie le résultat du calcul avec la variable `route` et décide ce qu'il faut faire ensuite. Le caractère double == n'est pas une erreur d'écriture, mais appartient à la syntaxe de Lua. Il indique à l'interpréteur que deux valeurs doivent être comparées entre elles. Le résultat retourné est vrai ou faux (`true` ou `false`).

Dans cet exemple, `route % 2` est comparé au numéro `1`. Finalement, on vérifie si le calcul de `route % 2` retourne la valeur `1`.

Le signe pourcentage fait également partie de la syntaxe Lua. Cela n'a rien à voir avec le calcul d'un pourcentage, mais % est `l'opérateur modulo`. L'expression `route % 2` ne peut retourner qu'un résultat égal à `0` ou `1`. Ainsi, le résultat du test est alternativement vrai ou faux. Lua utilise ce résultat dans les lignes suivantes pour décider si le train doit parcourir la boucle extérieure ou intérieure.

Si le résultat est vrai, alors Lua exécute tout ce qui suit, jusqu'à ce qu'il rencontre une autre condition ou la fin du test. Cependant, si le résultat est faux, Lua passe au test suivant ou à la fin et continue à partir de ce point.

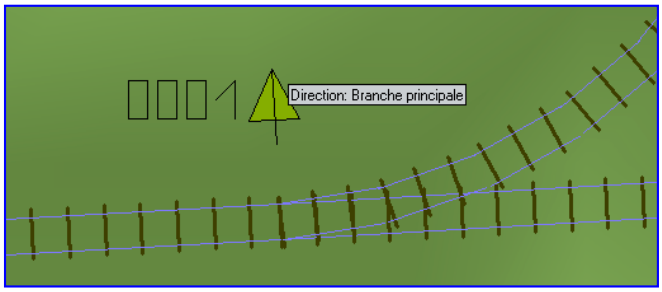
Pour faire simple, la structure `if` n'est rien de plus qu'un commutateur à deux états.

Si `route` contient un nombre impair, le résultat de `route % 2` est égal à `1` et dans ce cas, Lua exécute le bloc d'instructions situé entre then et else car le résultat du test est vrai (true).

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

La fonction `EEPSetSwitch(1,1)` place l'aiguillage 1 en position 1, c'est-à-dire 'Direction : Branche principale' (*Figure 14*). Le train va donc parcourir la boucle intérieure.

Fig. 14



La fonction `EEPSetSignal(5,2)` active le signal n° 5 en position n° 2. le signal n° 5 est la barrière du passage à niveau sur la boucle intérieure et la position n° 2 est positionnée sur '***Arrêt***', c'est-à-dire fermée et abaissée.

Les chiffres utilisés dans la fonction `EEPSetSignal()` pour la position du signal correspondent à la position des valeurs du signal dans la liste de sélection (Figure 15), qui se trouve dans la fenêtre des propriétés du signal.

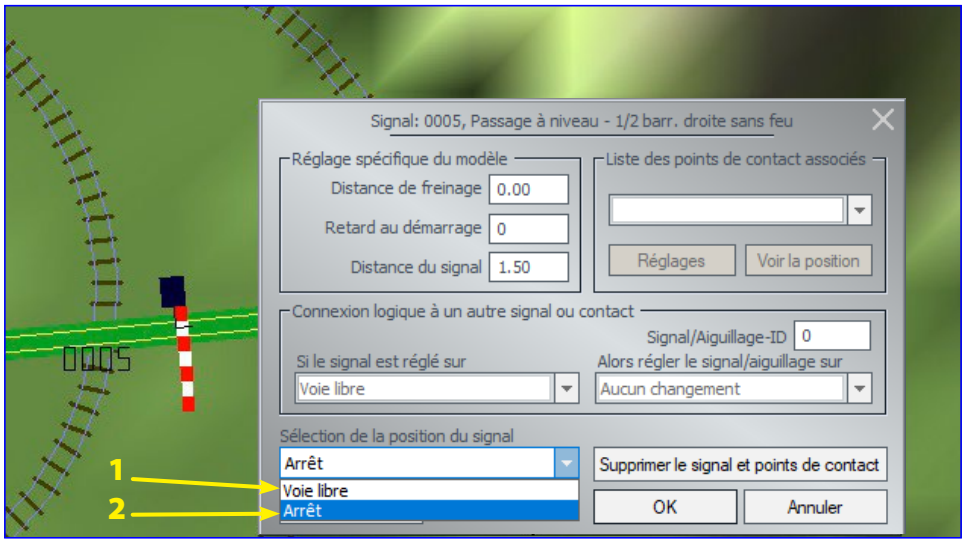


Fig. 15

Rien de plus est nécessaire quand la condition est remplie. Dans la ligne suivante, le mot-clé '`else`' signifie 'sinon' et informe l'interpréteur que les actions suivantes ne doivent être exécutées que si la première condition n'est pas remplie.

Si route contient un nombre pair, le résultat de `route % 2` n'est pas égal à `1` et le résultat du test retourne `false` (faux). Dans ce cas, l'interpréteur Lua passe immédiatement au mot-clé `else` et continue avec les instructions suivantes :

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

La fonction `EEPSetSwitch(1,2)` place l'aiguillage n° 1 en position n° 2, c'est-à-dire '***Embranchement***'. Le train va donc parcourir la boucle extérieure. Grâce à la fonction `EEPSetSignal(4,1)`, la barrière est fermée au niveau de la boucle extérieure.

Le script se termine par les deux lignes suivantes :

```
  end
end
```

Le mot `end` est aussi un mot-clé du langage de Lua. Le premier '`end`' n'indique pas la fin de la fonction mais la fin du test conditionnel. Comme vous venez de le voir, une fonction peut contenir plusieurs tests conditionnels. D'autres instructions du langage Lua peuvent également être présentes après le test `if - then - else`.

La fin de la fonction doit être obligatoirement indiquée à l'interpréteur Lua grâce au mot-clé end. Ici dans notre fonction `SETROUTE1`, il s'agit du deuxième end.

Peut-être vous êtes-vous déjà demandé pourquoi certaines lignes d'un script sont un peu décalées ? Tout programmeur qui se respecte utilise les indentations. Les blocs d'instructions indentés sont toujours bien présentés et clairement lisibles. L'interpréteur ignore l'indentation

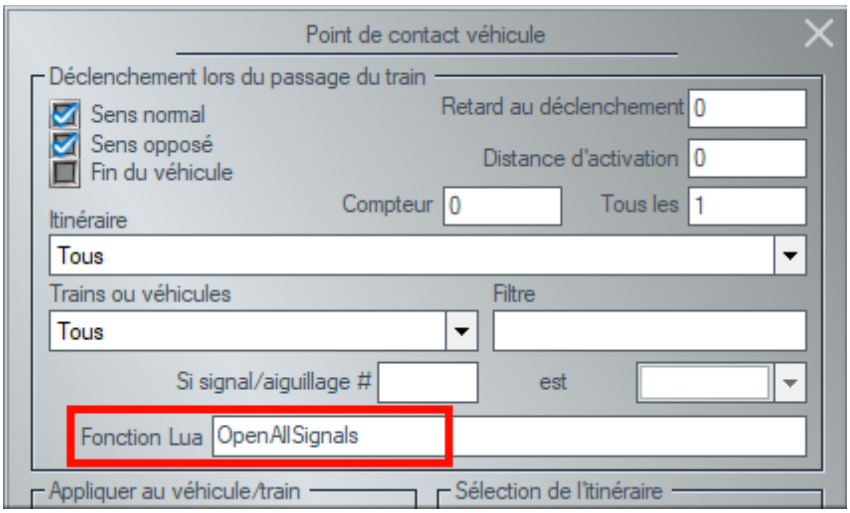
des lignes. Ainsi, lors d'une relecture, la compréhension du script est beaucoup plus grande. Et la fonction `SETROUTE1()` montre très bien à quel point les indentations peuvent être utiles.

Voici une autre fonction :

```
-- Ouvrir les signaux
function OpenAllSignals()
  print("Ouvrir les signaux")
  EEPSetSignal(4, 1)
  EEPSetSignal(5, 1)
end
```

Cette fonction nommée `OpenAllSignals()` est appelée dans le point de contact 1 (*Figure 16*), qui se trouve en haut derrière l'aiguillage n° 2. (*Figure 10*)

Fig. 16



L'orthographe du nom de cette fonction diffère sensiblement de la première. Alors que `SETROUTE1()` était écrit entièrement en majuscules, `OpenAllSignals()` alterne entre les lettres majuscules et minuscules. Cela ne fait aucune différence pour l'interpréteur tant que vous utilisez exactement la même notation pour la définition et l'appel de la fonction. Cela dit, l'orthographe de `OpenAllSignals()`, (Mettre une majuscule au début de chaque mot) est très répandue auprès des programmeurs. Cette méthode facilite la lecture des noms de fonctions.

La fonction `OpenAllSignals()` renvoie un texte et commute ensuite les deux barrières sur '***Voie libre***'. Les deux barrières sont donc relevées.

A partir de maintenant, vous avez lu beaucoup d'explications pour un exemple très simple. Mais vous avez aussi appris beaucoup de choses de base. Vous devriez consolider vos connaissances en modifiant le script de ce tutoriel avant de vous consacrer aux suivants.

Par exemple, vous pouvez vous intéresser de plus près à l'opérateur Modulo. Pourquoi ne pas remplacer `route % 2` par `route % 3` et voir ce qui change en conséquence. Vous devez laisser le train rouler pendant plusieurs tours avant de voir la différence.

Explication :

Le train parcourt une fois la boucle intérieure et deux fois la boucle extérieure. Avec `route % 3` vous obtenez les nombres 0 ; 1 ; et 2. Ainsi, la comparaison `(route % 3) == 1` retourne une fois `true` et deux fois `false`.

Et si le train doit rouler deux fois dans chaque ovale ?

Essayez d'écrire `(route % 4) < 2`. Cette fois-ci, (`route % 4`) renvoie les nombres 0 ; 1 ; 2 et 3. Les deux premiers sont inférieurs à 2 et les deux autres sont supérieurs. La condition est donc vraie (`true`) deux fois et fausse (`false`) deux fois.

Vous pouvez essayer d'autres modifications. Par exemple, vous pouvez intentionnellement faire en sorte que la commutation des barrières soit totalement erronée. Elles sont ouvertes lorsque le train passe et fermées lorsque le train ne passe pas.

Plus vous effectuerez de modifications et d'essais, plus vous serez à même d'apprendre et de progresser dans l'écriture de scripts avec le langage Lua. Ce qui est important avec les essais, c'est que vous apprenez à reconnaître les erreurs. Les erreurs ne sont pas un problème, elles sont au contraire bénéfiques et formatrices, car elles vous fournissent des informations précieuses.

5.2 Description du projet "Tutorial_34_LUA_2"

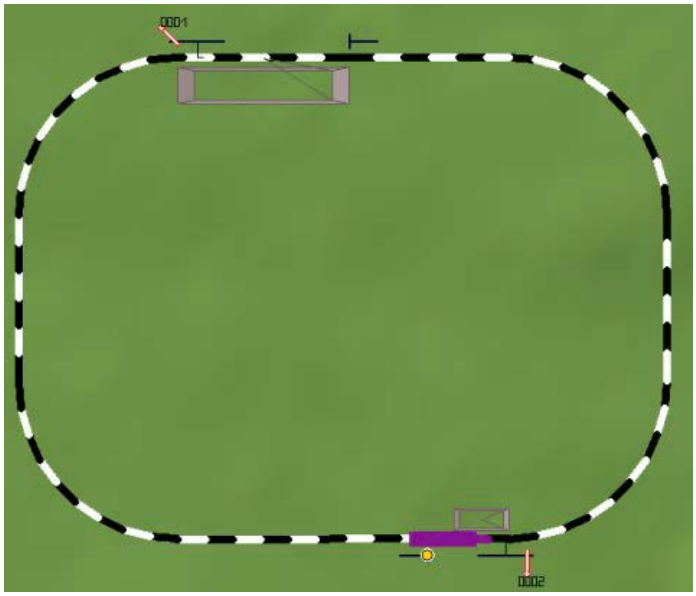


Fig. 17

Ce tutoriel décrit l'appel de fonction au moyen de signaux.

EEPRegisterSignal()
EEPOnSignal_x()

Dans le premier tutoriel, vous avez appris les bases. Ainsi, vous avez appris deux méthodes avec lesquelles EEP peut appeler des fonctions définies dans le script Lua. Tout d'abord, la fonction EEPMain() qui est appelée automatiquement cinq fois par seconde et deuxièmement, la possibilité d'appeler une fonction nommée par vos soins dans un point de contact.

Ce tutoriel vous présente la troisième façon dont EEP peut appeler une fonction Lua par un processus de commutation pour des signaux ou des aiguillages.

Ce processus s'appelle une fonction de rappel. 'Callback' est le mot anglais pour 'Rappel'. Une fonction de rappel est appelée lorsque l'état d'un signal ou d'un aiguillage change. La comparaison peut être similaire aux instructions if - then (Si l'état d'un signal change, alors...)

Vous connaissez déjà les premières lignes du script. D'abord, une variable nommée `I` est initialisée à 0 et ensuite un texte de bienvenue est affiché. La variable `I` n'est qu'un fragment du script d'origine et n'est pas pertinente dans ce tutoriel.

Voici les deux lignes suivantes :

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

La fonction EEPRegisterSignal() active la fonction de rappel pour un signal. Une fonction EEPRegisterSwitch() ferait la même chose pour un aiguillage. Le nombre entre parenthèses est le numéro du signal ou de l'aiguillage que vous voulez activer. Les zéros en tête peuvent être omis ici.

Cet enregistrement est nécessaire, car chaque signal et chaque aiguillage activerait une fonction de rappel pour toutes les opérations de commutation. Vous seriez alors obligé d'écrire les fonctions correspondantes pour tous les signaux et aiguillages dans votre script afin que l'appel de ces fonctions n'aboutisse pas à un message d'erreur.

Après l'enregistrement, vous retrouvez la définition obligatoire de la fonction `EEPMain()` et comme dans le tutoriel précédent, elle n'a pas d'autre tâche particulière que d'être appelée 5 fois par seconde.

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

Le reste du script se compose des définitions des fonctions `EEPOnSignal_1()` et `EEPOnSignal_2()`.

Elles décrivent grâce aux mots-clés ce qu'il faut faire lorsque le signal concerné est commuté.

La fonction `EEPOnSignal_1()` est appelée lors de la commutation du signal n° 1 par un clic de souris ou l'activation d'un point de contact. Il n'y a pas de point de contact dans ce tutoriel, vous devez donc changer le signal par un clic de souris pour observer l'effet.

Lorsque la fonction est appelée, le signal renvoie sa nouvelle position. Tout comme vous écrivez un texte entre parenthèses avec `print()`, le signal renvoie un nombre entre parenthèses après le nom de la fonction. Ce numéro doit être inclus dans les arguments (entre parenthèses) pour pouvoir être traité ultérieurement. Par conséquent, une variable dénommée statut est utilisée entre parenthèses après le nom de la fonction :

```
function EEPOnSignal _ 1(statut)
```

Veuillez noter que l'ID du signal n'est pas un paramètre, mais une partie du nom de la fonction. Cela signifie que chaque signal utilise sa propre fonction de rappel. De cette manière, EEP empêche les signaux multiples de modifier les fonctions de rappel lors de la commutation simultanée. Aucun rappel n'est perdu.

La fonction affiche d'abord un message dans la fenêtre d'événement :

```
print(" Statut de la fonction de rappel du signal 1 : ")
```

Suivi d'un test conditionnel dans lequel l'état du signal est vérifié :

```
if (statut == 1) then
```

Si la position du signal est définie à 1, c'est-à-dire '***Voie libre***', alors le statut correspondant à la position est écrit dans la fenêtre d'événement :

```
print("    Ouverture du signal (", statut, ")" );
```

et le signal 2 est mis en position 2, c'est-à-dire '***Arrêt***'.

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

Sinon
```
else
```

Si la position du signal n° 1 est déjà définie à 2, c'est-à-dire '***Arrêt***', alors le statut correspondant à la position est écrit dans la fenêtre d'événement :

```
print("    Fermeture du signal (", statut, ")" );
```

et le signal n° 2 est mis en position 1, c'est-à-dire '***Voie libre***' :

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

Fin du test conditionnel
```
end
```

Fin de la fonction
```
end
```

Il existe une fonction similaire mais plus simple pour le signal n° 2 :

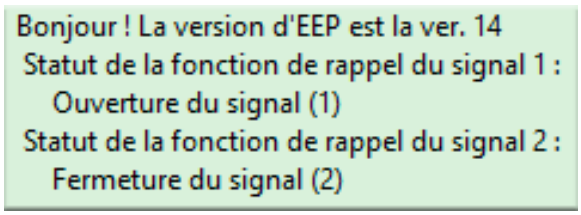
```
function EEPOnSignal_2(statut)
  print(" Statut de la fonction de rappel du signal 2 : ")

  if (statut == 1) then
    print("     Ouverture du signal (", statut, ")" );
  else
    print("     Fermeture du signal (", statut, ")" );
  end
end
```

Cette fonction affiche les statuts correspondants en fonction de la position du signal.

Lorsque vous démarrez le système, le train s'arrête au signal n° 1, le signal n° 2 de l'autre côté du circuit est positionné sur 'Voie libre'. Maintenant, si vous utilisez la combinaison [Maj + clic gauche] pour définir le signal devant vous sur la position 'Voie libre', le signal de l'autre côté commute sur la position 'Arrêt'.

La fenêtre d'événement (*Figure 18*) affiche les lignes suivantes :

Fig. 18



Basculez maintenant le signal devant vous sur la position '*Arrêt*' avec une autre combinaison ***[Maj + clic gauche]***. Vous remarquerez la commutation du signal de l'autre côté pour '*Voie libre*'. Il est facile de reconnaître la similitude avec le lien du test conditionnel `if - then` des signaux vu précédemment.

Cependant, la fonction de rappel peut allez beaucoup plus loin et le prochain tutoriel vous en donnera un petit aperçu.

Mais avant, attardons-nous sur l'appel de la fonction `EEPSetSignal()` dans la définition de la fonction `EEPOnSignal_1()`. Ici, il n'y a pas deux arguments entre parenthèses comme dans le tutoriel précédent, mais trois :

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

Le premier argument est une valeur numérique représentant l'ID du signal. Le deuxième argument correspond à l'état du signal. Le 1 du troisième argument signifie que le signal en question appelle également la fonction de rappel. C'est la raison pour laquelle, en plus des textes du signal n° 1, les textes du signal n° 2 sont affichés dans la fenêtre d'événement.

Il est temps de faire une petite expérience :

Supprimez le troisième chiffre dans les deux appels des fonctions `EEPSetSignal()`. N'oubliez pas de supprimer la virgule après le deuxième chiffre, sinon, l'interpréteur Lua délivrera un message d'erreur.

Si vous redémarrez le système, vous pouvez à nouveau commuter les deux signaux en appuyant sur [***Maj + clic gauche***] sur le signal devant vous. Rien n'a changé jusqu'à présent. Mais si vous regardez le texte dans la fenêtre d'événement, vous remarquerez que l'affichage du signal n° 2 n'apparaît pas.

Si vous modifiez le signal arrière directement en cliquant dessus, les textes apparaissent. Ce qui signifie que la fonction de rappel du signal n° 2 existe et fonctionne toujours, rien n'a changé. Mais la fonction est appelée uniquement si le signal est modifié directement ou par l'intermédiaire d'un point de contact.

Si vous souhaitez que la fonction Lua `EEPSetSignal()` exécute successivement une fonction de rappel, vous devez lui demander explicitement en spécifiant le chiffre 1 comme troisième argument lors de l'appel de la fonction.

Vous pouvez faire d'autres tentatives avec ce tutoriel.

Par exemple, pourquoi ne pas étendre la fonction de rappel pour le signal n° 2 de manière à ce qu'elle ne se limite pas seulement à afficher du texte, mais également provoquer la commutation du signal n° 1 ?

Tout ce que vous avez à faire est de modifier la fonction `EEPOnSignal_2` et de rajouter l'appel des fonctions `EEPSetSignal()` appropriées.

5.3 Description du projet "Tutorial_35_LUA_3"

Fig. 19

Ce tutoriel contient un second exemple d'utilisation des fonctions de rappel. Les méthodes utilisées sont les mêmes que dans le "***Tutoriel_34_LUA_2***".

EEPRegisterSignal()
EEPOnSignal_x()

Il n'y a pas de locomotive dans ce tutoriel. A la fin de chaque division de voies, il y a quatre feux de signalisation, que vous pouvez commuter par [***Shift + clic gauche***]. Prêtez attention au comportement des trois autres feux tricolores et des aiguillages. Chaque fois que vous passez un des quatre feux tricolores au vert, les autres passent au rouge et les aiguillages commutent pour ouvrir la voie de circulation correspondante au signal d'approche (feu vert).

