# Le format du fichier route**.csv**

➟ Référence rapide

⚠ Encore en construction

Les commandes qui ont encore besoin de documentation : **2**

## ■ Contenus



## ■ 1. Aperçu général

Un itinéraire CSV permet de créer un itinéraire dans un fichier texte.

Le fichier est un fichier texte codé dans n'importe quel encodage arbitraire, cependant, UTF-8 avec une marque d'ordre d'octet est le choix préféré. Le modèle d'analyse pour les nombres est **libre** (sauf mention contraire), cependant, vous êtes encouragés à produire une sortie néanmoins stricte. Le fichier doit être situé dans un répertoire dont le dossier courant ou parent comprend les dossiers du chemin de fer et du train. Le nom du fichier est arbitraire, mais doit avoir l'extension **.csv**. Le fichier est interprété sur une base de ligne par ligne, de haut en bas, où chaque ligne est divisée en expressions, qui sont interprétées de gauche à droite.

Le fichier route consiste en une série de commandes pour définir les objets qui sont utilisés tout au long de l'itinéraire (Structure namespace). Les propriétés supplémentaires pour la route, le train à utiliser par défaut et les images de fond à utiliser peut aussi être défini. Enfin, le fichier route contient des instructions de namespace pour la voie. Ici, les positions de la voie (habituellement en mètres) sont utilisés pour définir le moment où la voie devrait être courbe, où les stations sont placées, quand un mur doit commencer et finir, et ainsi de suite. De manière générale, les instructions de Track namespace doivent être utilisée après l'utilisation de toute les autres instructions namespace.

Le format suppose implicitement un rail 0 qui ne peut pas être explicitement commencé ou terminé. Au lieu de

cela, il est présent du début de la route à la fin, et il indique le rail sur lequel le joueur conduit le train. Tous les autres rails du format CSV sont purement visuels et n'ont pas de but fonctionnel.

Géométriquement, vous pouvez courber et monter l'implicite rail 0, tandis que tous les autres rails sont définis par rapport au rail 0 et suit le rail 0 dans les changements de courbes et de hauteur. À moins surchargée, le format du fichier est construit autour d'une taille de bloc fixe de 25 mètres de longueur, et il n'est possible que pour certaines commandes d'être utilisé sur les limites des 25 mètres du bloc. Le placement des objets suppose toujours un système de coordonnées non-courbe qui relie les blocs de manière linéaire.

➟ Voir aussi la référence rapide pour route CSV ...

## ■ 2. Syntaxe

Pour chaque ligne du fichier, des espaces blancs au début et à la fin de cette ligne sont ignorés. Ensuite, les lignes sont divisées en expressions individuelles, séparées par des virgules (U+002C). Ainsi, chaque ligne est de la forme suivante :

$Expression_1$, $Expression_2$, $Expression_3$, ..., $Expression_n$

À son tour, chaque expression peut avoir n'importe quelle formes suivantes :

● Commentaires

Un commentaire est complètement ignorée par l'analyseur. Pour créer un commentaire, l'expression doit commencer par un point-virgule (U+003B).

● Positions et longueurs des voies

*Position*

Un nombre non-négatif stricte (à point flottant) correspondant à une position de la voie. Toutes les commandes ultérieures de Track namespace sont associés à cette position de voie.

$Part_1$:$Part_2$:...:$Part_n$

C'est une façon plus complexe de préciser les positions de la voie pour une utilisation en conjonction avec Options.UnitOfLength. Chacun des $Part_i$ est un nombre stricte (à point flottant) . $Part_1$ sera multiplié par $Factor_1$, $Part_2$ avec $Factor_2$, et ainsi de suite, alors tous les produits sont ajoutés pour former la position de la fin de la voie. Cette position de la voie doit être non négatives. Les parties sont séparées par des virgules (U+003A). S'il vous plaît consulter Options.UnitOfLength pour plus d'informations sur la façon de définir les facteurs.

Partout où les arguments de commandes représentent des longueurs, ils peuvent aussi être entré en utilisant la notation deux points. Ces cas seront surlignés en vert dans la suite.