Tout d'abord, les quatre feux tricolores doivent être enregistrés de manière à ce qu'ils appellent une fonction de rappel lors de la commutation. Les signaux (et les aiguillages) qui ne sont pas enregistrés ne peuvent pas appeler de fonctions Lua pendant les opérations de commutation.

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

Chaque script Lua pour EEP nécessite la définition de la fonction `EEPMain()`. Dans ce tutoriel, elle n'a pas d'autre tâche particulière que d'être appelée 5 fois par seconde.

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

La fonction de rappel pour un signal s'appelle `EEPOnSignal_x()`, où le `x` doit être remplacé par le numéro du signal. Les zéros initiaux d'un numéro de signal doivent être omis dans le nom de la fonction.

Cette fonction de rappel doit être définie dans le script pour que Lua sache comment réagir quand le signal est commuté.

```
function EEPOnSignal_4(statut)
```

La variable `statut` entre parenthèses après le nom de la fonction contient la position du signal.

Dans ce tutoriel, vous devez vérifier si le feu tricolore est passé au vert.

```
if(statut == 1) then
```

Si le feu tricolore a été réglé sur vert, l'aiguillage 1 ainsi que les signaux 5, 6 et 7 sont commutés :

```
  print("Set route 1")
  EEPSetSwitch(1, 1)
  EEPSetSignal(5, 2)
  EEPSetSignal(6, 2)
  EEPSetSignal(7, 2)
  -- Sinon si le feu était déjà vert, ce bloc d'instructions ne sera
  pas exécuté
end
```

Fin de la définition de la fonction de rappel pour le signal n° 4.

```
end
```

Les mêmes fonctions de rappel doivent également être définies pour les trois autres feux tricolores. Elles diffèrent seulement, dans la position des aiguillages et des signaux qui sont changés :

```
function EEPOnSignal_5(statut)
  if(statut == 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(statut)
  if(statut == 1) then
    print("Set route 3")

    EEPSetSwitch(1, 2)
    EEPSetSwitch(2, 1)
    EEPSetSwitch(3, 2)
    EEPSetSignal(4, 2)
    EEPSetSignal(5, 2)
    EEPSetSignal(7, 2)
```

```
  end
end

function EEPOnSignal_7(statut)
  if(statut == 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
```

Vous pouvez effectuer les essais suivants avec ce tutoriel :

Vous pouvez modifier les feux tricolores dans la fenêtre du plan 2D. Vous remarquerez que les autres signaux et aiguillages ne réagissent pas. Il s'agit d'une différence essentielle entre la fenêtre de planification et la vue 3D : Lua ne fonctionne que lorsqu'un projet est affiché en mode 3D.

Vous pouvez désactiver l'enregistrement d'un feu tricolore en modifiant la ligne concernée en commentaire :

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

Si vous actionnez ensuite ce feu tricolore, il n'y aura plus aucune action sur les autres feux de signalisation. La fonction nécessaire est toujours présente, mais un signal non enregistré ne déclenche pas de fonction de rappel.

Vous pouvez modifier le nom d'une fonction de rappel. Remplacer :

`function EEPOnSignal_4(status)` par `function EEPOnSignal_x(status)`

Si vous faites ensuite [***shift + clic gauche***] sur le signal n° 4 dans la vue 3D, vous recevrez un message d'erreur. Ce signal appelle la fonction nommée `EEPOnSignal_4()` qui n'est plus définie et non pas la fonction `EEPOnSignal_x()`.

L'interpréteur Lua ne se préoccupe pas des fonctions qui ne sont jamais appelées. Mais appeler une fonction qui n'a pas été définie, génère un message d'erreur.

N'oubliez pas d'ouvrir la fenêtre d'événement (*Figure 3*) pour que vous puissiez voir les messages d'erreur pendant vos essais.

5.4 Description du projet "Tutorial_36_LUA_4"

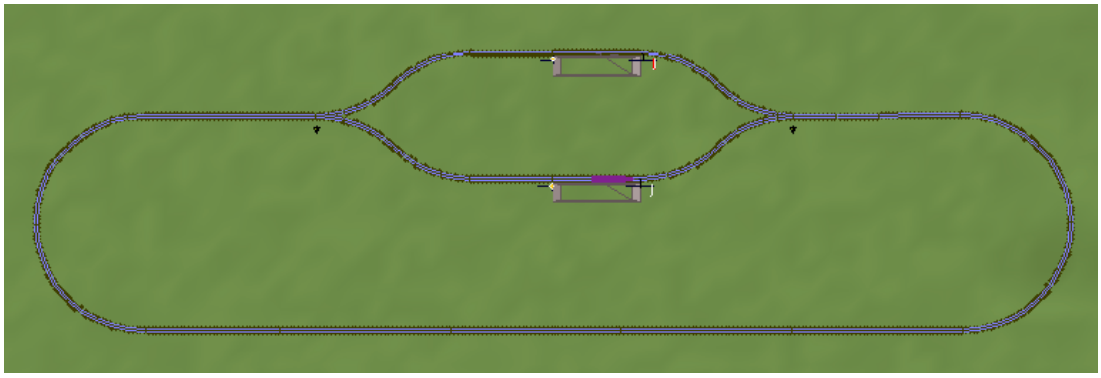


Fig. 20

Ce tutoriel vous montre un exemple simple d'utilisation de tableaux (array) dans un script Lua.

EEPMain()

Quatre points de contact sur le système appellent quatre fonctions différentes. Chacune de ces fonctions modifie le texte affiché dans la fenêtre d'événements une fois par seconde par l'intermédiaire de la fonction EEPMain().

Tout d'abord, deux tableaux sont définis et les éléments suivants sont assignés :

```
arr_lr = {"DROIT", "GAUCHE"}
arr_oc = {"OUVERT", "FERME"}
```

Les tableaux ne sont ni plus ni moins que des variables avec plusieurs emplacements de stockage. Parce que ces derniers peuvent contenir plusieurs valeurs sous un seul nom, ils se prêtent bien à un traitement efficace des données. En informatique, on les appelle également 'tableaux indicés'.

Ensuite, trois variables sont initialisées :

```
s = 0
TXT_info_1 = "Pas de train"
TXT_info_2 = "Pas de train"
```

La première variable nommée s est utilisée dans la fonction EEPMain() comme valeur de comptage pour les itérations. Des textes prêts à être affichés sont assignés aux deux autres variables.

La fonction EEPMain() sert de temporisateur dans ce tutoriel. Etant donné que cette fonction est appelée par EEP cinq fois par seconde (toutes les 200 millisecondes), elle est parfaitement adaptée aux tâches qui doivent être répétées en continu. Si vous comptez les itérations, vous pouvez définir une mesure du temps avec laquelle vous pouvez déclencher des actions.

```
function EEPMain()

  s = s + 1
  -- refresh info window every 1 second
  if (s > 5) then
    PrintInfo()
    s=0
  end

  return 1
end
```

La valeur de la variable `s` est incrémentée de `1` à chaque fois qu'EEP appelle la fonction `EEPMain()` et le résultat est à nouveau affecté à la même variable.

Le système vérifie ensuite si la valeur est supérieure à `5`. Si c'est le cas, la fonction nommée `PrintInfo()` est appelée et la valeur `0` est assignée de nouveau à la variable `s`. Sinon, il ne se passe rien d'autre. Enfin, `EEPMain()` renvoie la valeur `1` à EEP pour être appelée de nouveau.

Ensuite, nous avons une fonction appelée `SetSwitch()`, qui définit aléatoirement l'aiguillage n° 1 sur 'Branche principale' ou 'Embranchement'. Cette fonction est appelée par le point de contact peu avant l'entrée dans la gare à deux voies.

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

La fonction `math.random()` appartient à la bibliothèque Lua. Cela signifie que l'interpréteur connaît cette fonction qui peut être utilisée immédiatement. Celle-ci génère un nombre aléatoire entre les deux valeurs, qui sont définies entre parenthèses et séparées par une virgule.

Dans notre exemple, la fonction génère aléatoirement 1 ou 2, qui est ensuite affecté à la variable `Etat_Courant`. Dans la ligne suivante, `EEPSetSwitch()` utilise cette variable pour définir l'aiguillage n° 1 sur 'Branche principale' ou 'Embranchement'.

La fonction `PrintInfo()` ci-dessous est appelée à partir de la fonction `EEPMain()` toutes les secondes. Cette fonction affiche les différentes informations dans la fenêtre d'événement qui doit être ouverte (*Figure 3*) pour pouvoir lire ces messages.

```
function PrintInfo()
```

Efface le contenu actuel de la fenêtre des événements.
`clearlog()`

Ensuite, la date et l'heure actuelles du PC (et non l'heure d'EEP !) sont affichées. La fonction **`os.date()`** utilisée provient également de la bibliothèque Lua. Les paramètres de cette fonction déterminent le texte et le format d'affichage.

```
print(os.date("Date et heure courante : %c"))
```

Une ligne vide est insérée.

```
print("")
```

L'état du signal n° 4 est défini et mémorisé dans une variable.

```
signal4 _ state = EEPGetSignal(4)
```

Ensuite, `print()` affiche une ligne composée de deux parties. D'abord un texte fixe, puis une entrée du tableau `arr_oc`. La valeur de la variable `signal4_state` détermine si la première ou la deuxième entrée du tableau est utilisée :

```
print("Etat du signal n° 4 : ", arr _ oc[signal4 _ state])
```

La procédure pour le signal n° 3 est ensuite répétée :

```
signal3 _ state = EEPGetSignal(3)
print("Etat du signal n° 3 : ", arr _ oc[signal3 _ state])
```

Une autre ligne vide est insérée...

```
print("")
```

... et procéder de la même manière que précédemment avec les deux signaux :

```
print( "Etat du l'aiguillage n° 1 : ", arr _ lr[EEPGetSwitch(1)] )
```

Toutefois, ici, une variable n'a pas été utilisée comme un stockage temporaire. A la place, la position de l'aiguillage est déterminée directement par son numéro pour sélectionner la valeur correspondante dans le tableau `arr_lr`.

Une autre ligne vide est insérée....

```
print("")
```

et enfin l'affichage de deux textes stockés dans les variables `TXT_info_1` et `TXT_info_2`.

```
print ("Information gare n° 1 : ", TXT _ info _ 1)
print ("Information gare n° 2 : ", TXT _ info _ 2)
end
```

Les deux fonctions suivantes sont appelées par les points de contact au début des deux voies. Vous affectez de nouveaux textes aux variables `TXT_info_1` et `TXT_info_2`.

```
function SetInfoFromCP1()
  TXT _ info _ 1 = "Le train est arrivé en gare n° 1"
end

function SetInfoFromCP2()
  TXT _ info _ 2 = "Le train est arrivé en gare n° 2"
end
```

Et enfin, voici les deux fonctions qui sont appelées par les points de contact au bout des deux quais après les signaux 3 et 4. Comme les deux fonctions précédentes, elles modifient les textes mémorisés dans les variables `TXT_info_1` et `TXT_info_2`. Elles positionnent également les signaux sur la position '***Arrêt***'.

```
function SetInfoFromCP3()
  TXT_info_1 = "Le train quitte la gare n° 1"
  EEPSetSignal(4, 2)
end

function SetInfoFromCP4()
  TXT_info_2 = "Le train quitte la gare n° 2"
  EEPSetSignal(3, 2)
end
```

5.5 Description du projet "Tutorial_37_LUA_5"

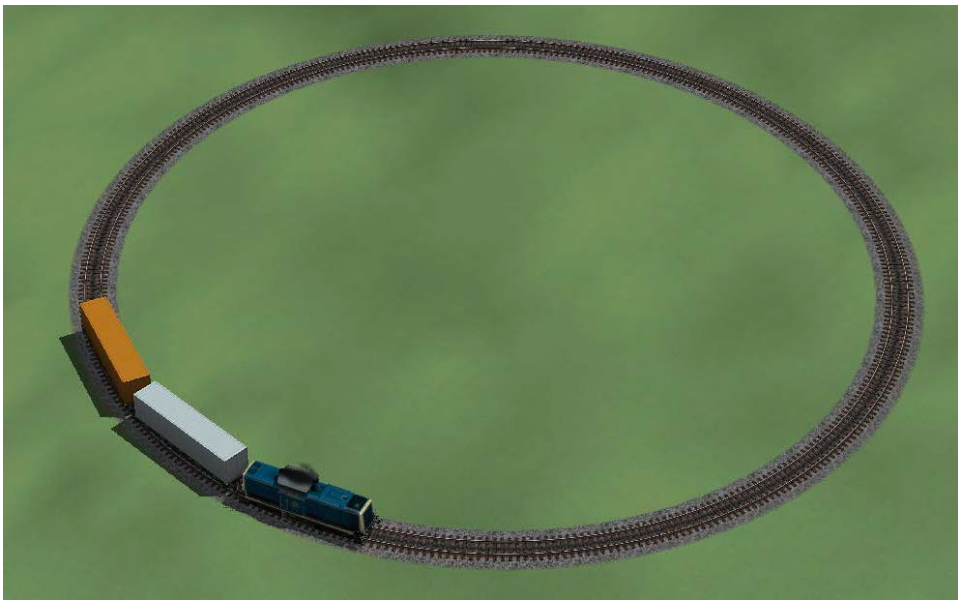


Fig. 21

Utilisation d'un 'compteur' pour initier une action.

EEPSetTrainSpeed()
EEPRollingstockSetCouplingRear()

La fonction EEPMain() est appelée 5 fois par seconde, donc le 'compteur' est incrémenté toutes les 1/5 secondes.

Cette fonction est utilisée ici pour programmer les séquences d'événements suivantes :

- La modification de la vitesse du train : EEPSetTrainSpeed(nom, vitesse)
- Couplage et désaccouplage des véhicules : EEPRollingstockSetCouplingRear (nom, état du couplage) où l'état du couplage signifie : 1 = accroché, 2 = décroché.

L'instruction `hResult, value = EEPRollingstockGetCouplingRear(nom)` renvoie la valeur 3 si un couplage est actif.

Vous trouverez ci-dessous, le script qui permet de mettre en œuvre les séquences d'événements pour la modification de la vitesse du train, le couplage et désaccouplage des véhicules.

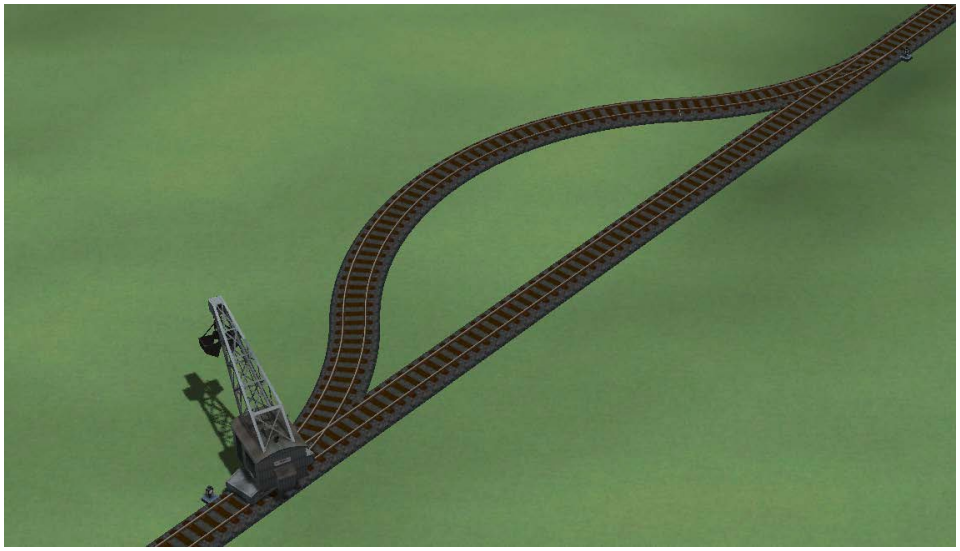
```
cycle=0
clearlog()

print("Bienvenue dans la version ", EEPVer, " d'EEP")
print("")
```

```
function EEPMain()

  cycle = cycle + 1
  if (cycle == 1) then

    -- DEPART --
    hResult = EEPSetTrainSpeed("#DB 212 309", 30)
    print("Valeur du compteur pour le départ : ", cycle, ", résultat :
    ", hResult)

  elseif (cycle == 30) then

    -- ARRET --
    hResult = EEPSetTrainSpeed("#DB 212 309", 0)
    print("Valeur du compteur pour l'arrêt : ", cycle, ", résultat : ",
    hResult)
    print("")

  elseif (cycle == 45) then

    -- DESACCOUPLAGE --
    hResult = EEPRollingstockSetCouplingRear("DB 212 309", 2)
    print( "Valeur du compteur pour le désaccouplage : ", cycle, ",
    résultat : ", hResult )

    -- DEPART --
    hResult = EEPSetTrainSpeed("#DB 212 309", 30)
    print( "Valeur du compteur pour le départ : ", cycle, ", résultat
    : ", hResult )
    print("")

  elseif (cycle == 55) then

    -- COUPLAGE --
    hResult = EEPRollingstockSetCouplingRear( "DB 212 309", 1 )
    print("Valeur du compteur pour le couplage : ", cycle, ", résultat
    : ", hResult )

    -- MARCHE ARRIERE --
    hResult = EEPSetTrainSpeed("#DB 212 309", -30)
    print("Valeur du compteur pour la marche arrière : ", cycle, ",
    résultat : ", hResult )
    print("")

  elseif (cycle > 55) then

    hResult, value = EEPRollingstockGetCouplingRear("DB 212 309")
    if (hResult) then
```

```
      if (value == 3) then
        -- DETECTION DU COUPLAGE --
        print("Couplage détecté (valeur du compteur) : ", cycle)
        print("")
        print("Remise à 0 du compteur")
        print("")
        cycle = 0
      end
    end

  end

  return 1

end  -- Fin de la fonction EEPMain
```

5.6 Description du projet "Tutorial_38_LUA_6"

Fig. 22



Les points de contact appellent les fonctions : Contact1() et Contact2().

Chaque fonction contient une liste d'instructions qui déclenchent les événements suivants :

- Vitesse de la grue via la fonction EEPSetTrainSpeed(nom, vitesse),
- Active le déplacement des éléments mobiles stockés dans un groupe (Les groupes contiennent les réglages pour les éléments mobiles et sont définis dans l'éditeur 3D pour un véhicule) via la fonction EEPRollingstockSetSlot(Nom du véhicule, n° du groupe),
- Commutation des aiguilles via la fonction EEPSetSwitch(ID, Position)

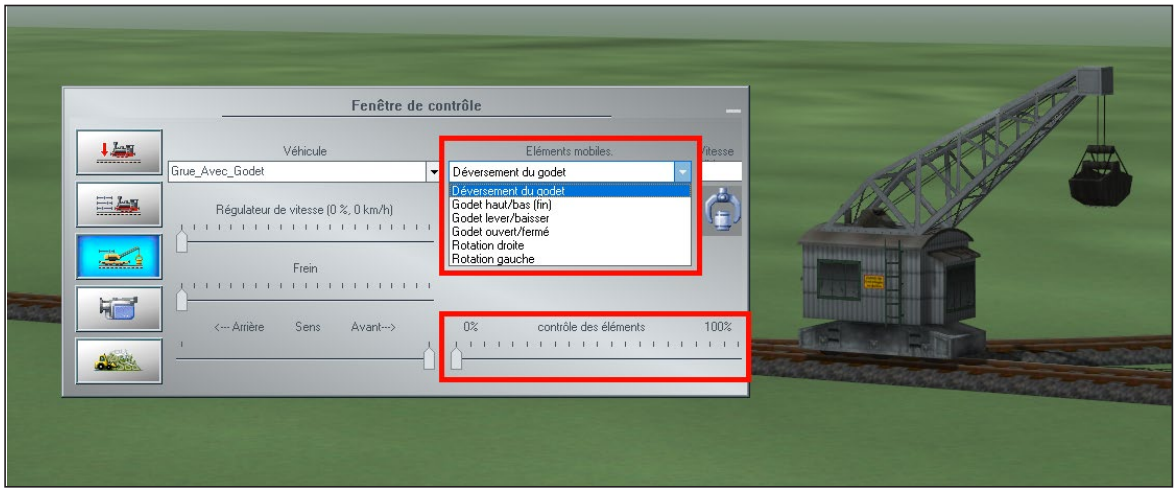
Voici le script :

```
clearlog()

print("Bienvenue dans la version ", EEPVer, " d'EEP")

function EEPMain()
  return 1
end

function Contact1()
  EEPRollingstockSetSlot("Ladekran2 Greifer", 1) -- 1er groupe
  EEPSetTrainSpeed("#Ladekran2 Greifer", 30) -- Marche avant
  EEPSetSwitch(1, 2)
  EEPSetSwitch(2, 1)
end

function Contact2()
```

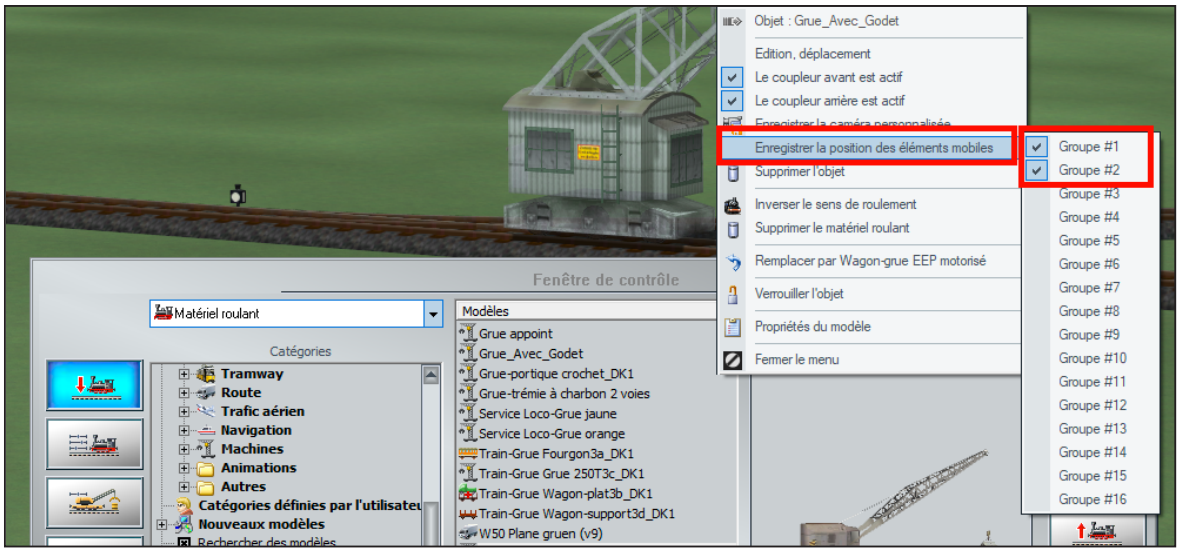
```
  EEPRollingstockSetSlot("Ladekran2 Greifer", 2) -- 2ème groupe
  EEPSetTrainSpeed("#Ladekran2 Greifer", -30) -- Marche arrière
  EEPSetSwitch(1, 1)
  EEPSetSwitch(2, 2)
end
```

Pour rappel, voici comment enregistrer une position (ou plusieurs en fonction du modèle) dans un groupe :

1. Commencez par choisir l'élément à paramétrer dans la liste déroulante '***Elements mobiles***' (1er encadré). Ensuite choisissez pour cette position, une valeur avec le curseur sur une échelle de 0 à 100 % (2eme encadré).



2. Une fois que vous avez terminé le réglage des positions, cliquez sur le bouton de l'éditeur (en haut), faites un clic droit sur la grue pour ouvrir le menu contextuel et choisissez l'option '*Enregistrer la position des éléments mobiles*'. Il ne vous restera plus qu'à sélectionner un groupe disponible parmi les 16. Un coche devant un groupe indique que celui-ci a déjà été configuré.



5.7 Description du projet "Tutorial_39_LUA_7"

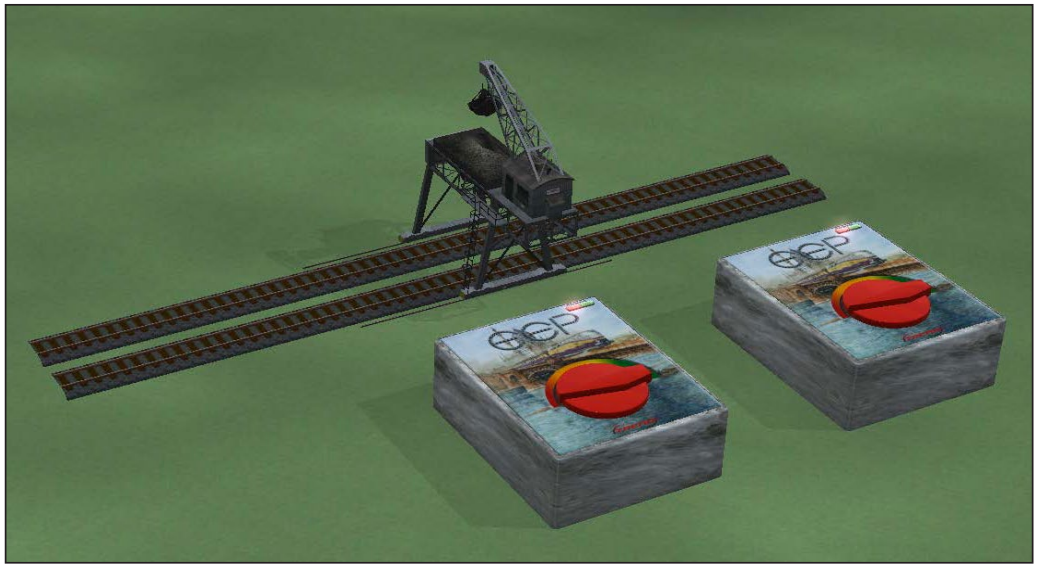


Fig. 23

Contrôle des éléments mobiles avec un script Lua

Si l'utilisateur appuie sur l'un des deux boutons rouges [***Shift + clic gauche***], tous les éléments mobiles sont commandés en même temps ou l'un après l'autre.

Voici une partie du script :

```
counter1 = 0
counter2 = 100

clearlog()

print("Bienvenue dans la version ", EEPVer, " d'EEP !")
print("")

EEPRegisterSignal(1)
EEPRegisterSignal(2)

function EEPMain()

  counter1 = counter1 + 1
  counter2 = counter2 + 1

  -- COMMUTE AUTOMATIQUEMENT LE SIGNAL 1 SUR ARRET --
  if(counter1 > 12) then
    -- hResult, value = EEPGetSignal(1)
    -- print("-->>", hResult, "    ", value)

    -- Récupère la position du signal n° 1
    value = EEPGetSignal(1)
    -- print("-->>  ", value)
    -- if (hResult) then
```

```
    -- Si la valeur est égale à la position 1
    if (value == 1) then
      -- Définir la position du signal n° 1 à 2
      hResult = EEPSetSignal(1, 2)
    end
  -- end
end

-- COMMUTE AUTOMATIQUEMENT LE SIGNAL 2 SUR ARRET --
if (counter2 > 12) then
  -- hResult, value = EEPGetSignal(2)

  -- Récupère la position du signal n° 2
  Value = EEPGetSignal(2)

  -- Si la valeur est égale à la position 1
  if (Value == 1) then
    -- Définir la position du signal n° 2 à 2
    hResult = EEPSetSignal(2, 2)
  end
end

-- APPEL DES FONCTIONS ADEQUATES SELON LA --
-- VALEUR STOCKEE DANS LA VARIABLE counter2 --

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("Tous les éléments mobiles sont mis en mouvement
    simultanément")
    print("")

    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
    "Rotation gauche", 50)
```

```
    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
    "Godet lever/baisser", 50)
    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
    "Godet ouvert/fermé", 0)
    EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
    "Schutt (Kohle Greifer)", 0)

    counter1 = 0
  end
end

function EEPOnSignal_2(status)
  if(status == 1) then
    print("Exécution de la séquence pas à pas")
    print("")
    counter2 = 0
  end
end

function GoStep1()
  state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
  "Godet lever/baisser", 100)
  print("Etape 1 : ", state)
end

function GoStep2()
  state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
  "Rotation gauche", 0)
  print("Etape 2 : ", state)
end

function GoStep3()
  state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
  "Godet ouvert/fermé", 100)
  print("Etape 3/1 : ", state)

  state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
  "Schutt (Kohle Greifer)", 100)
  print("Etape 3/2 : ", state)
end

function GoStep4()
  state = EEPRollingstockSetAxis("Bekohlungskranbrücke 1",
  "Schutt (Kohle Greifer)", 0)
  print("Etape 4 : ", state)
end
```

5.8 Description du projet "Tutorial_40_LUA_8"

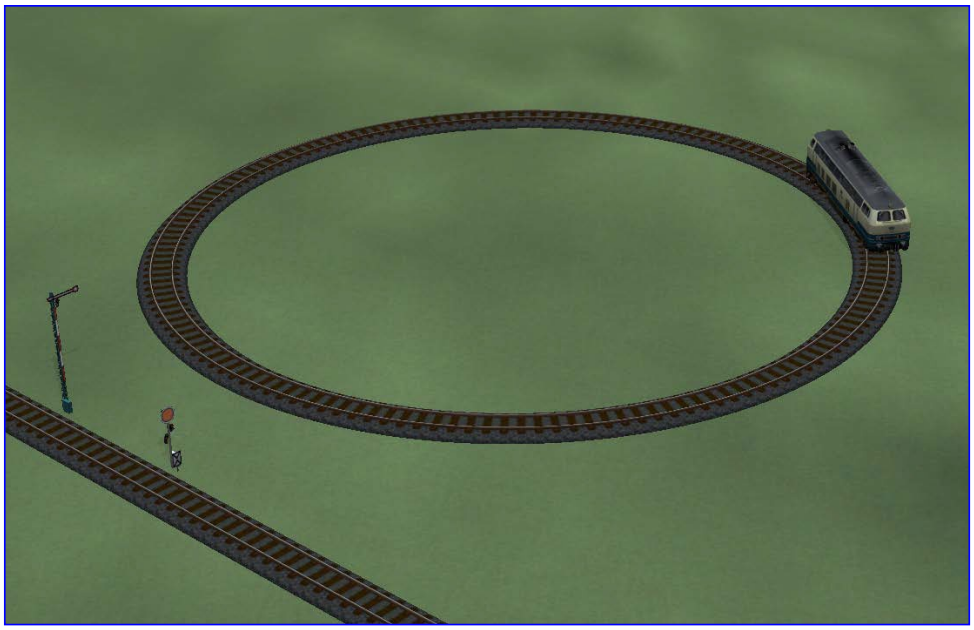


Fig. 24

Nouvelles fonctions pour l'enregistrement et le chargement de données.

```
hResult, data = EEPLoadData(slot_nr)
```
et
```
hResult = EEPSaveData(slot_nr, data)
```

Ces fonctions permettent d'enregistrer et de charger des données stockées dans un fichier (*. anl3). Dans cet exemple, les données sont affichées dans la fenêtre d'événement qui doit être activée dans les paramètres du programme.

Voici une partie du script :

```
-- On déclare 3 variables
A = 0
B = 10.0
C = "EEP est COOL"

clearlog() -- Efface la fenêtre d'événements

print("Bienvenue dans la version ", EEPVer, " d'EEP !")
print("")

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

```
function LoadData()

  -- clearlog()
  hResult, A = EEPLoadData(1) -- Charge le contenu depuis l'emplacement 1
  if (hResult) then -- Si la valeur retournée est vraie (true)
      print("A chargé : ", A)
      EEPSetSignal(1, A)
    else
      print("A n'existe pas")
  end

  hResult, B = EEPLoadData(2) -- Charge le contenu depuis l'emplacement 2
  if (hResult) then -- Si la valeur retournée est vraie (true)
      print("B chargé : ", B)
    else
      print("B n'existe pas")
  end

  hResult, C = EEPLoadData(3) -- Charge le contenu depuis l'emplacement 3
  if (hResult) then -- Si la valeur retournée est vraie (true)
      print("C chargé : ", C)
      print("")
    else
      print("C n'existe pas")
  end

end

function SaveData1()

  A = 1
  hResult, B = EEPGetTrainSpeed("#DB _ 218 _ 202")
  C = "EEP est COOL"
  SaveData()

end

function SaveData2()

  A = 2
  B = 10
  C = "EEP, c'est COOL"
  SaveData()

end
```

```
function SaveData()

  --clearlog()

  hResult = EEPSaveData(1, A)
  if (hResult) then
    print("A Enregistré - OK")
  end

  hResult = EEPSaveData(2, B)
  if (hResult) then
    print("B Enregistré - OK")
  end

  hResult = EEPSaveData(3, C)
  if (hResult) then
    print("C Enregistré - OK")
    print("")
  end

end
```

5.9 Description du projet "Tutorial_44_LUA_1"

Fig. 25

Ce tutoriel présente la gestion de la fumée, la lumière et le feu (incendie) dans des structures immobilières. voici les 3 fonctions qui permettent d'y parvenir :

EEPStructureSetSmoke()
EEPStructureSetLight()
EEPStructureSetFire()

Grâce à ces fonctions, ces événements sont exécutés automatiquement. Dans le bâtiment de gauche, la fumée apparait et disparaît alternativement, celui du milieu la lumière et dans le bâtiment de droite le feu.

Plusieurs variables sont initialisées au début du script :

```
I=0
hResult = 0  -- Variable pour enregistrer les résultats
IsSmoke = 0  -- Variable pour la fonction fumée
IsLight = 0-- Variable pour la fonction lumière
IsFire = 0 -- Variable pour la fonction incendie
```

La fonction `EEPMain()` est ensuite utilisée pour activer et désactiver les différentes propriétés des bâtiments à intervalles réguliers.

La variable I sert de compteur pour chaque appel de la fonction `EEPMain()`. Il est augmenté de `1` à chaque appel. Un certain nombre de tests vérifient la valeur de cette variable et décident ensuite des actions à effectuer.

La variable `hResult` contiendra le résultat vrai (true) ou faux (false) des différentes fonctions. Elle est utilisée uniquement dans ce but. Si la structure désignée possède la caractéristique d'émettre de la fumée, de la lumière et du feu, la valeur retournée sera vraie (true) sinon la

valeur retournée est fausse (**`false`**). Naturellement, ici dans notre exemple, la valeur sera toujours vraie, car nos structures possèdent les caractéristiques énumérées ci-dessus.

Les variables `IsSmoke`, `IsLight` et `IsFire` sont utilisées pour enregistrer la valeur (activée ou désactivée) de la propriété respective de chaque bâtiment.

```
function EEPMain()

  I = I + 1

  if (I == 15) then
    SetSmoke(true)       -- Appelle la fonction
    SetLight(false)      -- Appelle la fonction
    else
    if (I == 30) then
      SetSmoke(false)    -- Appelle la fonction
      SetLight(true)          -- Appelle la fonction
    end
  end

  if (I == 10) then
    SetFire(true)  -- Appelle la fonction
    else
    if (I == 20) then
      SetFire(false)     -- Appelle la fonction
    end
  end
```

En fonction de la valeur de I, soit : 15, 30, 10 ou 20, la fonction `EEPMain()` appelle différentes fonctions avec comme paramètre, true ou false pour activer ou désactiver la propriété correspondante.

Ensuite, l'état actuel des propriétés est déterminé à chaque itération et est sauvegardé respectivement dans les variables `IsSmoke`, `IsLight` et `IsFire` :

```
  hResult, IsSmoke = EEPStructureGetSmoke ( "#1 _ Maison de chargement
Lauscha" )
  hResult, IsLight = EEPStructureGetLight ( "#2 _ Service Bâtiment indus-
triel" )
  hResult, IsFire = EEPStructureGetFire ( "#3 _ Immeuble 3 niveaux en feu" )
```

et l'affiche sous forme de texte dans la fenêtre d'événement :

```
  print("Compteur : ", I," - Fumée : ", IsSmoke," - Lumière : ", IsLight,
  " -  Incendie : ", IsFire)
```

Enfin, le compteur `I` est remis à `0` lorsqu'il atteint la valeur `30` :

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

Ensuite, `EEPMain()` renvoie la valeur 1 afin d'être à nouveau appelée par EEP.

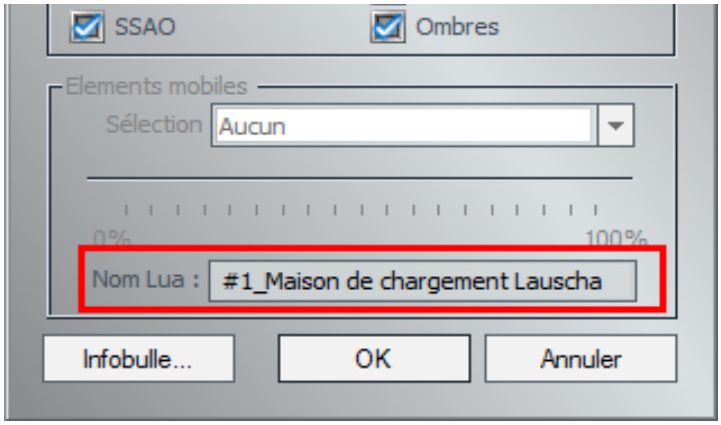
```
  return 1
end
```

Les trois fonctions suivantes activent ou désactivent la fumée, la lumière et le feu. Lorsqu'elle est appelée, chaque fonction reçoit la valeur `true` ou `false`. Cette valeur est enregistrée dans la variable `switch` pour être ensuite utilisée dans les fonctions EEP et activer (`true`) ou désactiver (`false`) la propriété.

```
-- Active ou désactive la fonction pour la fumée
function SetSmoke ( switch )
  EEPStructureSetSmoke ( "#1 _ Maison de chargement Lauscha", switch )
end

-- Active ou désactive la fonction pour la lumière
function SetLight ( switch )
  EEPStructureSetLight ( "#2 _ Service Bâtiment industriel", switch )
end

-- Active ou désactive la fonction pour le feu
function SetFire ( switch )
  EEPStructureSetFire ( "#3 _ Immeuble 3 niveaux en feu", switch )
end
```

Veuillez noter que les noms des modèles, qui doivent apparaître comme paramètres dans les fonctions, ont un numéro préfixé au début de leurs noms. Vous pouvez maintenant trouver ce '***Nom Lua***' dans le menu des propriétés du modèle. Veuillez utiliser uniquement le nom Lua pour toutes les fonctions liées à l'immobilier. Sans le numéro préfixé, l'interpréteur ne peut pas trouver la propriété.

Le numéro est obligatoire, mais le nom est facultatif.



Fenêtre des propriétés de l'objet avec le nom Lua.

5.10 Description du projet "Tutorial_45_LUA_2"

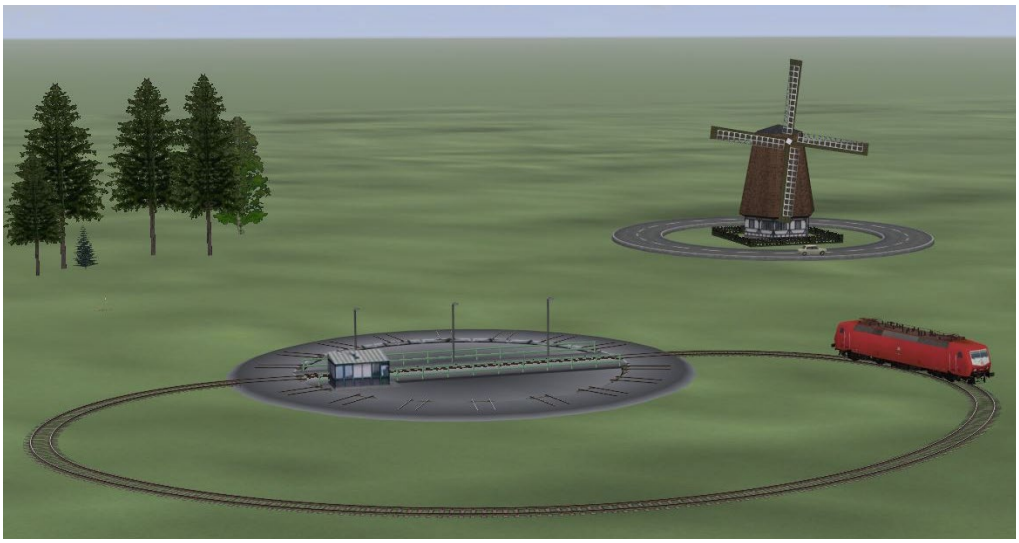


Fig. 26

Ce tutoriel montre la manipulation des éléments mobiles dans les structures immobilières et les objets ferroviaires.

EEPStructureAnimateAxis()
EEPStructureGetAxis()

Au premier plan, une locomotive roule sur la voie circulaire et s'arrête sur le plateau tournant à chaque passage. Celui-ci pivote de 180° et la locomotive repart dans l'autre sens. A l'arrière-plan, une voiture fait le tour du moulin et active ou désactive la rotation des ailes grâce à deux points de contact.

Tout d'abord, les variables suivantes sont initialisées :

```
UNLIMITED = 1000      -- Variable pour le mouvement continu
clearlog()            -- Efface la fenêtre d'événement EEP
hResult = 0           -- Variable pour enregistrer les résultats
Angle = 0             -- Variable pour l'angle du plateau tournant
PreviousAngle = -1    -- Variable pour l'angle précédent du plateau tournant
Speed = 0             -- Variable pour la vitesse
```

La fonction EEPMain() renvoie l'angle actuel du plateau tournant chaque fois qu'une rotation est effectuée et appelle la fonction CheckAnRunTrain(). Elle renvoie ensuite la valeur 1 pour être de nouveau appelée.