Quand les unités *n* sont définie via Options.UnitOfLength, mais moins de paramètres sont donnés en utilisant la notation deux points, les paramètres sont droit associatif, ce qui signifie, les paramètres sur la gauche seront ignorées. Par conséquent, chacune des longueurs suivantes sont équivalentes : *0:0:2*, *0:2*, and *2*.

● Commandes

Commandes sans arguments :

*NameOfTheCommand*

Commandes avec arguments :

```
NameOfTheCommand Argument1;Argument2;Argument3;...;Argumentn
NameOfTheCommand(Argument1;Argument2;Argument3;...;Argumentn)
```

Commandes avec indices et arguments :

```
NameOfTheCommand(Index1;Index2;...;Indexm)
Argument1;Argument2;Argument3;...;Argumentn
NameOfTheCommand(Index1;Index2;...;Indexm).Suffix
Argument1;Argument2;Argument3;...;Argumentn
NameOfTheCommand(Index1;Index2;...;Indexm).Suffix(Argument1;Argume
nt2;Argument3;...;Argumentn)
```

Règles :

*NameOfTheCommand* est insensible à la casse. Indices et arguments sont séparés par des virgules (U+003B). Les espaces blancs autour de *NameOfTheCommand* et chacun des indices et arguments sont ignorés. Les espaces blancs entourant chacune des parenthèses sont ignorés.

Si des indices sont utilisés, ils sont enfermés par des parenthèses d'ouverture (U+0028) et des parenthèses de fermeture (U+0029).Au moins un argument, ou un *Suffixe* est obligatoire lors de l'utilisation des indices.

Il ya deux variantes sur la façon de coder les arguments. Sauf pour les $-directives ($Chr, $Rnd, $Sub, ...), vous pouvez librement choisir quelle variante utiliser. Variante 1 : Le premier argument est séparé de la commande, des indices ou du *suffixe* par au moins un espace (U+0020). Variante 2 : Les arguments sont entourées par des parenthèses d'ouverture (U+0028) et des parenthèses de fermeture (U+0029). Dans ce dernier cas, le *suffixe* est obligatoire lorsqu'il est utilisé en conjonction avec les indices. Les espaces blancs entourant chacune des parenthèses sont ignorés.

S'il vous plaît noter que pour certaines commandes, le *suffixe* est obligatoire quel que soit le style que vous utilisez pour encoder les arguments. Dans la suite, le suffixe sera en gras lorsqu'il est obligatoire, et grisé quand il est facultatif.

### ● L'instruction **With**

```
With Prefix
```

Toutes les commandes ultérieures qui commencent par un point (U+002E) sont précédés par des *préfixes*. Par exemple :

▶
```
With Route
.Gauge 1435
.Timetable 1157_M
```

Est équivalent à :

▶
```
Route.Gauge 1435
Route.Timetable 1157_M
```

## ■ 3. Prétraitement

Avant que l'une des commandes dans le fichier route soit réellement interprétés, les expressions sont d'abord traitées. La première chose qu'il fait est de remplacer toutes les occurrences de $-directives dans une expression de droite à gauche. Les directives $Chr, $Rnd et $Sub ne peuvent être imbriquées en aucune façon, d'ailleurs $Include, $If, $Else et $EndIf ne doivent pas apparaître à l'intérieur d'une autre directive.

⚠ La syntaxe de $-directives ne peut être choisi librement, mais doit respecter les formes présentées ci-dessous.

---

```
$Include(File)
$Include(File:TrackPositionOffset)
$Include(File_1; Weight_1; File_2; Weight_2; ...)
```

***File***$_i$: Un fichier à inclure du même format (CSV / RW) que le fichier principal.

***Weight***$_i$: Un nombre positif (à point flottant) donnant une importance probable au fichier correspondant.

Cette directive choisit au hasard dans les fichiers spécifiés en fonction de leurs probabilités associées et inclut le contenu du fichier sélectionné dans le fichier principal. Le contenu est copié à la place de la directive $Include, ce qui signifie que vous devez prendre soin de suivre les positions et le dernier With statement utilisée, par exemple. Si le dernier argument important dans la séquence est omis, il est traité comme 1.