```
function EEPMain()
  hResult, Angle = EEPStructureGetAxis("#10 _ Pont tournant 360°-18 voies",
  "Pont")
  print("Angle de rotation : ", Angle)
  CheckAndRunTrain()          -- Appelle la fonction
```

```
  return 1
end
```

Un point de contact sur la route autour du moulin appelle la fonction `CallFromContactPoint1()`. Il contient un appel à la fonction Lua `EEPStructureAnimateAxis()` qui stoppe la rotation des ailes du moulin :

```
-- Cette fonction est appelée à partir du point de contact
-- Les ailes du moulin sont arrêtées
function CallFromContactPoint1()
  EEPStructureAnimateAxis("#12 _ Windmühle", "Muehlrad", 0);
end
```

La fonction nécessite trois paramètres :

Le nom de la structure immobilière, le nom de l'élément mobile et une valeur qui détermine le mouvement de celui-ci.

Veuillez noter que les noms des modèles, qui doivent apparaître comme paramètres dans les fonctions, ont un numéro préfixé au début de leurs noms. Vous pouvez maintenant trouver ce '***Nom Lua***' dans le menu des propriétés du modèle. Veuillez utiliser uniquement le nom Lua pour toutes les fonctions liées à l'immobilier. Sans le numéro préfixé, l'interpréteur ne peut pas trouver la propriété.

Le numéro est obligatoire, mais le nom est facultatif.

La valeur -1 en troisième position provoque l'arrêt des ailes du moulin.

La fonction `CallFromContactPoint2()` est également appelée par un autre point de contact situé sur la route :

```
-- Cette fonction est appelée à partir du point de contact
-- Les ailes sont en mouvement continu (Avec la valeur 1000,
-- représentée par la variable UNLIMITED)
function CallFromContactPoint2()
  EEPStructureAnimateAxis("#12 _ Windmühle", "Muehlrad", UNLIMITED);
end
```

Cette fonction diffère de la précédente. Le troisième paramètre utilisé cette fois-ci dans `EEPStructureAnimateAxis()` est la variable `UNLIMITED`. La valeur `1000` a été enregistrée dans cette variable au début du script. Le nombre `1000` provoque une rotation continue des ailes du moulin.

La fonction suivante `ContactPointOnTurntable()` est appelée par un point de contact situé au milieu du plateau tournant. La locomotive appelle donc cette fonction lorsqu'elle se déplace sur le plateau. Le point de contact règle également la vitesse de la locomotive à 0 :

```
-- Cette fonction est appelée à partir du point de contact
-- le plateau tournant est tourné de 9 pas (rotation à 180°)
```

```
function ContactPointOnTurntable()
  EEPStructureAnimateAxis("#10 _ Pont tournant 360°-18 voies", "Pont", 9)
end
```

Le premier paramètre de la fonction `EEPStructureAnimateAxis()` est le '***Nom Lua***' du plateau tournant, le second est le nom de l'élément mobile. Le troisième paramètre spécifie combien de pas le plateau tournant doit effectuer. Pour le modèle utilisé, 9 pas correspondent à un demi-tour.

La fonction `CheckAndRunTrain()`, est appelée par `EEPMain()` en permanence et définie comme suit :

```
-- Cette fonction est appelée en continu par EEPMain
-- Lorsque le pont tournant est arrêté et si la vitesse du train est à 0,
  il redémarre
function CheckAndRunTrain()
```

Tout d'abord, Elle lit la position actuelle du plateau tournant et l'enregistre dans la variable `Angle`:

```
  hResult, Angle = EEPStructureGetAxis("#10 _ Pont tournant 360°-18 voies",
  "Pont")
```

Elle détermine ensuite la vitesse actuelle de la locomotive et la sauvegarde dans la variable Speed :

```
  hResult, Speed = EEPGetTrainSpeed("#DB 120-119 or EpIV")
```

Après la position actuelle du plateau tournant, la position de l'angle précédent et la vitesse actuelle de la locomotive sont affichées dans la fenêtre d'événement :

```
  print("Angle : ", Angle, "   Angle précédent : ", PreviousAngle,
  "   Vitesse : ", Speed)
```

La fonction vérifie ensuite si le plateau tournant et la locomotive sont immobilisés :

```
  -- Si Angle = angle précédent et la vitesse = 0, défini la vitesse à 50
  if (Angle == PreviousAngle and Speed == 0) then
```

Remettre la locomotive en marche à 50 km/h :

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

Ou sinon ne fait rien

```
  end
```

Enfin, l'angle actuel du plateau tournant est transmis à la variable `PreviousAngle`:

```
  -- L'angle précédent est défini sur la valeur de l'angle actuel
  PreviousAngle = Angle
```

Tant que le plateau tournant est en mouvement, la nouvelle valeur de la position déterminée dans la variable Angle sera différente de celle transmise à la variable `PreviousAngle`. Ce

constat s'appuie sur la comparaison ci-dessus pour vérifier si le plateau tournant est immobile.

`end`

Fin de la fonction et par conséquent fin du script.

5.11 Description du projet "Tutorial_46_LUA_3"

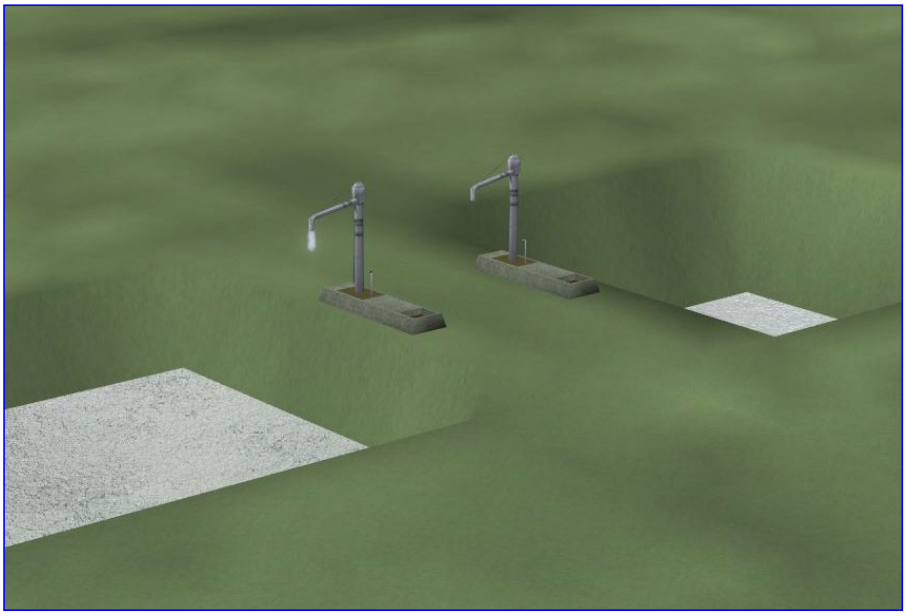


Fig. 27

Ce tutoriel démontre la possibilité de déplacer les éléments mobiles de certains objets dans une nouvelle position sans animation. Il montre également comment ceux-ci peuvent être déplacés sur le terrain au moyen de Lua.

`EEPStructureSetAxis()`
`EEPStructureGetAxis()`
`EEPStructureSetPosition()`

La grue à eau située à gauche fonctionne automatiquement. Elle pivote vers le bassin et l'eau coule jusqu'à ce que celui-ci soit plein. Puis la grue pivote dans l'autre sens et le bassin se vide. L'opération se répète ensuite invariablement.

La grue à eau située à droite peut être actionnée par vous-même. Tournez-la vers le bassin [***clic gauche + maj***] et celui-ci se remplira.

Les variables suivantes sont d'abord initialisées :

```
WaterLevel = -8          -- Variable du niveau de l'eau de la grue à gauche
FillTempo = 0.3          -- Variable pour la vitesse de remplissage
Water = 1                -- Variable pour l'eau
FillNone = 0             -- Variable pour le remplissage
FillUp = 1               -- Variable si le bassin est à remplir
FillOut = 2              -- Variable si le bassin est à vider
WaterLevelCondition = -8   -- Variable du niveau de l'eau grue à droite
AxisValueCondition = 50    -- Variable pour la valeur de l'élément mobile
hResult = 0              -- Variable pour enregistrer les résultats
```

La première des deux fonctions appelées par `EEPMain()` contrôle l'animation de la grue hydraulique située sur le côté gauche.

La seconde fonction est responsable de la grue hydraulique et du bassin sur le côté droit.

`EEPMain()` renvoie ensuite la valeur `1` pour être de nouveau appelée.

```
function EEPMain()

  ControllWaterFillingCycle()       -- Appelle la fonction
  ControllWaterFillingCondition()   -- Appelle la fonction

  return 1
end
```

La fonction `ControllWaterFillingCycle()` vérifie si le bassin gauche doit être rempli ou vidé. Elle teste l'égalité des valeurs de la variable `Water` avec les variables `FillUp` ou `FillOut` et appelle ensuite la fonction `Waterfill()` avec le paramètre `Filltempo`. Une fois avec le signe moins et une fois sans.

En utilisant la variable `FillTempo`, vous pouvez facilement modifier la vitesse de remplissage en lui attribuant une valeur différente lors de l'initialisation. Vous n'avez pas besoin de rechercher dans le script en entier et de modifier chaque entrée individuellement pour déterminer la vitesse de remplissage, mais seulement de changer la valeur affectée à la variable.

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

L'animation réelle se trouve dans la fonction `WaterFill()` qui est un peu plus complexe :

```
function WaterFill(fill)
```

Comme le bassin ne possède pas de propriété d'animation (Elément mobile) pour augmenter le niveau de l'eau, cette fonction monte ou abaisse le niveau à chaque appel. Ainsi, la nouvelle hauteur du niveau de l'eau doit être calculée à partir de l'ancienne :

```
  WaterLevel = WaterLevel + fill
```

`WaterLevel` contient la nouvelle hauteur du bassin. Pour ce faire, les trois valeurs de position doivent être renseignées et pas uniquement la hauteur à modifier. La fonction requiert obligatoirement les quatre paramètres : le nom de la propriété et les positions X, Y, Z.

Nous avons raccourci le nom dans cette documentation parce que sinon la ligne serait trop longue. En fait, le numéro seul de l'objet suffirait. Son nom est facultatif.

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

Ensuite, on vérifie si le bassin est plein. Si oui, la fonction `StopFilling()` est appelée :

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

Le test ci-dessous vérifie également si le bassin est vide. Si c'est le cas, la fonction `StartFilling()` est appelée pour remplir le bassin.

```
      if ( Water == FillOut ) then
        if (WaterLevel < -8.0) then
          StartFilling()
        end
      end
  end -- Fin du test
end -- Fin de la fonction
```

La fonction `StopFilling()` arrête le remplissage du bassin et fait tourner la grue hydraulique

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

Ensuite, on renseigne la variable `Water` pour savoir si le bassin est rempli ou vidé

```
  Water =  FillOut
end
```

La fonction `StartFilling()` fonctionne sur le même principe :

```
function StartFilling()
  EEPStructureSetAxis("#1 _ Wasserkran", "Wasser", 100)
```

```
  EEPStructureSetAxis("#1 _ Wasserkran", "Dreharm", 0)
  Water =  FillUp
end
```

Ainsi, vous avez le détail de tout ce qui contrôle la gestion de remplissage et du vidage du bassin de gauche.

La grue à eau située à droite est conçue pour un fonctionnement manuel. La fonction `ControllWaterFillingCondition()` appelée par `EEPMain()`, contrôle son comportement.

```
function ControllWaterFillingCondition()
```

La première étape consiste à vérifier si la grue à eau est au-dessus du bassin ou non. La position est déterminée...

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

... dans la fenêtre d'évènements...

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

...et vérifie si la rotation est supérieure à la valeur 80

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

Si c'est le cas, on vérifie si l'eau coule toujours dans le bassin. Comme celui-ci n'a pas de véritable propriété de remplissage, mais est simulé par la propriété hauteur, le test conditionnel `if` vérifie si celle-ci est toujours inférieure à `-2` mètres.

```
    -- Remplissage --
    if (WaterLevelCondition < -2) then
```

Dans ce cas, l'ouverture de l'eau est déclenchée par la grue hydraulique et la valeur de la position verticale du niveau de l'eau est augmentée de `0,75` mètres

```
      -- Pas encore rempli --
      EEPStructureSetAxis("#3", "Wasser", 100)
      WaterLevelCondition = WaterLevelCondition + 0.75
```

Ensuite la propriété est définie à sa nouvelle position :

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

Si la hauteur de la propriété n'est pas inférieure à `-2` mètres, alors rien ne se produit.

```
    end
```

Etant donné que la fonction `ControllWaterFillCondition()` est appelée en continu par `EEPMain()`, le niveau du bassin augmente un peu plus à chaque appel jusqu'à ce que le niveau maximal du basin soit atteint.

Si la grue hydraulique n'est pas orientée au-dessus de l'étang, alors...

```
  else
```

... le remplissage du bassin est arrêté...

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

... et vérification pour voir s'il y a encore de l'eau dans le bassin :

```
    -- Vidage --
    if (WaterLevelCondition > -8) then
      -- Pas encore vidé --
```

Dans ce cas, le niveau est réduit. Le principe est identique à celui du remplissage :

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

    end
```

Sinon, il ne se passe rien

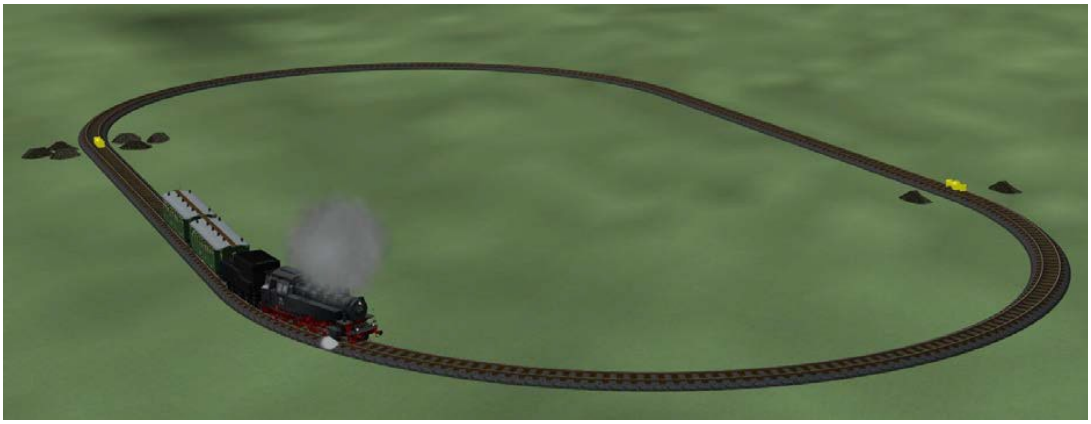
```
  end
```

Ici se termine la définition de cette fonction :

```
end
```

5.12 Description du projet "Tutorial_48_LUA"

Fig. 28



Ce tutoriel décrit comment utiliser les fonctions EEP pour la lumière, la fumée et le sifflet d'un train.

EEPSetTrainLight()
EEPSetTrainSmoke()
EEPSetTrainHorn()

Il y a deux points de contact sur le circuit, chacun d'eux appelle une fonction Lua. Depuis l'introduction du plugin 2 à EEP 11, les points de contact renvoient le nom du train dès qu'ils sont franchis.

Dans la définition de la fonction, le nom est stocké dans la variable name :

```
function TurnOnEffects(name)
```

La ligne...

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

...allume la lumière. La fonction a besoin des deux paramètres suivants : le nom complet du train et la valeur `true` pour allumer la lumière. Si le nom du train est correct, la fonction retourne `true`, sinon elle retourne `false`. Cette valeur est enregistrée dans la variable `hResult` et affichée par la suite avec l'instruction `print()`.

Les lignes...

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

...fonctionnent de la même manière que la lumière et activent la production de fumée et le klaxon d'avertissement du train.

Dans la fonction suivante :

```
function TurnOffEffects(name)
```

Le même principe est utilisé pour éteindre la lumière et la fumée. Le klaxon d'avertissement n'est pas désactivé, car il a cessé depuis longtemps.

5.13 Description du projet "Tutorial_49_LUA"

Fig. 29

Ce tutoriel explique comment contrôler les éléments mobiles d'un train et comment actionner et arrêter un crochet de grue ou saisir des charges.

`EEPSetTrainAxis()`
`EEPSetTrainHook()`

Le grue se déplace au-dessus d'un point de contact et appelle ainsi une fonction qui l'immobilise, met la flèche en position, prend la caisse et la dépose à un autre endroit. Ensuite, la flèche revient à sa position initiale.

On initialise les variables suivantes au début du script :

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

La variable `TimeCounter` sert de compteur et est incrémentée de `1` chaque fois que la fonction `EEPMain()` est appelée. Ce compteur est utilisé pour déterminer le moment auquel une des actions doit être exécutée.

La variable `TrainName` sera utilisée pour enregistrer le nom du train. La variable `bCraneIsReady` sert de drapeau indiquant si le train est en position de départ.

Le processus complet de ce tutoriel se déroule comme comme suit :

La grue se déplace au-dessus d'un point de contact qui appelle la fonction Lua `TakeGoodsStart(name)` qui arrête la grue et positionne la flèche au-dessus de la caisse. Les

appels de fonction des points de contact comportent toujours le nom du train déclencheur depuis l'introduction du plug-in 2 à EEP 11. Ici, la variable `name` récupère le nom du train.

Si vous ouvrez la fenêtre des propriétés du point de contact, vous trouverez le nom de la fonction `TakeGoodsStart` dans le champ '***Fonction Lua***' , mais sans parenthèses derrière et aucun paramètre. EEP complète le nom du train de manière complètement transparente pour l'utilisateur. Comme précédemment, seul le nom de la fonction peut figurer dans les points de contact !

La fonction de point de contact est définie dans le script comme suit :

```
-- Etape 1 - Arrête le train et tourne la flèche de la grue
function TakeGoodsStart(name)

  TrainName = name

  hResult = EEPSetTrainSpeed(name, 0)
  print("-- La grue est à l'arrêt : ", hResult)

  hResult = EEPSetTrainAxis(name, "Ausleger heben/senken", 100)
  print("-- Flèche en haut : ", hResult)
  hResult = EEPSetTrainAxis(name, "Drehung nach rechts", 90)
  print("-- La grue pivote à droite : ", hResult)

  hResult = EEPSetTrainHook(name, false)
  print("-- Désactive la fonction crochet : ", hResult)

  TimeCounter = 0
  bCraneIsReady = true

end
```

La ligne `TrainName = name` garantit que le nom du train est toujours disponible même après avoir quitté la fonction. La variable TrainName est requise ultérieurement dans la fonction `EEPMain()`.

La fonction `EEPSetTrainSpeed(name, 0)` arrête le train et enregistre le résultat de l'action dans la variable `hResult`.

La fonction `EEPSetTrainAxis(name, "Ausleger heben/senken", 100)` permet de relever la flèche.

La fonction `EEPSetTrainAxis(name, "Drehung nach rechts", 90)` fait pivoter la grue vers la droite.

La fonction `EEPSetTrainHook(name, false)` désactive le crochet.

Toutes ces fonctions utilisent la variable `name` comme premier argument pour désigner le train qui a déclenché l'action via le point de contact.
La ligne `TimeCounter = 0` réinitialise le compteur à zéro, afin que le timing soit parfait pour les actions suivantes déclenchées dans `EEPMain()`, et ne commence qu'une fois la fonction

`TakeGoodsStart` terminée.

La ligne `bCraneIsReady = true` garantit que la fonction `EEPMain()` a maintenant la permission de faire fonctionner la grue.

Veuillez noter que toutes les actions de la fonction `TakeGoodsStart()` se produisent immédiatement dès que le contact est franchi. Le compteur est donc remis à zéro au moment où le train passe au-dessus du point de contact et pas seulement lorsque les actions 'La grue est à l'arrêt, Flèche en haut, Désactive la fonction crochet' sont terminées ! Il en va de même pour `bCraneIsReady`.

Les actions suivantes sont exécutées dans l'ordre chronologique et se déroulent donc dans la fonction `EEPMain()`.

```
function EEPMain()

  TimeCounter = TimeCounter + 1

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

    if (TimeCounter == 60) then
      print("-- Chargement de la cargaison")
      EEPSetTrainHook(TrainName, true)
      print("-- La grue est en place")
      EEPSetTrainAxis(TrainName, "Ausleger heben/senken", 100)
    end

    if (TimeCounter == 80) then
      print("-- La grue pivote à droite")
      EEPSetTrainAxis(TrainName, "Drehung nach rechts", 70)
    end

    if (TimeCounter == 100) then
      print("-- Dépose de la marchandise")
      EEPSetTrainHook(TrainName, false)
    end

    if (TimeCounter == 110) then
      print("-- Rotation de la grue en position initiale")
      EEPSetTrainAxis(TrainName, "Ausleger heben/senken", 0)
      EEPSetTrainAxis(TrainName, "Drehung nach rechts", 100)
    end

  end
  return 1
end
```

Le test 'if bCraneIsReady then' vérifie si le train est prêt. La variable bCraneIsReady a été initialisée avec la valeur false au début du script. Par conséquent, la fonction EEPMain() ne fait rien jusqu'à ce que le train passe au-dessus du point de contact et règle la valeur de cette variable à true.

Les cinq tests conditionnels suivants vérifient si EEPMain() a été exécuté 30, 60, 80, 100 ou 110 fois. Ce compteur est remis à zéro au début du script par le franchissement du train lorsqu'il passe au-dessus du point de contact.

Après 30 itérations, EEPMain() appelle la fonction TakeGoods() et lui transmet le nom du train stocké dans la variable TrainName.

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

Cette fonction fait appel à EEPSetTrainAxis() qui a pour rôle d'abaisser la flèche.

Après 60 itérations, le crochet est activé pour charger la cargaison

EEPSetTrainHook(TrainName, true) et...

...la flèche est à nouveau relevée

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

Après 80 itérations, la grue sera pivoté à droite

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

après 100 itérations, le crochet décroche la marchandise pour la déposer

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

Enfin, la grue revient à sa position initiale après 110 itérations

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

et le script est terminé.

Pour rappel, la fonction EEPMain() est appelée cinq fois par seconde. Cela signifie, par exemple, qu'après 30 itérations, 6 secondes se sont écoulées.

5.14 Description du projet "Tutorial_50_LUA"

Fig. 30

Ce tutoriel montre comment définir l'itinéraire d'un train et lui attribuer un nouvel itinéraire avec les deux fonctions suivantes :

EEPGetTrainRoute()
EEPSetTrainRoute()

Le mouvement change à chaque tour de la route. Il conduit alternativement à Bergheim et à Hochspeyer.

Les variables `hResult` et `Route` sont initialisées au début du script. `hResult` sera utilisé pour stocker la valeur `true` ou `false` (se reporter à l'annexe). La variable `route` sera utilisé pour enregistrer le nom des itinéraires.

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

La fonction EEPMain() n'a pas de fonction particulière dans ce tutoriel et son appel répété est arrêté en affectant le numéro `0` comme valeur de retour.

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

Un point de contact pour événement sonore appelle la fonction `ChangeRoute()`.

Depuis l'introduction du plugin 2 à EEP 11, les points de contact renvoient le nom du train dès qu'ils sont déclenchés. Ici, le nom du train est stocké dans la variable `name`.

Si vous ouvrez la fenêtre des propriétés du point de contact, vous trouverez le nom de la fonction `ChangeRoute` dans le champ '***Fonction Lua***' , mais sans parenthèses derrière et aucun paramètre. EEP complète le nom du train de manière complètement transparente pour l'utilisateur.

Comme dans le tutoriel précédent, seul le nom de la fonction peut figurer dans les points de contact !

```
function ChangeRoute(name)

  hResult, Route = EEPGetTrainRoute(name)

  print("Route actuelle : ", Route, "(", hResult, ")")

  if hResult then

    if Route == "To Bergheim" then
      print("Itinéraire vers Hochspeyer")
      hResult = EEPSetTrainRoute(name, "To Hochspeyer")
    else
      print("Itinéraire vers Bergheim")
      hResult = EEPSetTrainRoute(name, "To Bergheim")
    end

  end

end
```

Au début de la fonction, `EEPGetTrainRoute()` détermine le nom de l'itinéraire prédéfini.

L'interpréteur vérifie ensuite si le nom de l'itinéraire peut être déterminé :

```
if hResult then
```

`hResult` peut retourner `true` ou `false`. Par conséquent, vous n'avez pas besoin de faire une comparaison avec un opérateur, mais vous pouvez utiliser la valeur de la variable directement dans le test.

Si `hResult` renvoie la valeur `true`, l'étape suivante consiste à vérifier si le nom renvoyé de l'itinéraire est 'To Bergheim' :

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

Si c'est le cas, l'itinéraire 'To Hochspeyer' est affecté au train :

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

Sinon, l'itinéraire "To Bergheim" lui est affecté.

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

Dans les deux cas, la valeur de retour des fonctions est récupérée dans la variable `hResult`, mais n'est plus utilisée.

Notez que les noms d'itinéraire sont des chaînes de caractères, c'est-à-dire des éléments de texte. Ils doivent être délimités en conséquence avec des guillemets comme une chaîne de caractères.

5.15 Description du projet "Tutorial_51_LUA"

Fig. 31

Ce tutoriel montre comment une partie du train peut être désaccouplée et comment les coupleurs d'un train peuvent être modifiés par un script.

`EEPSetTrainCouplingFront()` -- Non traité dans ce tutoriel
`EEPSetTrainCouplingRear()`
`EEPTrainLooseCoupling()`

Le train désaccouple le dernier wagon en haut du cercle. Un peu plus loin, il s'arrête, est rattrapé par le wagon et continue le tour dès que le wagon est à nouveau attelé.

La fonction `EEPMain()` n'a pas de fonction particulière dans ce tutoriel et son appel répété est arrêté en affectant le numéro `0` comme valeur de retour.

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

Deux points de contact contrôlent les événements sur le système. La première appelle la fonction `Unconnect()`.

Depuis l'introduction du plugin 2 à EEP 11, les points de contact renvoient le nom du train dès qu'ils sont déclenchés. Ici, le nom du train est stocké dans la variable `name`.

Si vous ouvrez la fenêtre des propriétés du point de contact, vous trouverez le nom de la fonction dans le champ '***Fonction Lua***' , mais sans parenthèses derrière et aucun paramètre. EEP complète le nom du train de manière complètement transparente pour l'utilisateur. Comme dans le tutoriel précédent, seul le nom de la fonction peut figurer dans les points de contact !

```
function Unconnect(name)

  print("Train : ", name)
  hResult = EEPTrainLooseCoupling(name, true, 3)
```

```
  if hResult then
    print("Désaccouplage")
  end
end
```

La variable `name` stocke le nom du train déclencheur.

Ensuite, la section du train derrière le troisième matériel roulant est désolidarisée.

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

Le paramètre true en deuxième position détermine la direction du comptage. Ici, `true` indique que le comptage se fait de l'avant. Si le paramètre `false` était défini, le comptage se ferait de l'arrière.

La partie avant du train passe le deuxième point de contact un peu plus loin. Celui-ci appelle la fonction `Stop()` :

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

La première ligne de cette fonction réduit la vitesse à 0.

La deuxième ligne réinitialise le coupleur arrière sur 'Accouplement' vu que celui-ci été désaccouplé au précédent point de contact.

Comme le dernier wagon a été désaccouplé à pleine vitesse, il a suffisamment d'énergie pour rattraper le train peu de temps après et se reconnecter. La vitesse cible fixée pour le wagon est toujours de 50 km/h. Ainsi, il transmet cette vitesse lorsqu'il est attelé au train, car dans EEP, lorsqu'un wagon est de nouveau attelé, le matériel roulant toujours en mouvement est toujours l'élément déterminant. En retour, il définit le nom et la vitesse du train nouvellement formé. Le train continue donc de rouler à 50 km/h et le cycle recommence.

5.16 Description du projet "Tutorial_55_Gleisbesetzt"

Fig. 32

Ce tutoriel démontre l'utilisation des fonctions d'occupation des voies.

`EEPRegisterRailTrack()`
`EEPIsRailTrackReserved()`

Le wagon frigorifique bloque le passage du train de marchandises. Les deux voies (Id 2 et Id 20) entre les deux aiguillages sont vérifiées chaque fois que la fonction `EEPMain()` est appelée pour voir si elles sont occupées par du matériel roulant. Si tel est le cas, le signal pour le train de marchandises est réglé sur la position 'Stop'. Lorsque le wagon frigorifique est écarté du chemin, Lua reconnaît que les voies sont dégagées et règle le signal pour le train de marchandises sur '***Voie libre***'. Pour dégager la voie, vous pouvez soit :

1. Actionner le grand transformateur sur la droite pour faire sortir une locomotive de manœuvre de la remise pour déplacer le wagon frigorique,
2. Pousser le wagon frigorifique avec la touche [Ctrl gauche] vers la voie qui mène à la remise,
3. Supprimer complètement le wagon.

Les deux voies doivent être enregistrées avant que lua puisse vérifier si elles sont occupées :

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

Les variables `Gleis1Reg` et `Gleis2Reg` sont utilisées pour enregistrer les voies concernées (ici dans notre exemple, il s'agit des voies avec l'Id 20 et l'Id 2). La valeur retournée par la fonction `EEPRegisterRailTrack` est vraie si les voies ont été enregistrées avec succès, sinon la valeur retournée est fausse.

Le contrôle de l'occupation des voies a lieu dans la fonction `Signal5Automatik()`, qui est appelée chaque fois que la fonction `EEPMain()` est exécutée, si les voies ont été enregistrées avec succès.

```
function Signal5Automatik()
```

Comme la fonction `EEPIsRailTrackReserved(xx)` renvoie deux valeurs, celles-ci sont d'abord stockées temporairement :

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

La première valeur retournée indique si la voie désignée a été trouvée, c'est-à-dire si elle existe et si elle est enregistrée :

```
  if hResult1 and hResult2 then
```

La deuxième valeur retournée indique si le wagon frigorifique est sur une des voies :

```
      if Besetzt1 or Besetzt2 then
```

Si une des voies est occupée, la valeur retournée est vraie donc le signal est réglé sur 'Stop' :

```
        EEPSetSignal(5,2) -- valeur 2 = Stop
```

Pour que le train de marchandises puisse circuler, il faut que le tronçon situé entre les deux aiguillages soit libre de tout matériel roulant. Ce qui exige que les valeurs des variables `Besetzt1` et `Besetzt2` soient fausses toutes les deux.

```
      else
```

Le tronçon est libre et le train de marchandises peut désormais circuler. Le signal est réglé sur 'Voie libre' :

```
        EEPSetSignal(5,1) -- valeur 1 = Voie libre
      end
```

Par souci de sécurité, on vérifie également si la voie désignée existe et a été enregistrée. Si c'était le cas, un message en informe l'utilisateur dans la fenêtre d'évènements :

```
    else
      print("Echec lors de la vérification des voies !")
  end
```

Le script vérifie également si la version EEP est compatible. Le minimum requis pour ce tutoriel est la version 11.3 d'EEP.

Le plugin n°3 pour EEP 11 est nécessaire et doit être installé, mais ne peut pas être vérifié par Lua. Par conséquent, l'appel répété de la fonction EEPMain() est arrêté si les deux voies n'ont pas pu être enregistrées.

6. Récapitulatif

Avec cette documentation, notre objectif est de vous donner un premier aperçu du monde de la programmation. A ce stade, nous aimerions vous donner quelques conseils : Essayez de mettre en application des tâches pas trop difficiles, surtout au début de votre apprentissage en tant que programmeur Lua. Travaillez selon le principe 'Commencer petit pour devenir grand'. Si vous êtes capable de résoudre une tâche simple, alors vous serez capable par la suite d'enrichir votre expérience pour une tâche plus complexe. D'autre part, si vous vous lancez dès le départ, dans une tâche trop ambitieuse, il y aura un risque d'échec. Commencez petit au début pour un meilleur apprentissage et ainsi, votre expérience ne fera que grandir et améliorer vos connaissances.

Il est préférable d'essayer et de reprendre dans un premier temps, les fonctions décrites dans les tutoriels et les adapter à des situations que vous pourriez rencontrer dans vos projets. Il est ainsi très facile de suivre les différentes étapes de votre apprentissage et vous ne vous perdez pas dans une tâche trop compliquée.

7. Perspective

Le nombre de fonctions Lua spécifiques à EEP continuera à s'étendre dans les années à venir. Les fonctions disponibles aujourd'hui, ne représentent que le début du processus visant à améliorer le rôle de Lua dans EEP.

Nous vous souhaitons beaucoup de succès et aussi beaucoup de plaisir avec l'automatisation de vos projets avec les scripts Lua !

L'équipe EEP

Annexe I

Commentaires

Les commentaires sont utilisés dans les scripts comme aide-mémoire et comme explications pour ceux et celles qui veulent relire et comprendre un script.

La syntaxe propre de Lua exige qu'un commentaire soit précédé de deux traits d'union :

```
-- Ceci est un commentaire
```

Tout ce qui suit après les deux traits d'union est ignoré par l'interpréteur, même s'il contient des fonctions, des mots-clés ou d'autres éléments du langage Lua.

```
-- La fonction suivante n'est pas exécutée : print("Commentaire")
```

N'importe quel caractère peut apparaître après les deux premiers traits d'union même d'autres traits d'union :

```
---------- Je suis aussi un commentaire ----------
```

Les commentaires peuvent également être derrière le code à exécuter :

```
print("Hallo") -- affiche le texte entre guillemets
```

Il est conseillé de fournir un script annoté avec des commentaires nombreux et aussi descriptifs que possible. Ils vous aident à conserver une bonne visibilité des éléments qui composent votre script. Si vous créez un script et que vous voulez le mettre à la disposition d'autres utilisateurs (soit pour l'utiliser, soit pour comprendre comment celui-ci fonctionne) alors les commentaires sont indispensables. Vous éviterez au lecteur d'avoir à réfléchir et à retracer toute l'analyse du processus de votre script. Inversement, gardez bien à l'esprit de ne pas non plus trop le surcharger de commentaires inutiles pour assurer une bonne visibilité de l'ensemble.

Les programmeurs aiment aussi utiliser les commentaires pour désactiver temporairement des parties de leur script. De cette façon, vous n'avez pas besoin de supprimer les lignes de programme et de les réécrire plus tard :

```
-- EEPSetSignal(1, 2)
-- EEPSetSwitch(3, 1)
-- print("Signal n° 1 réglé sur 'Voie libre' et l'aiguillage n° 3 réglé sur 'Embranchement'")
```

Si vous avez à nouveau besoin de ces lignes dans le script, supprimez simplement les deux traits d'union devant.

Vous devriez utiliser cette astuce lorsque vous expérimentez les tutoriels. Désactivez certaines lignes de code pour voir comment le script se comporte. Cependant, lors de la réactivation des lignes de code, veuillez vous assurer que vous n'enlevez pas accidentellement les traits d'union avant une vraie ligne de commentaire. Sinon, l'interpréteur lira le texte suivant comme du code et essaiera de l'exécuter, ce qui entraînera certainement des messages d'erreur.

Les variables

Les variables sont des emplacements mémoire dans lesquels une valeur quelconque est stockée pour une utilisation ultérieure. Le contenu de ces mémoires peut être modifié par le programme.

Vous pouvez comparer approximativement une variable avec un bloc-notes dans lequel vous conservez une liste de valeurs, par exemple. La valeur de cette note change à chaque fois que le programme modifie la valeur de la note, mais la note reste toujours la même. Ainsi, vous savez toujours où chercher pour vérifier la valeur stockée dans la variable.

Une variable reçoit un nom. Vous pouvez alors obtenir la valeur stockée sous ce nom chaque fois que vous en avez besoin et également changer la valeur si la situation l'exige.

La syntaxe Lua est dotée d'une règle de nommage stricte pour les variables et les fonctions.

Seuls les éléments suivants sont autorisés :

- Lettres majuscules et minuscules de l'alphabet latin,
- Nombres,
- Le trait de soulignement (Underscore, la touche du chiffre 8 alphanumérique).

Sont interdits les éléments suivants :

- Les trémas tels que ä, ö, ü et autres lettres exotiques et caractères spéciaux tels que ß, Ø, €, #,
- Tous les signes de ponctuation et les caractères spéciaux tels que point, virgule, hachage, étoile, plus et moins.
- L'espace.

De plus, un nom de variable ne peut pas commencer par un numéro.

Les variables peuvent être utilisées n'importe où comme marqueurs génériques pour un élément. Elles peuvent contenir des chiffres, des textes et plus encore.

Une application typique est la liste de contrôle mentionnée au début.

Tout d'abord, nous créons une variable en lui attribuant une valeur :

```
StrichListe = 0
```

Ensuite, nous lui attribuons une nouvelle valeur en lisant la valeur courante, en ajoutant 1 et en sauvegardant à nouveau le résultat dans la même variable :

```
StrichListe = StrichListe + 1  -- Maintenant la variable StrichListe possède la valeur 1
```

Et nous pouvons répéter ce processus aussi souvent que nous le souhaitons.:

```
StrichListe = StrichListe + 1  -- Maintenant la variable StrichListe possède la valeur 2
```

Cette méthode fonctionne parce que l'expression à droite du caractère = est d'abord calculée, puis le résultat est assigné à la variable à gauche du caractère =.

Les fonctions

Vous devez distinguer la différence entre la définition et l'appel d'une fonction. Pour mieux comprendre cela, nous vous recommandons d'ouvrir le tutoriel 'Tutorial_33_LUA_1.anl3'. Le script contenu dans ce projet contient à la fois des appels de fonctions et des définitions de fonctions.

La cinquième ligne du script contient :

```
clearlog()
```

Ici, c'est l'appel de la fonction appelée `clearlog()`. Elle est exécutée dès que l'interpréteur traite cette ligne du script.

Il en va de même pour :

```
print("Bonjour, vous utilisez la version d'EEP n° ", EEPVer)
```

qui est aussi un appel à la fonction `EEPVer`.

Maintenant, voyons la définition d'une fonction :

A la fin du script, vous trouverez une définition de fonction. Il y en a une avant, mais par souci de clarté, nous allons choisir la dernière en bas du script :

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

La définition d'une fonction est initiée par le mot-clé `function`. cette syntaxe indique à l'interprèteur de ne pas exécuter les instructions suivantes, mais de se rappeler le nom de la fonction après le mot-clé, ici le nom de la fonction est `OpenAllSignals`.

Étant donné que la liste des déclarations peut varier en nombre, l'interpréteur doit également savoir où la fonction se termine. Le mot-clé `end` est utilisé pour cela.

Dans cet exemple, Lua se souvient que la fonction définie et nommée par vous-même et portant le nom `OpenAllSignals` affiche un texte grâce à la fonction Lua `print()` et commute ensuite les signaux 4 et 5.

Le but d'une fonction est de simplifier le traitement des tâches qui doivent être exécutées plus d'une fois. Dans ce cas, il s'agit d'ouvrir les barrières des deux passages à niveau à chaque tour où la locomotive tourne, lorsque le train est passé.

Une fois que vous avez créé cette tâche, vous pouvez l'utiliser aussi souvent que vous le souhaitez en appelant la fonction par son nom.

Si vous étudiez le script du Tutoriel 33 en détail, vous remarquerez que d'une part on utilise des fonctions pour lesquelles il n'y a pas d'arguments et d'autre part on définit des fonctions qui ne sont pas appelées. La raison réside dans l'interaction entre EEP et Lua. Les fonctions `clearlog()`, `print()` et `EEPSetSignal()` sont définis directement dans EEP et peuvent donc être utilisés immédiatement. Inversement, les fonctions `EEPMain()`, `SETROUTE1()` et `OpenAllSignals()` sont appelées par EEP et doivent donc être définies.

L'interpréteur reconnaît les fonctions grâce aux parenthèses après le nom. Elles servent également 'd'espace mémoire' pour la transmission des arguments que la fonction doit utiliser lorsqu'elle est exécutée. Par exemple, les parenthèses de la fonction `print()` contiennent le texte que `print()` doit afficher dans la fenêtre d'événements.