La directive $Include est utile pour scinder un gros fichier en fichiers plus petits, pour le partage des sections communes de code entre plusieurs itinéraires, et de choisir au hasard parmi un plus grand bloc de code. S'il vous plaît noter que les fichiers inclus peuvent eux-mêmes inclure d'autres fichiers, mais vous devez vous assurer qu'il n'y a pas de dépendances circulaires, par exemple le fichier A incluant le fichier B, et le fichier B incluant le fichier A, etc. Vous devez utiliser une extension de fichier différent de .csv pour les fichiers inclus afin que les utilisateurs ne puissent pas les sélectionner accidentellement dans le menu principal (sauf si cela est souhaité).

Si n'importe quel *File*$_i$ est suivie par :*TrackPositionOffset*, alors toutes les expressions dans le fichier inclus sont décalés de la position de la voie spécifiée **en mètres.** Par exemple, $Include(stations.include:2000) décale tous les positions de voie du fichier part.include de 2000 mètres avant de les inclure. Il est important de comprendre que "track positions" ne sont réellement comprise qu'après que $-directives aient été traitées, de sorte que toutes les expressions dans le fichier inclus sont simplement signalé être décalé plus tard ils forment ensuite la position de la voie. Cela signifie que si le fichier inclus contient des expressions telles que 1$Rnd(2;8)00, elles sont décalées, aussi, même si à ce stade, elles ne forment pas encore les positions de la voie.

⚠ La caractéristique offset de la position de la voie est uniquement disponible dans la version de développement 1.2.11 et au-dessus.

---

```
$Chr(Ascii)
```

***Ascii***: Un nombre entier dans l'intervalle de 20 à 127, ou 10 ou 13, correspondant à un caractère ASCII du même code.

Cette directive est remplacé par le caractère ASCII représenté par *Ascii*. Ceci est utile si vous voulez inclure des caractères qui font partie des règles de syntaxe ou ils seraient ignorés. Une liste de caractères pertinents :

| Code | Signification | Caractère |
|---|---|---|

| | | |
|---|---|---|
| 10 | Nouvelle ligne | *newline* |
| 13 | Nouvelle ligne | *newline* |
| 20 | Espace | *space* |
| 40 | Ouvre une parenthèse | ( |
| 41 | Ferme une parenthèse | ) |
| 44 | Virgule | , |
| 59 | Point virgule | ; |

La séquence $Chr(13)$Chr(10) est traitée comme un saut de ligne unique. L'insertion de $Chr(59) peut être interprété comme un début de commentaire ou comme un séparateur d'arguments, selon l'endroit où il est utilisé.

---

`$Rnd(Start; End)`

***Start***: Un entier représentant la limite inférieure.
***End***: Un entier représentant la limite supérieure.

Cette directive est remplacé par un entier aléatoire dans l'intervalle de *Start* vers *End*. C'est utile pour ajouter un itinéraire aléatoire.

*Exemple pour l'utilisation de la directive $Rnd :*

```
10$Rnd(3;5)0, Track.FreeObj 0; 1
```

Le résultat possible de l'exemple précédent est exactement une de ces lignes :

```
1030, Track.FreeObj 0; 1
1040, Track.FreeObj 0; 1
1050, Track.FreeObj 0; 1
```

---

`$Sub(Index) = Expression`

***Index***: Un entier non négatif représentant l'index d'une variable.
***Expression***: L'expression stocker dans la variable.

Cette directive ne devrait apparaître comme une expression unique. Elle est utilisée pour assigner *Expression* à une variable identifiée par *Index*. L'ensemble de directive $Sub est remplacée par une chaîne vide lorsque l'affectation est faite. Il est utile de stocker des valeurs obtenues par la directive $Rnd afin de réutiliser les mêmes valeurs généré aléatoirement. Voir la définition suivante de la directive $Sub(*Index*) par exemples.

⚠ **Note d'implémentation :** Alors qu'il est aussi possible de stocker des chaînes non numériques, il n'est pas possible d'inclure des virgules via $Chr(44) et les faire travailler comme un séparateur de déclaration. Cependant, il est possible de mettre un point-virgule comme premier caractère via $Chr(59) et le faire fonctionner comme un commentaire. Mais Pour les expressions conditionnelles, vous devriez utiliser $Include ou $If.

---

`$Sub(Index)`

***Index***: Un entier non négatif représentant l'index de la variable à récupérer.
Cette directive est remplacé par le contenu de l'index de la variable. La variable doit avoir été assigné avant de le récupérer.

*Exemple pour l'utilisation des directives $Sub(Index)=Value et $Sub(Index) :*

```
$Sub(0) = $Rnd(3; 5)
1000, Track.FreeObj $Sub(0); 47
1020, Track.FreeObj $Sub(0); 47
1040, Track.FreeObj $Sub(0); 47
```

Dans l'exemple précédent, tous les trois commandes Track.FreeObj sont garantis d'utiliser la même valeur généré aléatoirement afin de placer un objet libre sur le même rail. Si vous avez inséré la directive $Rnd(3;5) dans chacune des trois lignes individuellement, les objets pourraient être placés sur des rails différent à chaque fois.

---

## $If(*Condition*)

***Condition***: Un nombre qui représente **false** si zéro, et autrement **true**.
La directive $If permet de ne traiter que les expressions suivantes si la condition spécifiée est évaluée à vrai (tout nombre non nul). $If doit être suivie par $EndIf dans n'importe quelle dernière expression. Éventuellement, des directives $Else peuvent apparaître entre $If et $EndIf.

---

## $Else()

La directive $Else permet de ne traiter que les expressions suivantes si la condition de la précédente directive $If est évalue à faux (zero).

---

## $EndIf()

La directive $EndIf marque la fin du bloc $If qui a été commencé auparavant.

*Exemple de $If, $Else et $EndIf*

```
$Sub(1) = 0
With Track
$If($Sub(1))
    1020, .FreeObj 0; 1
    1040, .FreeObj 0; 2
$Else()
    1030, .FreeObj 0; 3
$EndIf()
```

*Un autre exemple de $If, $Else et $EndIf*

```
With Track
1050
$If($Rnd(0;1)), .FreeObj 0; 4, $Else(), .FreeObj 0; 5, $EndIf()
```

Il est possible d'imbriquer des blocs $If. Si vous placez $If/$Else/$EndIf sur des lignes séparées, vous pouvez employer l'indentation pour améliorer la lisibilité de la structure de bloc (comme dans le premier exemple).

---

**Finallement**, après que $-directives aient été traitées, toutes les expressions du fichier route sont triés en fonction de leurs positions associée à la voie.

*Exemple d'un fichier route partiel :*

```
1000, Track.FreeObj 0; 23
1050, Track.RailType 0; 1
10$Rnd(3;7)0, Track.Rail 1; 4
Track.FreeObj 1; 42
```

*Prétraitement de la directive $Rnd (en supposant que 3 est produit) :*

```
1000, Track.FreeObj 0; 23
1050, Track.RailType 0; 1
1030, Track.Rail 1; 4
Track.FreeObj 1; 42
```

*Le tri par les positions associé à la voie :*

```
1000, Track.FreeObj 0; 23
1030, Track.Rail 1; 4
Track.FreeObj 1; 42
1050, Track.RailType 0; 1
```

## 4. Les Options namespace

Les commandes à partir de namespace fournir des options génériques qui affectent la façon dont les autres commandes seront traitées. Vous devriez utiliser les commandes à partir de ce namespace avant de faire usage de commandes à partir d'autres namespaces.

---

**Options.UnitOfLength** *FactorInMeters*$_1$; *FactorInMeters*$_2$; *FactorInMeters*$_3$; ...; *FactorInMeters*$_n$

***FactorInMeters***$_i$: Un nombre (à point flottant) représentant combien de mètres on souhaite pour l'unité. La valeur par défaut est 1 pour *FactorInMeters*$_1$, et 0 pour tous les autres.

Cette commande peut être utilisée pour spécifier l'unité de longueur utilisé pour d'autres commandes. Lorsque vous entrez une position générique de voie de la forme *Part*$_1$:*Part*$_2$:...:*Part*$_n$, chacun des *Part*$_i$ sera multipliée par les *FactorInMeters*$_i$ correspondants, alors tous ces produits sont ajoutés pour former une seule position de voie représentée en mètres. Idéalement, vous devriez régler la longueur du bloc à un multiple entier d'une des unités que vous utilisez via Options.BlockLength.