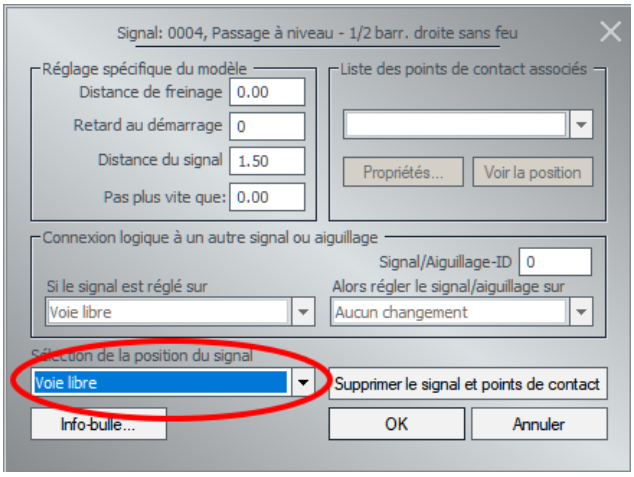
Les parenthèses sont nécessaires aussi bien pour appeler une fonction que pour définir une fonction. Ceci s'applique également si les parenthèses ne contiennent pas de contenu, c'est-à-dire si elles ne transfèrent aucun argument, comme c'est le cas avec `clearlog()`, par exemple.

L'espace mémoire, qui est représenté par les parenthèses, rend les fonctions polyvalentes. Par exemple, la fonction `print()` peut afficher des textes très différents, bien que la même fonction soit toujours utilisée pour cela.

Il en va de même pour la fonction `EEPSetSignal()`, qui est utilisée plusieurs fois dans le script. Elle peut commuter n'importe quel signal dans n'importe quelle position, bien que cette fonction n'ait été définie qu'une seule fois dans EEP. Lorsque vous l'appelez, écrivez simplement deux chiffres entre les parenthèses : d'abord l'ID, c'est-à-dire le numéro du signal, puis la position souhaitée du signal. Les deux numéros doivent être séparés par une virgule pour qu'ils soient correctement reconnus et assignés pendant l'exécution.

La position du signal, spécifiée par le deuxième chiffre, correspond à la position sous 'Sélection de la position du signal' dans les propriétés du signal. (Figure 33).

Fig. 33



L'opérateur modulo - %

L'opérateur modulo est si important que nous aimerions lui consacrer un chapitre spécifique.

Dans la syntaxe Lua, % (pourcentage) est le caractère pour désigné l'opérateur 'Modulo'.

Le caractère % dans le code Lua n'a rien à voir avec le calcul de pourcentage classique. Les développeurs de Lua ont simplement utilisé ce signe à d'autres fins.

En programmation, il y a souvent des situations dans lesquelles seuls les chiffres compris dans une petite plage sont nécessaires. Dans le cadre d'EEP, par exemple, il peut s'agir des numéros de voie d'une gare ferroviaire. L'opérateur modulo s'assure qu'à partir d'une valeur limite donnée, le programme recommence à partir du début. Cette limite doit suivre le signe %.

Vous pouvez à peu près comparer cette horloge à une montre analogique. De 1 h à 12 h, elle affiche exactement ces chiffres. Mais à partir de 13 heures, elle affiche de nouveau 1 heure du matin, à 14 heures, elle affiche 2 heures du matin et ainsi de suite.

Une différence par rapport à l'horloge est que l'opérateur Modulo commence toujours par 0 et non par 1.

Si la valeur à gauche est inférieure à la valeur à droite de %, le résultat est le nombre à gauche :

```
0 % 5 = 0
1 % 5 = 1
2 % 5 = 2
3 % 5 = 3
4 % 5 = 4
```

Si la valeur à droite du % est atteinte, le résultat est 0 :

```
5 % 5 = 0
```

Le 5 est soustrait du chiffre de gauche jusqu'à ce que le reste soit inférieur à 5:

```
6 % 5 = 1
7 % 5 = 2
8 % 5 = 3
9 % 5 = 4
10 % 5 = 0
```

Bien sûr, l'opérateur Modulo fonctionne aussi avec des nombres négatifs :

```
-1 % 5 = 4
```

Il vous sera peut-être plus aisé de bien comprendre l'opérateur Modulo si nous mettons deux rangées de chiffres l'une sur l'autre. Au-dessus une série de nombres et en dessous, le résultat pour l'opérateur Modulo % 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
```

Lors de l'application de l'opérateur Modulo sur des compteurs incrémentés en temps réel, vous devez prendre en compte l'ordre des commandes :

```
(Valeur + 1) % 4 donne toujours des nombres de 0 à 3
(Valeur % 4) + 1 donne toujours des nombres de 1 à 4
```

La deuxième méthode est le plus souvent en adéquation avec l'EEP, car les numéros de voie, les réglages des signaux et des aiguillages et bien d'autres choses encore ne commencent pas par 0, mais par 1.

Tableaux (Arrays)

Dans Lua, les arrays ne sont rien de plus que des tableaux unidimensionnels. C'est pourquoi nous parlerons exclusivement de tableaux dans ce chapitre. Un tableau est utile parce qu'il regroupe plusieurs valeurs sous un seul nom.

Dans la syntaxe de Lua, un tableau est créé en assignant 'quelque chose' entre deux accolades :

```
monTableau = {}
```

Les parenthèses peuvent rester vides au début ou recevoir un contenu immédiatement.

S'il y a plusieurs entrées, elles doivent être séparées par des virgules :

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

Les emplacements dans un tableau sont appelés index. Dans la syntaxe Lua, l'index est écrit entre crochets. Un index peut être un numéro ou un nom :

`monTableau[1]` représente la première entrée du tableau.

Si nous avons rempli le tableau avec les valeurs précédentes, alors

`print(monTableau[1])` affiche le chiffre n° 4 dans la fenêtre d'évènements.

avec

```
monTableau[1] = 123
```

la première entrée du tableau serait remplacée par le chiffre 123.

Les noms pour les emplacements d'un tableau peuvent être les suivants :

```
monTableau["Signal"]
```

On peut aussi l'écrire sous cette forme :

```
monTableau.Signal
```

La deuxième variante est plus lisible et plus pratique que la première, tout en ayant la même signification.

Lors de l'initialisation, le caractère = est utilisé pour les affectations afin de donner un nom aux emplacements du tableau.

```
monAiguillage = { ID = 2 , Position = 1, Parcours = 4 }
```

`monAiguillage.ID` est = à 2

`monAiguillage.Position` est = à 1

`monAiguillage.Parcours` est = à 4

Puisqu'un emplacement du tableau peut contenir un tableau complet, des tableaux multidimensionnels sont également possibles :

```
monAiguillage = {}
monAiguillage[1] = { ID = 2 , Position = 1, Parcours = 4 }
monAiguillage[2] = { ID = 7 , Position = 2, Parcours = 4 }
```

`monAiguillage[2].Position` est = à 2

Les tableaux sont un moyen simple et pratique de gérer de grandes quantités de données.

Annexe II

Variables et fonctions Lua spécifiques à EEP

Ce manuel détaille les fonctions introduites dans EEP jusqu'à la version n° 14.

Variables système

EEPVer	EEPVer
Type	Variable
Utilisé dans	Script
Défini dans	EEP
Requis	EEP 10.2 plug-in 2
Objectif	Retourne le numéro de version d'EEP.

```
if EEPVer < 11 then
  print("Aucune fonction de train
  supportée dans cette version EEP !")
end
```

EEPTime	EEPTime
Type	Variable
Utilisé dans	Script
Défini dans	EEP
Requis	EEP 10.2 plug-in 2
Objectif	EEPTime fournit une variable qui représente l'heure courante dans le projet EEP. La valeur est égale aux secondes écoulées depuis minuit (temps EEP).

```
if EEPTime == oldTime + 50 then
     print("50 secondes sont écoulées.")
     oldTime = EEPTime
  elseif EEPTime > oldTime + 50 then
     print("Plus de 50 secondes se sont écoulées."
     oldTime = EEPTime
  else
     print("50 secondes ne se sont pas encore écoulées.")
end
```

EEPTimeH	EEPTimeH
Type	Variable
Utilisé dans	Script
Défini dans	EEP
Requis	EEP 10.2 plug-in 2
Objectif	Retourne la partie horaire des heures d'EEPTime, exprimée sous forme de valeur comprise entre 0 et 23.

```
print(" Il est actuellement : ",EEPTimeH,":",EEPTimeM)
```

EEPTimeM	EEPTimeM
Type	Variable
Utilisé dans	Script
Défini dans	EEP
Requis	EEP 10.2 plug-in 2
Objectif	Retourne la partie horaire des minutes d'EEPTime, exprimée sous forme de valeur comprise entre 0 et 23.

```
print(" Il est actuellement : ",EEPTimeH,":",EEPTimeM)
```

EEPTimeS	EEPTimeS
Type	Variable
Utilisé dans	Script
Défini dans	EEP
Requis	EEP 10.2 plug-in 2
Objectif	Retourne la partie horaire des secondes d'EEPTime, exprimée sous forme de valeur comprise entre 0 et 59.

```
if EEPTimeS == 15 then
     EEPSetSignal(1, 1) -- Feu de signalisation vert
  elseif EEPTimeS == 45 then
     EEPSetSignal(1, 1) -- Feu de signalisation rouge
end
```

Fonctions système

clearlog()	clearlog()
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètres	Aucun
Valeurs renvoyées	Aucune
Requis	EEP 10.2 plug-in 2
Objectif	Efface la fenêtre d'événement Lua.

```
clearlog()
```

print()	print("Text1", "Text2", ..., TextN)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètres	Multiples
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2
Objectif	Écrit des arguments dans la fenêtre d'événement Lua (activée dans les propriétés EEP)
Commentaires	• Les nombres sont convertis en chaînes de caractères. • Accepte plusieurs arguments. Utilisez la virgule comme séparateur. Retour chariot avec \n. • Retourne l'intégralité du texte affiché en une seule chaîne de caractères.

```
print(" Il est actuellement : ",EEPTimeH,":",EEPTimeM)
```

EEPMain()	EEPMain()
Type	Fonction
Utilisé dans	**Script**
Défini dans	EEP
Paramètres	Aucun
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2
Objectif	Cette fonction est appelée par EEP 5 fois par seconde (cad toutes les 200 millisecondes) Elle est utilisée pour toutes les actions nécessitant des répétitions constantes.
Commentaires	• La déclaration de cette fonction est obligatoire dans tous les scripts. • La fonction est appelée par EEP sans aucun paramètre. • La fonction doit renvoyer un nombre différent de 0 pour être appelée à nouveau. • Le renvoi de la valeur 0 désactive l'appel répété de cette fonction. Toutes les autres fonctions de votre script restent actives. • Si cette fonction renvoie autre chose qu'un nombre, EEP cessera d'utiliser le script.

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

Fonctions de gestion des signaux

EEPSetSignal()	EEPSetSignal(ID, Etat, Callback)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux ou trois
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2
Objectif	Commutation d'un signal.
Commentaires	• Le premier argument est une valeur numérique représentant l'ID du signal. • Le deuxième argument est l'état du signal. • Si le nombre 1 est entré comme troisième argument (facultatif), la fonction EEPOnSignal_x() pour ce signal est appelée lorsque son état change. A utiliser avec précaution ! Le signal doit être enregistré et la fonction correspondante déclarée. (Voir page suivante). Une utilisation incorrecte de cette fonction peut entraîner des boucles infinies. • La fonction retourne 1 si le signal et l'aspect demandé existent. Elle renvoie 0 si l'un des deux n'a pu être trouvé.

```
-- Commute le signal 0023 sur 1 (l'état dépend du signal)
EEPSetSignal(23, 1)
-- Commute le signal 0045 sur 1 et appelle la fonction
   EEPOnSignal_45()
EEPSetSignal(45, 1, 1)
```

EEPGetSignal()	EEPGetSignal(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2
Objectif	Renvoie l'état actuel d'un signal
Commentaires	• L'argument est l'ID du signal. • La valeur retournée est une représentation numérique de l'état actuel du signal. La valeur correspond à la position de l'état du signal dans la liste d'effets de ses propriétés. • La valeur retournée est égale à 0 si le signal n'existe pas.

```
Etat_Courant = EEPGetSignal(1)
if Etat_Courant == 0 then
     print("Le signal 1 n'existe pas")
  elseifd Etat_Courant == 1 then
     print("Le signal 1 est défini sur Voie libre")
  elseifd Etat_Courant == 1 then
     print("Le signal 1 est défini sur Arrêt")
end
```

EEPRegisterSignal()	EEPRegisterSignal(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2

```
EEPRegisterSignal(1)

function EEPOnSignal_1(Nouvel_Etat)
  print("Signal 1 commuté sur ", Nouvel_Etat)
end
```

Objectif	Enregistre un signal pour la fonction de rappel EEPOnSignal_x() La condition de cet enregistrement empêche les signaux de déclencher un rappel lorsqu'aucune fonction appropriée n'a été déclarée.
Commentaires	• L'enregistrement est obligatoire pour les signaux pour lesquels vous souhaitez déclencher la fonction EEPOnSignal_x() chaque fois que l'état change. • L'argument passé à la fonction est l'ID du signal. • La valeur retournée est égale à 1 si le signal existe ou 0 si le signal n'existe pas.

EEPOnSignal_x()	EEPOnSignal_x(Nouvel_Etat)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Aucune
Requis	EEP 10.2 plug-in 2

```
EEPRegisterSignal(1)

function EEPOnSignal_1(Nouvel_Etat)
  print("Signal 1 commuté sur ", Nouvel_Etat)
end
```

Objectif	Chaque changement d'aspect induit par un contact ou par une commande manuelle (directement ou dans une séquence liée) déclenche cette fonction de rappel si le signal a été enregistré pour la fonction de rappel EEPRegisterSignal(ID). **Important** : Modifier cette fonction ou l'état d'un signal lié par la fonction Lua ne déclenchera pas le rappel, à moins que le troisième argument de cette fonction ne soit défini à 1.
Commentaires	• Le nom de la fonction ne doit pas se terminer par _x mais par l'ID du signal. Pour le signal 0012 par exemple, le nom correct de la fonction serait EEPOnSignal_12(). **Veuillez noter** : Les zéros en tête doivent être omis dans le nom de la fonction ! • La fonction est appelée avec le nouvel état du signal comme argument. Le numéro correspond à la position dans la liste des effets du signal telle qu'elle se trouve dans les propriétés de celui-ci. Utilisez une variable de votre choix pour stocker cette valeur. • EEP n' a pas besoin de valeur en retour pour appeler cette fonction.

EEPGetSignalTrainsCount()	EEPGetSignalTrainsCount(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 13.2 plug-in 2
Objectif	Renvoie le nombre de véhicules roulants arrêtés au signal spécifié.
Commentaires	• L'argument est l'ID du signal. • La valeur de retour est le nombre de trains, voitures, tramways, etc, à l'arrêt devant le signal.

```
EEPGetSignalTrainsCount(1)
```

EEPGetSignalTrainName()	EEPGetSignalTrainName(ID, Position)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13.2 plug-in 2
Objectif	Renvoie l'état actuel d'un signal
Commentaires	• Le premier argument est l'ID du signal. • Le deuxième argument est le numéro de position du train. voiture, tramway, etc, à l'arrêt devant le signal. • La valeur de retour est le nom du train. voiture, tramway à l'arrêt devant le signal.

```
EEPGetSignalTrainName(1, 1)
```

Fonctions de gestion des aiguillages

EEPSetSwitch()	EEPSetSwitch(ID, Direction, Callback)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux ou trois
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2
Objectif	Commute un aiguillage.
Commentaires	• Le premier argument est une valeur numérique représentant l'ID de l'aiguillage. • Le deuxième argument est l'état du point de commutation. • Si le nombre 1 est entré comme troisième argument (facultatif), la fonction EEPOnSwitch() pour cet aiguillage est appelée lorsque son état change. A utiliser avec précaution ! L'aiguillage doit être enregistré et la fonction correspondante déclarée. (Voir page suivante). Une utilisation incorrecte de cette fonction peut entraîner des boucles infinies. • La fonction retourne 1 si l'aiguillage et la direction demandée existent. Elle renvoie 0 si l'un des deux n'a pu être trouvé.

```
-- Aiguillage 0067 défini à 1 (principal)
EEPSetSwitch(67, 1)

-- Aiguillage 0089 défini à 1 et appelle EEPOnSwitch_89()
EEPSetSwitch(89, 1, 1)
```

EEPGetSwitch()	EEPSetSwitch(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2
Objectif	Renvoie l'état actuel d'un aiguillage.
Commentaires	• L'argument passé est une valeur numérique représentant l'ID de l'aiguillage. • Le deuxième argument est l'état du point de commutation. • La valeur retournée est une représentation numérique de l'état actuel de l'aiguillage. La valeur correspond à la position de l'état de l'aiguillage dans la liste d'effets de ses propriétés. • La valeur retournée est égale à 0 si l'aiguillage n'existe pas.

```
Direction_Actuelle = EEPGetSwitch(1)
if Direction_Actuelle == 0 then
     print("L'aiguillage ID 1 n'existe pas !")
  elseif Direction_Actuelle == 1 then
     print("L'aiguillage ID 1 est défini sur branche principale")
  elseif Direction_Actuelle == 2 then
     print("L'aiguillage ID 1 est défini sur Embranchement")
end
```

EEPRegisterSwitch()	EEPRegisterSwitch(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 10.2 plug-in 2

```
Direction _ Actuelle = EEPGetSwitch(1)
if Direction _ Actuelle == 0 then
     print("L'aiguillage ID 1 n'existe pas !")
  elseif Direction _ Actuelle == 1 then
     print("'aiguillage ID 1 est défini sur branche princi-
pale")
  elseif Direction _ Actuelle == 2 then
     print("'aiguillage ID 1 est défini sur Embranchement")
end
```

Objectif	Enregistre un aiguillage pour la fonction de rappel EEPOnSwitch_x(). La condition de cet enregistrement empêche les aiguillages de déclencher un rappel lorsqu'aucune fonction appropriée n'a été déclarée.
Commentaires	• L'enregistrement est obligatoire pour les aiguillages pour lesquels vous souhaitez déclencher la fonction EEPOnSwitch_x() chaque fois que la direction change. • L'argument passé à la fonction est l'ID de l'aiguillage. • La valeur retournée est égale à 1 si l'aiguillage existe ou 0 s'il n'existe pas.

EEPOnSwitch_x()	EEPOnSwitch _ x(Direction)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Aucune
Requis	EEP 10.2 plug-in 2

```
EEPRegisterSwitch(1)

function EEPOnSwitch _ 1(Direction)
  print("Aiguillage 1 commuté sur ", Direction)
end
```

Objectif	Chaque changement d'aspect induit par un aiguillage ou par une commande manuelle (directement ou dans une séquence liée) déclenche cette fonction de rappel si l'aiguillage a été enregistré pour la fonction de rappel EEPRegisterSwitch(ID). **Important** : Modifier cette fonction ou l'état d'un aiguillage lié par la fonction Lua ne déclenchera pas le rappel, à moins que le troisième argument de cette fonction ne soit défini à 1.
Commentaires	• Le nom de la fonction ne doit pas se terminer par _x mais par l'ID de l'aiguillage. Pour le l'aiguillage 0034 par exemple, le nom correct de la fonction serait EEPOnSwitch_34(). **Veuillez noter** : Les zéros en tête doivent être omis dans le nom de la fonction. • La fonction est appelée avec le nouvel état de l'aiguillage comme argument. Le numéro correspond à la position dans la liste des effets de l'aiguillage telle qu'elle se trouve dans les propriétés de celui-ci. Utilisez une variable de votre choix pour stocker cette valeur. • EEP n' a pas besoin de valeur en retour pour appeler cette fonction.

Fonctions d'enregistrement

EEPSaveData()	EEPSaveData(**Emplacement, Valeur|"Chaine"|Booléan|nil**)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.0

```
EEPSaveData(1, 42) -- Enregistre une valeur
EEPSaveData(2, " Emplacement 2") -- Enregistre une chaîne
EEPSaveData(3, true) -- Enregistre un booléan
EEPSaveData(4, nil) -- Supprime l'emplacement 4
```

Objectif

Stockage permanent des données.
Les données enregistrées sont stockées dans le même fichier de projet de l'utilisateur (*.anl3), mais ne sont pas visibles dans l'éditeur EEP Lua.
Cette fonction évite la perte de données lorsque le script est rechargé.

Commentaires

- Cette fonction fournit un mécanisme permettant au code Lua de stocker les données dans des emplacements numérotés de 1 à 1000.
- Chaque emplacement peut contenir une valeur numérique, chaîne ou booléenne. Les chaînes de caractères ne doivent pas contenir de caractères de formatage (guillemets par exemple).
- Le premier argument est l'ID de l'emplacement.
- Le deuxième argument concerne les données à stocker. Supprimez l'emplacement en lui attribuant la valeur nil.
- La valeur retournée est vraie lorsque l'enregistrement s'est déroulé avec succès (c'est-à-dire l'emplacement cible trouvé), sinon la valeur retournée est fausse.
- Chaque emplacement est écrit en utilisant la fonction EEPSaveData() et lu grâce à la fonction EEPLoadData(). La valeur actuelle de chaque emplacement sera également sauvegardée avec le programme Lua de l'utilisateur lors de la sauvegarde du projet. Ceci maintiendra la synchronisation entre la disposition du projet et les données. La section du script contenant ces données n'est pas visible dans l'éditeur Lua d'EEP.

EEPLoadData()	EEPLoadData(**Emplacement**)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.0

```
EEPSaveData(1, 42) -- Enregistre une valeur
EEPSaveData(2, " Emplacement 2") -- Enregistre une chaine
EEPSaveData(3, true) -- Enregistre un booléan
EEPSaveData(4, nil) -- Supprime l'emplacement 4
```

Objectif	Charge le contenu des données depuis l'emplacement spécifié. Utilisez cette fonction pour restaurer les données lorsque vous rechargez votre script.
Commentaires	• Fournit un moyen pour le code Lua de lire les données des emplacements numérotés de 1 à 1000. • Chaque emplacement peut contenir une valeur numérique, chaîne ou booléenne. Les chaînes de caractères ne doivent pas contenir de caractères de formatage (guillemets par exemple). • L'argument passé à la fonction est l'ID de l'emplacement. • La première valeur retournée est vraie si l'emplacement contient des données, sinon la valeur retournée est fausse. • La deuxième valeur retournée contient les données contenues dans l'emplacement. • Les données de l'emplacement sont transférées du script vers la mémoire lors du chargement d'un projet.

Fonctions de gestion des trains

EEPSetTrainSpeed()	EEPSetTrainSpeed("#Nom", Vitesse)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.0
Objectif	Fixe une vitesse cible pour un convoi ferroviaire.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le deuxième argument est la vitesse. Une valeur négative entraîne une marche arrière. • L'influence du signal courant est annulée. • La première valeur retourne vraie (**true**) si la vitesse cible est prise en compte, sinon fausse (**false**) dans le cas contraire.