Exemples de facteurs de conversion :

| Unité souhaitée | Facteur de conversion |
|---|---|
| mile | 1609.344 |
| chain | 20.1168 |
| mètre | 1 |
| yard | 0.9144 |

| | |
|---|---|
| foot | 0.3048 |

Par la suite, tous les arguments de toutes les commandes sont surlignées en vert quand ils sont touchés par Options.UnitOfLength.

---

### `Options.UnitOfSpeed` *FactorInKmph*

***FactorInKmph***: Un nombre (à point flottant) qui représente combien de kilomètres heures a l'unité souhaitée. La valeur par défaut est 1.

Cette commande peut être utilisée pour spécifier l'unité de vitesse utilisé pour d'autres commandes. Exemples de facteurs de conversion :

| Unité souhaitée | Facteur de conversion |
|---|---|
| mètres/seconde | 3.6 |
| miles/hour | 1.609344 |
| kilomètres/heure | 1 |

Par la suite, tous les arguments de toutes les commandes seront surlignés en bleu quand ils sont touchés par Options.UnitOfSpeed.

---

### `Options.BlockLength` *Length*

***Length***: Un nombre positif (à point flottant) qui représente la longueur d'un bloc, par défaut mesurée en mètres. La valeur par défaut est de 25 mètres.

Cette commande peut être utilisée pour spécifier la longueur d'un bloc. Ce paramètre est global et ne peut être changé au milieu de la route. *Length* est seulement exprimée dans l'unité spécifiée par Options.UnitOfLength si Options.UnitOfLength est utilisé avant Options.BlockLength.

---

### `Options.ObjectVisibility` *Mode*

***Mode*** : Le mode déterminant la manière dont les objets apparaissent et disparaissent. La valeur par défaut est 0.

▸ Les options disponibles pour *Mode*:

**0**: Héritage : Un objet est rendu invisible dès la fin du bloc où il réside est dépassé par la caméra. Cela ne fonctionne pas bien quand la caméra tourne autour. Auto-intersection de voie (par exemple des boucles) n'est pas possible.
**1**: Voie de base : Un objet est rendu invisible dès que sa fin est dépassé par la caméra. Ceci est mesuré dans les positions voie. Auto-intersection de voie (par exemple des boucles) n'est pas possible.

---

### `Options.SectionBehavior` *Mode*

***Mode***: Le mode déterminant la manière dont la commande Track.Section est traitée.

▸ Les options disponibles pour *Mode* :

**0**: Défaut : Dans Track.Section $Aspect_0$; $Aspect_1$; ...; $Aspect_n$, n'importe quel $Aspect_i$ se réfèrent à l'aspect que doit avoir la section si les sections *i* suivantes sont claires.
**1**: Simplifié : Dans Track.Section $Aspect_0$; $Aspect_1$; ...; $Aspect_n$, la section porte le plus petit aspect qui est plus élevé que celui de la section suivante.

---

### `Options.CantBehavior Mode`

***Mode***: Le mode déterminant comment cant dans la commande Track.Curve est traitée.

▸ Les options disponibles pour *Mode*:

**0**: Unsigned: Le paramètre *CantInMillimeters* dans Track.Curve n'a pas de signe, c'est à dire le signe est ignoré et le dévers est toujours vers le centre de courbe (vers l'intérieur). Cant ne peut pas être appliqué sur une voie rectiligne.
**1**: Signed: Le paramètre *CantInMillimeters* dans Track.Curve a un signe, à savoir les valeurs négative pour l'extérieur et positive pour l'intérieur de la voie courbe. Cant peut être appliqué sur une voie rectiligne, où la valeurs négatives pour la gauche et positives pour la droite.

---

### `Options.FogBehavior Mode`

***Mode***: Le mode déterminant comment la commande Track.Fog est traitée. La valeur par défaut est 0.

▸ Les options disponibles pour *Mode* :

**0**: Bloc de base : La couleurs et les plages du brouillard sont interpolées à partir du début du bloc où Track.Fog est utilisé avec les anciens paramètres à la fin du bloc avec les nouveaux paramètres.
**1**: Interpolée : La couleurs et les plages du brouillard sont interpolées entre les commandes Track.Fog adjacentes. Ce comportement imite Track.Brightness.

## ■ 5. La Route namespace

---

Les commandes de ce namespace définissent les propriétés générale de la route.