```
EEPSetTrainSpeed("#Mon Train", 80)
```

EEPGetTrainSpeed()	EEPGetTrainSpeed("#Nom")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.0
Objectif	Détermine la vitesse réelle d'un convoi ferroviaire.
Commentaires	• L'argument est le nom complet du train en tant que chaîne de caractères. • La première valeur retourne vraie (**true**) si le convoi est en mouvement, sinon fausse (**false**) si le convoi est à l'arrêt. • La deuxième valeur retourne la vitesse du convoi.

```
m_Result, m_Data = EEPGetTrainSpeed("#VT98;001")
```

EEPSetTrainRoute()	EEPSetTrainRoute("#Nom", "Itinéraire")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Assigne un itinéraire à un convoi ferroviaire.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le second argument est l'itinéraire en tant que chaîne de caractères. • La valeur retournée est vraie (true) si le train spécifié et l'itinéraire indiqué existent, sinon la valeur retournée est fausse (false) si l'un des deux n'existe pas.

```
EEPSetTrainRoute("#mon Train", "mon Itineraire")
```

EEPGetTrainRoute()	EEPGetTrainRoute("#Nom")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.2 Plugin 2
Objectif	Retourne l'itinéraire d'un convoi ferroviaire.
Commentaires	• L'argument est le nom complet du train en tant que chaîne de caractères. • La première valeur retourne vraie (**true**) si l'itinéraire du convoi a pu être déterminé, sinon fausse (**false**) dans le cas contraire. • La deuxième valeur retourne l'itinéraire du convoi.

```
m_Result, m_Itineraire = EEPGetTrainRoute("#mon Train")
```

EEPSetTrainLight()	EEPSetTrainLight("#Nom", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Allume ou éteint les feux du train spécifié.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le second argument prend la valeur **true** (allumé) ou **faux** (éteint). • La valeur retournée est vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPSetTrainLight("#mon Train", true)
```

EEPSetTrainSmoke()	EEPSetTrainSmoke("#Nom", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Active ou désactive la production de fumée dans un convoi ferroviaire.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le deuxième argument est **true** = fumée activée ou **false** = fumée désactivée. • La valeur retournée est vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPSetTrainSmoke("#mon Train", true)
```

EEPSetTrainHorn()	EEPSetTrainHorn("#Nom", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Active ou désactive le son d'avertissement (sifflet, klaxon du train) spécifié.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le 2ème argument est **true** pour faire retentir le klaxon, sifflet... et **false** pour le couper. • La valeur retournée est vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPSetTrainHorn("#mon Train", true)
```

EEPSetTrainCouplingFront()	EEPSetTrainCouplingFront("#Nom", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Active ou désactive l'attelage avant d'un train.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le deuxième argument est **true** si l'attelage est accroché ou **false** si attelage est décroché. • La valeur retournée est vraie (**true**) si le train désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPSetTrainCouplingFront("#mon Train", true)
```

EEPSetTrainCouplingRear()	EEPSetTrainCouplingRear("#Nom", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Active ou désactive l'attelage arrière d'un train.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le deuxième argument est **true** si l'attelage est accroché ou **false** si attelage est décroché. • La valeur retournée est vraie (**true**) si le train désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPSetTrainCouplingRear("#mon Train", true)
```

EEPTrainLooseCoupling()	EEPTrainLooseCoupling("#Nom", true|false, Position)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Trois
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Sépare un convoi ferroviaire à la position indiquée.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le deuxième argument définit si vous comptez les véhicules à partir de l'avant ou de l'arrière (**true** = de l'avant, **false** = de l'arrière). • Le troisième argument détermine la position à partir de laquelle la séparation a lieu. • La valeur retournée est vraie (**true**) si le train et la position désignés existent, sinon la valeur retournée est fausse (**false**).

```
EEPTrainLooseCoupling("#mon Train", true, 3)
```

EEPSetTrainHook()	EEPSetTrainHook("#Nom", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Active ou désactive le crochet du wagon grue spécifié pour soulever des marchandises.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le deuxième argument prend la valeur **true** (accroché) ou **false** (décroché). • La valeur retournée est vraie (**true**) si le train désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPSetTrainHook("#mon Train", true)
```

EEPSetTrainAxis()	EEPSetTrainAxis("#Nom", "ElemMobi", Position)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Trois
Valeurs renvoyées	Une
Requis	EEP 11.2 Plugin 2
Objectif	Animation d'un élément mobile sélectionné dans une composition de train.
Commentaires	• Le premier argument spécifie le nom complet du train en tant que chaîne de caractères. • Le deuxième argument est le nom de l'élément mobile comme chaîne de caractères. • Le troisième argument définit la position cible de l'élément mobile. • La valeur retournée est vraie (**true**) si le train et l'élément mobile désignés existent, sinon la valeur retournée est fausse (**false**).

```
EEPSetTrainAxis("#mon Train", "Drehung nach links", 3)
```

Fonctions de gestion du matériel roulant

EEPRollingstockSetCouplingFront()	`EEPRollingstockSetCouplingFront("Nom", Etat_Coupleur)`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.0
Objectif	Active ou désactive l'attelage avant d'un matériel roulant.
Commentaires	• Le premier argument spécifie le nom complet du matériel roulant en tant que chaîne de caractères. • Le deuxième argument est l'état du couplage souhaité : 1 = Accroché, 2 = Décroché. • La valeur retournée est vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**).

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

EEPRollingstockGetCouplingFront()	`EEPRollingstockGetCouplingFront("Nom")`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.0
Objectif	Renvoie la position de l'attelage avant d'un matériel roulant.
Commentaires	• L'argument est le nom complet du matériel roulant en tant que chaîne de caractères. • La première valeur retourne vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**). • La deuxième valeur retourne la position du coupleur : 1 = prêt, 2 = décroché, 3 = accroché.

```
m_Result, m_Position = EEPRollingstockGetCouplingFront("Castor 1;001")
```

EEPRollingstockSetCouplingRear()	EEPRollingstockSetCouplingRear("Nom", Etat_Coupleur)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.0
Objectif	Active ou désactive l'attelage arrière d'un matériel roulant.
Commentaires	• Le premier argument spécifie le nom complet du matériel roulant en tant que chaîne de caractères. • Le deuxième argument est l'état du couplage souhaité : 1 = Accroché, 2 = Décroché. • La valeur retournée est vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**).

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

EEPRollingstockGetCouplingRear()	EEPRollingstockGetCouplingRear("Nom")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Une
Valeurs renvoyées	Deux
Requis	EEP 11.0
Objectif	Renvoie la position de l'attelage arrière d'un matériel roulant.
Commentaires	• L'argument est le nom complet du matériel roulant en tant que chaîne de caractères. • La première valeur retourne vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**). • La deuxième valeur retourne la position du coupleur : 1 = prêt, 2 = décroché, 3 = accroché.

```
m_Roulant = "fals 175 Kalk"
m_Result, m_Position = EEPRollingstockGetCouplingRear(m_Roulant)
```

EEPRollingstockSetAxis()	EEPRollingstockSetAxis("Nom", "ElemMobi", Position)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Trois
Valeurs renvoyées	Une
Requis	EEP 11.0

```
m_Train = "Bekohlungskranbrücke 1"
m_ElemMobi = "Drehung links"

m_Result = EEPRollingstockSetAxis(m_Train, m_ElemMobi, 50)
```

Objectif	Déplace l'élément mobile indiqué du matériel roulant spécifié à la position souhaitée.
Commentaires	• Le 1er argument est le nom complet du matériel roulant en tant que chaîne de caractères. • Le deuxième argument est le nom complet de l'élément en tant que chaîne de caractères. • Le troisième argument est la position cible de l'élément mobile. • La valeur retournée est vraie (**true**) si le matériel roulant désigné existe, sinon la valeur retournée est fausse (**false**).

EEPRollingstockGetAxis()	EEPRollingstockGetAxis("Nom", "ElemMobi")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Deux
Requis	EEP 11.0

```
m_Train = "Bekohlungskranbrücke 1"
m_ElemMobi = "Drehung links"

m_Result , m_Position = EEPRollingstockGetAxis(m_Train, m_ElemMobi)
```

Objectif	Renvoie la position actuelle d'un élément mobile du matériel roulant spécifié.
Commentaires	• Le 1er argument est le nom complet du matériel roulant en tant que chaîne de caractères. • Le deuxième argument est le nom complet de l'élément en tant que chaîne de caractères. • La première valeur retournée est vraie (**true**) si le matériel roulant et l'élément mobile désignés existent, sinon la valeur retournée est fausse (**false**) si au moins l'un d'entre eux n'existe pas. • La deuxième valeur retourne la position actuelle de l'élément en tant que nombre.

EEPRollingstockSetSlot()	EEPRollingstockSetSlot("Nom", Groupe)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.0

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

Objectif	Déplace tous les éléments mobiles du matériel roulant à la position stockée dans le groupe.
Commentaires	• Réglez d'abord tous les éléments mobiles à la position voulue et mémorisez-les dans l'un des 16 groupes (cf menu contextuel). Lorsque cette fonction est exécutée, tous les éléments passent de leurs positions actuelles à la position mémorisée dans le groupe. • Le 1er argument est le nom complet du matériel roulant en tant que chaîne de caractères. • Le deuxième argument est le numéro du groupe (sur les 16) dans lequel sont mémorisées les positions souhaitées des éléments mobiles. • La valeur retournée est vraie (**true**) si le matériel roulant désigné et le groupe existent, sinon la valeur retournée est fausse (**false**). Il n'est pas vérifié si le groupe contient déjà des réglages.

Fonctions de gestion des structures

EEPStructureSetSmoke()	EEPStructureSetSmoke("#Nom _ Lua", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.1 Plugin 1

```
EEPStructureSetSmoke("#1 _ Lauscha _ train station", true)
```

Objectif	Active ou désactive la fumée (par exemple la fumée d'une cheminée) d'une structure.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • Le deuxième argument est une valeur booléenne, vraie (**true**) pour activer l'émission de fumée ou fausse (**false**) pour la désactiver. • La valeur retournée est vraie (**true**) si la structure désignée existe et autorise la possibilité d'émettre de la fumée, sinon la valeur retournée est fausse (**false**).

EEPStructureGetSmoke()	EEPStructureGetSmoke("#Nom _ Lua")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.1 Plugin 1

```
Name = " #1 _ Lauscha _ train station"
hResult, hData = EEPStructureGetSmoke(Name)
```

Objectif	Indique si la fumée (par exemple, la fumée de cheminée) d'une structure est actuellement émise ou pas.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • La première valeur retournée est vraie (**true**) si la structure désignée existe et possède la caractéristique d'émettre de la fumée, sinon la valeur retournée est fausse (**false**). • La deuxième valeur retournée est vraie (**true**) si de la fumée est émise ou fausse (**false**) dans le cas contraire.

EEPStructureSetLight()	EEPStructureSetLight("#Nom _ Lua", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.1 Plugin 1
Objectif	Allume ou éteint les lumières d'une structure.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • Le deuxième argument est une valeur booléenne, vraie (**true**) pour allumer les lumières ou fausse (**false**) pour les éteindre. • La valeur retournée est vraie (**true**) si la structure désignée existe et autorise la gestion de la lumière ou de l'éclairage, sinon la valeur retournée est fausse (**false**).

```
EEPStructureSetLight("#1 _ Betriebsdienstgebaeude", true)
```

EEPStructureGetLight()	EEPStructureGetLight("#Nom _ Lua")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.1 Plugin 1
Objectif	Indique si les lumières d'une structure sont actuellement allumées ou éteintes.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • La première valeur retournée est vraie (**true**) si la structure désignée existe et possède la caractéristique d'émettre de la lumière, sinon la valeur retournée est fausse (**false**). • La deuxième valeur retournée est vraie (**true**) si les lumières sont allumées ou fausse (**false**) si elles sont éteintes ou en mode automatique.

```
m _ Nom = "#1 _ Betriebsdienstgebaeude"
hResult, hData = EEPStructureGetLight(m _ Nom)
```

EEPStructureSetFire()	EEPStructureSetFire("#Nom _ Lua", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.1 Plugin 1

```
EEPStructureSetFire("#1 _ Brandhaus _ 01 _ SB1", true)
```

Objectif	Active ou désactive la fonction d'incendie d'une structure.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • Le deuxième argument est une valeur booléenne, vraie (**true**) pour allumer un incendie ou fausse (**false**) pour les éteindre. • La valeur retournée est vraie (**true**) si la structure désignée existe et autorise la gestion des incendies, sinon la valeur retournée est fausse (**false**).

EEPStructureGetLight()	EEPStructureGetLight("#Nom _ Lua")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.1 Plugin 1

```
m _ Nom = "#1 _ Brandhaus _ 01 _ SB1"
hResult, hData = EEPStructureGetFire(m _ Nom)
```

Objectif	Indique si un incendie dans une structure est actuellement déclaré ou pas.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • La première valeur retournée est vraie (**true**) si la structure désignée existe et autorise la gestion des incendies, sinon la valeur retournée est fausse (**false**). • La deuxième valeur retournée est vraie (**true**) si un incendie est allumé ou fausse (**false**) si elles sont éteintes ou en mode automatique.

EEPStructureAnimateAxis()	EEPStructureAnimateAxis("Nom _ Lua", "ElemMobi", Position)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Trois
Valeurs renvoyées	Une
Requis	EEP 11.1 Plugin 1
Objectif	Déplace l'élément mobile spécifié de la structure ou de l'objet spécifié.
Commentaires	• Le premier argument est le nom Lua de la propriété sous forme de chaîne de caractères. Ceci diffère du nom du modèle par l'ID précédent du bien immobilier. • Le deuxième argument est le nom complet de l'élément mobile en tant que chaîne de caractères. • Le troisième argument est le nombre (positif ou négatif) de la position ou du pas pour le déplacement de l'élément mobile. Une valeur de 1000 ou -1000 démarre un mouvement sans fin si le modèle a été construit avec cette caractéristique (par exemple les ailes d'un moulin à vent). Une valeur définie à 0 arrête tout mouvement. • La valeur retournée est vraie (**true**) si la structure désignée et l'élément mobile existent, sinon la valeur retournée est fausse (**false**).

```
EEPStructureAnimateAxis("#1 _ Windmühle", "Muehlrad", 1000)
```

EEPStructureSetAxis()	EEPStructureSetAxis("Nom _ Lua", "ElemMobi", Position)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Trois
Valeurs renvoyées	Une
Requis	EEP 11.1 Plugin 1
Objectif	Règle l'élément mobile spécifié de la structure ou de l'objet vers une nouvelle position.
Commentaires	• Le premier argument est le nom Lua de la propriété sous forme de chaîne de caractères. Ceci diffère du nom du modèle par l'ID précédent du bien immobilier. • Le deuxième argument est le nom complet de l'élément mobile en tant que chaîne de caractères. • Le troisième argument définit la nouvelle position de l'élément mobile. • La valeur retournée est vraie (**true**) si la structure désignée et l'élément mobile existent, sinon la valeur retournée est fausse (**false**).

```
EEPStructureSetAxis("#1 _ Drehscheibe", "Brücke", 50)
```

EEPStructureGetAxis()	EEPStructureGetAxis("Nom _ Lua", "ElemMobi")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Deux
Requis	EEP 11.1 Plugin 1

```
m _ Result, m _ Position =
EEPStructureGetAxis("#1 _ Drehscheibe", "Brücke")
```

Objectif

Indique la position de l'élément mobile de la structure spécifiée ou de l'objet au sol.

Commentaires

- Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis.
- Le deuxième argument est le nom complet de l'élément mobile en tant que chaîne de caractères.
- La première valeur retournée est vraie (**true**) si la structure spécifiée et l'élément mobile existent, sinon la valeur retournée est fausse (**false**).
- La deuxième valeur retournée est la position actuelle de l'élément mobile spécifié.

EEPStructureSetPosition()	EEPStructureSetPosition("Nom _ Lua", PosX, PosY, PosZ)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Quatre
Valeurs renvoyées	Une
Requis	EEP 11.1 Plugin 1

```
EEPStructureSetPosition("#1 _ Strohballen", 1, 2, 3)
```

Objectif

Place la structure ou l'objet au sol spécifié à une nouvelle position.

Commentaires

- Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis.
- Le deuxième argument est la nouvelle position sur l'axe x de la structure.
- Le troisième argument est la nouvelle position sur l'axe y de la structure.
- Le quatrième argument est la nouvelle position sur l'axe z de la structure.
- Les structures ne peuvent pas être placées en dehors des limites d'un projet.
- La valeur retournée est vraie (**true**) si la structure désignée existe sinon la valeur retournée est fausse (**false**).
- La fonction peut également être utilisée pour les éléments de paysage parce que les deux, les structures et les éléments de paysage, partagent un groupe commun d'ID.

EEPStructureSetRotation()	`EEPStructureSetRotation("Nom _ Lua", RotX, RotY, RotZ)`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Quatre
Valeurs renvoyées	Une
Requis	EEP 11.1 Plugin 1
Objectif	Rotation de la structure ou de l'objet au sol spécifié vers une nouvelle position.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • Le deuxième argument est la nouvelle rotation sur l'axe x de la structure. • Le troisième argument est la nouvelle rotation sur l'axe y de la structure. • Le quatrième argument est la nouvelle rotation sur l'axe z de la structure. • La valeur retournée est vraie (**true**) si la structure désignée existe sinon la valeur retournée est fausse (**false**). • La fonction peut également être utilisée pour les éléments de paysage parce que les deux, les structures et les éléments de paysage, partagent un groupe commun d'ID.

```
EEPStructureSetRotation("#1 _ Strohballen", 0, 0, 25)
```

Fonctions de gestion des voies

Voies ferroviaires

EEPRegisterRailTrack()	`EEPRegisterRailTrack(ID)`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3
Objectif	Enregistrement d'un élément de voie ferrée pour les requêtes 'd'occupation'.
Commentaires	• L'argument est l'ID de l'élément de la voie ferrée. • La valeur retournée est vraie (**true**) si la voie désignée existe sinon la valeur retournée est fausse (**false**). • L'enregistrement est obligatoire pour la fonction EEPIsRailTrackReserved().

```
EEPRegisterRailTrack(1)
```

EEPIsRailTrackReserved()	`EEPIsRailTrackReserved(ID)`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.3 Plugin 3
Objectif	Indique si l'élément de voie spécifié est occupé par du matériel roulant.
Commentaires	• L'argument est l'ID de l'élément de la voie ferrée. • La première valeur retournée est vraie (**true**) si l'élément de voie désigné existe et a été enregistré avant la requête. • La deuxième valeur retournée est vraie (**true**) si l'élément de voie est occupé, sinon la valeur retournée est fausse (**false**). • N'oubliez pas de bien enregistrer la voie ferrée avant d'utiliser cette fonction !

```
m_Result, m_Data = EEPIsRailTrackReserved(1)
```

Voies routières

EEPRegisterRoadTrack()	EEPRegisterRoadTrack(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3
Objectif	Enregistrement d'un élément routier pour les requêtes "d'occupation"
Commentaires	• L'argument est l'ID de l'élément de la voie ferrée. • La valeur retournée est vraie (**true**) si la route désignée existe sinon la valeur retournée est fausse (**false**). • L'enregistrement est obligatoire pour la fonction EEPIsRoadTrackReserved().

```
EEPRegisterRoadTrack(1)
```

EEPIsRoadTrackReserved()	EEPIsRoadTrackReserved(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.3 Plugin 3
Objectif	Indique si l'élément routier spécifié est occupé par du matériel roulant.
Commentaires	• L'argument est l'ID de l'élément de la voie ferrée. • La première valeur retournée est vraie (**true**) si l'élément routier désigné existe et a été enregistré avant la requête. • La deuxième valeur retournée est vraie (**true**) si l'élément routier est occupé, sinon la valeur retournée est fausse (**false**). • N'oubliez pas de bien enregistrer l'élément routier avant d'utiliser cette fonction !

```
m_Result, m_Data = EEPIsRoadTrackReserved(1)
```

Voies de tramway

EEPRegisterTramTrack()	EEPRegisterTramTrack(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3
Objectif	Enregistrement d'un élément de voie de tramway pour les requêtes "d'occupation"
Commentaires	• L'argument est l'ID de l'élément de voie de tramway. • La valeur retournée est vraie (**true**) si la voie désignée existe sinon la valeur retournée est fausse (**false**). • L'enregistrement est obligatoire pour la fonction EEPIsTramTrackReserved().

```
EEPRegisterTramTrack(1)
```

EEPIsTramTrackReserved()	EEPIsTramTrackReserved(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.3 Plugin 3
Objectif	Indique si l'élément de voie de tramway spécifié est occupé par du matériel roulant.
Commentaires	• L'argument est l'ID de l'élément de voie de tramway. • La première valeur retournée est vraie (**true**) si l'élément de voie désigné existe et a été enregistré avant la requête. • La deuxième valeur retournée est vraie (**true**) si l'élément de voie est occupé, sinon la valeur retournée est fausse (**false**). • N'oubliez pas de bien enregistrer la voie de tramway avant d'utiliser cette fonction !

```
m_Result, m_Data = EEPIsTramTrackReserved(1)
```

Voies annexes (voies fluviales, routes aériennes, etc.)

EEPRegisterAuxiliaryTrack()	`EEPRegisterAuxiliaryTrack(ID)`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3
Objectif	Enregistre un élément de voie annexe pour les requêtes "d'occupation"
Commentaires	• L'argument est l'ID de l'élément de voie auxiliaire. • La valeur retournée est vraie (**true**) si la voie désignée existe sinon la valeur retournée est fausse (**false**). • L'enregistrement est obligatoire pour la fonction EEPIsAuxiliaryTrackReserved().

```
EEPRegisterAuxiliaryTrack(1)
```

EEPIsAuxiliaryTrackReserved()	`EEPIsAuxiliaryTrackReserved(ID)`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Deux
Requis	EEP 11.3 Plugin 3
Objectif	Indique si l'élément de voie auxiliaire spécifié est occupé par des éléments de transport.
Commentaires	• L'argument est l'ID de l'élément de voie de tramway. • La première valeur retournée est vraie (**true**) si l'élément de voie désigné existe et a été enregistré avant la requête. • La deuxième valeur retournée est vraie (**true**) si l'élément de voie est occupé, sinon la valeur retournée est fausse (**false**). • N'oubliez pas de bien enregistrer la voie auxiliaire avant d'utiliser cette fonction !

```
m_Result, m_Data = EEPIsAuxiliaryTrackReserved(1)
```

Contrôle de voie

EEPRegisterControlTrack()	EEPRegisterControlTrack(ID)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3
Objectif	Enregistrement d'un élément de contrôle de voie pour les requêtes "d'occupation"
Commentaires	• L'argument est l'ID de l'élément de contrôle à vérifier. • La valeur retournée est vraie (**true**) si l'élément de contrôle de voie désigné à tester existe sinon la valeur retournée est fausse (**false**). • L'enregistrement est obligatoire pour la fonction EEPIsControlTrackReserved().

```
EEPRegisterControlTrack(1)
```

EEPIsControlTrackReserved()	EEPIsControlTrackReserved(ID, (true))
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un (Opt. 2)
Valeurs renvoyées	Deux (Opt. 3)
Requis	EEP 11.3 Plugin 3 EEP 13.2 Plugin 2
Objectif	Indique si l'élément de contrôle de voie spécifié est occupé par du matériel roulant.
Commentaires	• L'argument est l'ID de l'élément de contrôle à vérifier. • A partir d'EEP 13.2 Plug-In 2, un deuxième argument **true** peut être donné en option, de sorte que la fonction renvoie le nom du train comme troisième valeur de retour. • La première valeur retournée est vraie (**true**) si l'élément de voie désigné à tester existe et a été enregistré avant la requête. • La deuxième valeur retournée est vraie (**true**) si l'élément de voie est occupé, sinon la valeur retournée est fausse (**false**). • La troisième valeur de retour est le nom du véhicules qui occupe l'itinéraire. S'il y a plusieurs véhicules sur la même voie, c'est le nom du véhicule le plus en avant qui est retourné. • N'oubliez pas de bien enregistrer l'élément de contrôle avant d'utiliser cette fonction !

```
m_Result, m_Data = EEPIsControlTrackReserved(1)
```

Fonctions de gestion pour les caméras

EEPSetCamera()	EEPSetCamera(Type, "Nom _ Camera")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3
Objectif	Active une position de caméra mémorisée.
Commentaires	• Le premier argument est le type de caméra : 0 = statique, 1 = dynamique, 2 = caméra mobile • Le deuxième argument est le nom de la caméra en tant que chaîne de caractères. • La valeur retournée est vraie (**true**) si la caméra désignée existe, sinon la valeur retournée est fausse (**false**).

```
EEPSetCamera(0,"Station")
```

EEPSetPerspectiveCamera()	EEPSetPerspectiveCamera(Position, "Nom _ du _ train")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3
Objectif	Active une caméra associée au train spécifié.
Commentaires	• Le premier argument est la position relative de la caméra et correspond à la sélection 1 à 9 1 = Coté gauche, 2 = Coté droit, ... 8 = Vue cabine, etc... • La valeur retournée est vraie (**true**) si le train et la caméra désignés existent, sinon la valeur retournée est fausse (**false**). • L'appel de cette caméra alors qu'elle est déjà active désactive le mode de suivi.

```
EEPSetPerspectiveCamera(1,"#Passenger train")
```

Fonctions de gestion des projets

EEPLoadProject()	`EEPLoadProject("Nom _ du _ Fichier _ projet _ EEP")`
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 3

```
EEPLoadProject("Tutorials\\Tutorial _ 54 _ LUA.anl3")
```

Objectif	Charge un autre projet à partir du sous-dossier spécifié 'Anlagen'.
Commentaires	• L'argument est le chemin d'accès (si nécessaire) à l'intérieur du dossier "Anlagen" et le nom du fichier, y compris le suffixe. anl3. • Séparer le caractère entre le nom du dossier et le nom du fichier par une double barre oblique inversée. • La valeur retournée est vraie (**true**) si le projet demandé existe, sinon la valeur retournée est fausse (**false**).

Fonctions de gestion pour les dépôts virtuels

EEPGetTrainFromTrainyard()	EEPGetTrainFromTrainyard(Depot, "TrainName", Num)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Trois
Valeurs renvoyées	Une
Requis	EEP 11.3 Plugin 2
Objectif	Envoyer un train spécifié à partir du dépôt virtuel désigné.
Commentaires	• Le premier argument est l'ID du dépôt du train. Vous pouvez trouver l'ID dans l'en-tête de la fenêtre des propriétés du dépôt. • Le deuxième argument est le nom complet du train en tant que chaîne de caractères. Si vous saisissez une chaîne vide, le train est spécifié par le troisième argument. • Le troisième argument est la position du train dans la liste du dépôt. Le nom du train dans le deuxième argument a priorité sur ce nombre et doit être une chaîne vide si vous voulez que ce nombre soit utilisé par la fonction. Cependant, un numéro doit toujours être renseigné. Entrez 0 si vous utilisez le nom du train dans le deuxième argument. • La valeur retournée est vraie (**true**) si le dépôt et le train désignés existent, que le train soit actuellement dans le dépôt et disponible ou non ! sinon la valeur retournée est fausse (**false**) si le dépôt n'existe pas ou si le train spécifié n'est pas répertorié dans ce dépôt.

```
EEPGetTrainFromTrainyard(1,"",1)
EEPGetTrainFromTrainyard(1,"#Rheingold",0)
```

EEPGetTrainyardItemsCount()	EEPGetTrainyardItemsCount(Depot)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 13.2 Plugin 2
Objectif	Renvoie le nombre de véhicules situés dans un dépôt virtuel.
Commentaires	• L'argument fourni comme paramètre est l'ID du dépôt virtuel. Vous pouvez obtenir le numéro ID du dépôt dans la fenêtre des propriétés du point de contact du dépôt. • La valeur retournée est le nombre de véhicules (trains, locomotives, voitures, etc) contenus dans un dépôt virtuel spécifié par son numéro.

```
EEPGetTrainyardItemsCount(1)
```

EEPGetTrainyardItemName()	EEPGetTrainyardItemName(Depot, Pos _ Liste)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13.2 Plugin 2
Objectif	Retourne le nom du véhicule à la position indiquée dans le dépôt spécifié.
Commentaires	• Le premier argument est l'ID du dépôt virtuel. • Le deuxième argument est la position du véhicule dans la liste. • La valeur retournée contient le nom du véhicule.

```
EEPGetTrainyardItemName(1, 2)
```

EEPGetTrainyardItemStatus()	EEPGetTrainyardItemStatus(Depot,"TrainName",Pos _ Liste)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Trois
Valeurs renvoyées	Une
Requis	EEP 13.2 Plugin 2
Objectif	Renvoie le statut d'un groupe de véhicules dans le dépôt.
Commentaires	• Le premier argument est l'ID du dépôt virtuel. • Le deuxième argument est le nom du train ou du véhicule. Si celui-ci est renseigné, Lua ignore le numéro de position dans la liste. • Le troisième argument est la position du véhicule dans la liste. Si une chaîne vide est spécifiée comme nom du train en deuxième position, alors la position du véhicule de la liste est prise en compte. Inversement, la position doit quand même obligatoirement être renseignée si un nom de véhicule est présent. Dans ce cas, veuillez mettre la valeur 0. • La valeur retournée est la situation du train ou du véhicule : 0 = Parti, 1 = Au dépôt.

```
EEPGetTrainyardItemStatus(1, "#Güterzug", 0)

ou

EEPGetTrainyardItemStatus(1, "", 3)
```

Fonctions de gestion des info-bulles

EEPChangeInfoStructure()	EEPChangeInfoStructure("Nom_Lua", "Text")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13
Objectif	Assigne un nouveau texte au texte de l'info-bulle d'une structure.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • Le deuxième argument est le nouveau texte. Utilisez \n pour les retours à la ligne. • La valeur retournée est vraie (**true**) si la structure désignée existe, sinon la valeur retournée est fausse (**false**). • La fonction peut également être utilisée pour les éléments de paysage parce que les deux, les structures et les éléments de paysage, partagent un groupe commun d'ID.

```
EEPChangeInfoStructure("#1", " Bonjour !")
```

EEPShowInfoStructure()	EEPShowInfoStructure("Nom_Lua", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13
Objectif	Active ou désactive le texte de l'info-bulle de la structure spécifiée.
Commentaires	• Le premier argument est le **nom Lua** de la structure en tant que chaîne. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID de la structure, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • Le deuxième argument est une valeur booléenne, **true** pour activer l'info-bulle ou **false** pour la désactiver. • La valeur retournée est vraie (**true**) si la structure désignée existe, sinon la valeur retournée est fausse (**false**). • La fonction peut également être utilisée pour les éléments de paysage parce que les deux, les structures et les éléments de paysage, partagent un groupe commun d'ID.

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

EEPChangeInfoSignal()	EEPChangeInfoSignal(ID, "Text")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13
Objectif	Assigne un nouveau texte pour l'info-bulle d'un signal.
Commentaires	• Le premier argument est l'ID du signal. • Le deuxième argument est le nouveau texte. Utilisez \n pour les retours à la ligne. • La valeur retournée est vraie (**true**) si le signal désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPChangeInfoSignal(1, " Bonjour !")
```

EEPShowInfoSignal()	EEPShowInfoSignal(ID, true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13
Objectif	Active ou désactive le texte de l'info-bulle du signal spécifié.
Commentaires	• Le premier argument est l'ID du signal. • Le deuxième argument est une valeur booléenne, **true** pour activer l'info-bulle ou **false** pour la désactiver. • La valeur retournée est vraie (**true**) si le signal désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPShowInfoSignal(1, true)
```

EEPChangeInfoSwitch()	EEPChangeInfoSwitch(ID, "Text")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13
Objectif	Assigne un nouveau texte pour l'info-bulle d'un aiguillage.
Commentaires	• Le premier argument est l'ID de l'aiguillage. • Le deuxième argument est le nouveau texte. Utilisez \n pour les retours à la ligne. • La valeur retournée est vraie (**true**) si l'aiguillage désigné existe, sinon la valeur retournée est fausse (**false**).

```
EEPChangeInfoSwitch(1, " Bonjour !")
```

EEPShowInfoSwitch()	EEPShowInfoSwitch(ID, true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Deux
Valeurs renvoyées	Une
Requis	EEP 13
Objectif	Active ou désactive le texte de l'info-bulle de l'aiguillage spécifié.
Commentaires	• Le premier argument est l'ID de l'aiguillage. • Le deuxième argument est une valeur booléenne, **true** pour activer l'info-bulle ou **false** pour la désactiver. • La valeur retournée est vraie (**true**) si l'aiguillage désigné existe, sinon la valeur retournée est fausse (**false**).

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

Fonctions de gestion affichage de texte via des modèles d'information

EEPShowInfoTextTop()	EEPShowInfoTextTop(r, g, b, sz, t, j, "Text")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Sept
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1

```
r = 1    -- Rouge
g = 1    -- Vert
b = 1    -- Bleu
sz = 1   -- Taille
t = 10   -- Durée
j = 1    -- Alignement
Text = 'Texte centré en haut pendant 10 secondes"
EEPShowInfoTextTop(r,g,b,sz,t,j,Text)
```

Objectif : Crée et affiche un texte d'information en haut de l'écran.

Commentaires

- Les trois premiers arguments déterminent la couleur.
- Le quatrième argument définit la taille du texte (de 0,5 à 2 fois). Une valeur de 1 donne une police de taille normale.
- Le cinquième argument détermine le temps d'affichage en secondes. Le temps minimum est de 5 secondes. Cela signifie qu'un texte est affiché pendant 5 secondes même si vous saisissez un 0
- Le sixième argument détermine l'alignement du texte :
  0 = justifié,
  1 = centré,
  2 = justifié à gauche,
  3 = justifié à droite
- Le septième argument est le texte à afficher sous forme de chaîne de caractères.
- La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

EEPShowInfoTextBottom()	EEPShowInfoTextBottom(r, g, b, sz, t, j, "Text")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Sept
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1

```
r = 1    -- Rouge
g = 1    -- Vert
b = 1    -- Bleu
sz = 1   -- Taille
t = 10   -- Durée
j = 1    -- Alignement
Text = 'Texte centré en bas pendant 10 secondes"
EEPShowInfoTextTop(r,g,b,sz,t,j,Text)
```

Objectif : Crée et affiche un texte d'information en bas de l'écran.

Commentaires

- Les trois premiers arguments déterminent la couleur.
- Le quatrième argument définit la taille du texte (de 0,5 à 2 fois). Une valeur de 1 donne une police de taille normale.
- Le cinquième argument détermine le temps d'affichage en secondes. Le temps minimum est de 5 secondes. Cela signifie qu'un texte est affiché pendant 5 secondes même si vous saisissez un 0
- Le sixième argument détermine l'alignement du texte :
  0 = justifié,
  1 = centré,
  2 = justifié à gauche,
  3 = justifié à droite
- Le Septième argument est le texte à afficher sous forme de chaîne de caractères.
- La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

EEPShowScrollInfoTextTop()	EEPShowScrollInfoTextTop(r, g, b, sz, t, j, sp, "Text")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Huit
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1

```
r = 1    -- Rouge
g = 1    -- Vert
b = 1    -- Bleu
sz = 1   -- Taille
t = 10   -- Durée
j = 1    -- Alignement
sp = 0.2   -- Vitesse
Text = 'Défilement du texte vers le haut pendant 20 se-
condes"
EEPShowInfoTextTop(r, g, b, sz, t, j, sp, Text)
```

Objectif

Crée et affiche un texte d'information défilant vers le haut de l'écran.

Commentaires

- Les trois premiers arguments déterminent la couleur.
- Le quatrième argument définit la taille du texte (de 0,5 à 2 fois). Une valeur de 1 donne une police de taille normale.
- Le cinquième argument détermine le temps d'affichage en secondes.
- Le sixième argument (pour l'alignement du texte) n'a aucun effet, mais est obligatoire. Veuillez entrer un numéro (par ex. 0).
- La valeur retournée est vraie (**true**) si la fonction a été exécutée avec succès.
- Le septième argument correspond à la durée du défilement pour le texte.
- Le huitième argument est le texte à afficher sous forme de chaîne de caractères.
- La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

EEPShowScrollInfoTextBottom()	EEPShowScrollInfoTextBottom(r, g, b, sz, t, j, sp, "Text")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Huit
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1

```
r = 1    -- Rouge
g = 1    -- Vert
b = 1    -- Bleu
sz = 1   -- Taille
t = 10   -- Durée
j = 1    -- Alignement
sp = 0.2   -- Vitesse
Text = 'Défilement du texte vers le bas pendant 20 se-
condes"
EEPShowScrollInfoTextBottom(r, g, b, sz, t, j, sp, Text)
```

Objectif

Crée et affiche un texte d'information défilant vers le bas de l'écran.

Commentaires

- Les trois premiers arguments déterminent la couleur.
- Le quatrième argument définit la taille du texte (de 0,5 à 2 fois). Une valeur de 1 donne une police de taille normale.
- Le cinquième argument détermine le temps d'affichage en secondes.
- Le sixième argument (pour l'alignement du texte) n'a aucun effet, mais est obligatoire. Veuillez entrer un numéro (par ex. 0).
- La valeur retournée est vraie (**true**) si la fonction a été exécutée avec succès.
- Le septième argument correspond à la durée du défilement pour le texte.
- Le huitième argument est le texte à afficher sous forme de chaîne de caractères.
- La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

EEPHideInfoTextTop()	EEPHideInfoTextTop()
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Aucun
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1
Objectif	Cache le texte d'information en haut de l'écran.
Commentaires	• Cette fonction ne nécessite aucun argument. • La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

EEPHideInfoTextBottom()	EEPHideInfoTextBottom()
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Aucun
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1
Objectif	Cache le texte d'information en bas de l'écran.
Commentaires	• Cette fonction ne nécessite aucun argument. • La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

Fonctions de gestion du son

EEPPlaySound()	EEPPlaySound("Fichier")
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1
Objectif	Lecture d'un fichier son à partir de n'importe quel emplacement.
Commentaires	• L'argument est le chemin (relatif au dossier Resourcen/Sounds/EEXP) et le nom du fichier sous forme de chaîne de caractères. • La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

```
EEPPlaySound("User/Bimmel1.wav")
```

EEPStructurePlaySound()	EEPStructurePlaySound("Lua _ Name", true ou false)
Type	Fonction
Utilisé dans	Script
Défini dans	EEP
Paramètre(s)	Un
Valeurs renvoyées	Une
Requis	EEP 13 Plug-In 1
Objectif	Active ou désactive le son d'un modèle sonore de la catégorie Elements de paysage/Sons. Si cette tonalité doit être contrôlée exclusivement par Lua, il est conseillé de régler la distance d'activation dans le modèle sur 0.
Commentaires	• Le premier argument est le nom Lua du modèle sonore en tant que chaîne de caractères. Vous pouvez aussi définir uniquement le signe # précédé avec l'ID du modèle, car cette désignation est suffisante comme nom Lua et tout ce qui suit peut être omis. • La valeur de retour est vraie (**true**) si la fonction a été exécutée avec succès.

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

8. Conclusion

Traduction française : Domi
https://www.eep-france.net/