---

### `Route.Comment Text`

***Text*** : Les commentaires de la route qui apparaîtront dans la boite de dialogue de la route sélectionnée.

⚠ Vous devrez utiliser les directives $Chr si vous souhaitez inclure des sauts de ligne, des virgules et autres dans le texte.

---

### `Route.Image File`

***File*** : Le nom du fichier de l'image qui apparaît dans la boîte de dialogue de la route sélectionnée, relatif au dossier de l'itinéraire.

---

## `Route.Timetable` *Text*

***Text***: Le texte qui apparaît en haut du Timetable (Horaire) par défaut.

⚠ Vous devrez utiliser les directives $Chr si vous souhaitez inclure des sauts de ligne, des virgules et autres dans le texte.

---

## `Route.Change` *Mode*

***Mode***: Le mode de système de sécurité du train au démarrage. La valeur par défaut est une implémentation spécifique.

▸ Les options disponibles pour *Mode* :

**-1**: Le système de sécurité est **activé** et les freins de *service* sont appliqués.
**0**: Le système de sécurité est **activé** et les freins d'**urgence** sont appliqués.
**1**: Le système de sécurité est *désactivé* et les freins d'**urgence** sont appliqués.

---

## `Route.Gauge` *ValueInMillimeters*

***ValueInMillimeters***: Un nombre (à point flottant) représentant l'écartement des rails, measuré en **millimètres** (0.001 mètres). La valeur par défaut est 1435.

☀ *Note:*
Route.Gauge est identique à Train.Gauge.

---

## `Route.Signal(`*AspectIndex*`).Set` *Speed*

***AspectIndex***: Un nombre (à point flottant) représentant l'aspect du signal. L'aspect de 0 correspond au rouge.
***Speed***: Un nombre (à point flottant) représentant la vitesse autorisée pour cet aspect, **par défaut** measuré en **kilomètres par heures** (km/h).
Utilisez cette commande pour associer les limites de vitesse aux aspects du signal. L'aspect 0 représente un signal rouge, des valeurs plus élevées représentent des aspects plus permissifs. Les valeurs par défaut (y compris pour les signaux japonais) sont les suivantes :

| *Index* | Aspect | Vitesse |
|---|---|---|
| 0 | ● | 0 km/h |
| 1 | ●● | 25 km/h |
| 2 | ● | 55 km/h |
| 3 | ●● | 75 km/h |
| 4 | ● | *illimité* |
| 5 | ●● | *illimité* |

---

**Route.RunInterval** *Interval*$_0$; *Interval*$_1$; ...; *Interval*$_{n-1}$

***Interval***$_i$ : Un nombre (à point flottant) représentant l'intervalle de temps entre les horaires du train du joueur et celle d'autres trains qui seront créés, mesuré en **secondes**. Les valeurs positives indiquent un train parti plus tôt, les nombres négatifs un train qui partira plus tard.

Cette commande crée un ou plusieurs trains précédant ou suivant. Ces autres trains sont visibles, pleinement opérationnels, et utilisent le même train que celui du joueur. Les autres trains suivent le même horaire que le joueur, mais sont décalées dans le temps par *Interval*$_i$. Via la commande Track.Sta, vous pouvez également définir les stations où seul le joueur ou seulement les autres trains doivent s'arrêter. Les trains supplémentaires n'apparaissent seulement une fois que la section dans laquelle ils sont placés a été vidée des autres trains, mais le train du joueur est introduit indépendamment de l'état de la section de signalisation actuelle. Par conséquent, vous devriez vous assurer que d'autres trains ont quitté la région où le train du joueur apparaîtra au moment ou le scénario commence.

☀ *Note:*
Route.RunInterval est identique à Train.Interval.

---

**Route.AccelerationDueToGravity** *Value*

***Value***: Un nombre positif (à point flottant) représentant l'accélération de la pesanteur en **metres par seconde carrée** (m/s²). La valeur par défaut est 9.80665.

---

**Route.Elevation** *Height*

***Height***: Un nombre (à point flottant) représentant la hauteur initiale au-dessus du niveau des mers, **par défaut** measuré en **metres** (m). La valeur par défaut est 0.

---

**Route.Temperature** *ValueInCelsius*

***Value***: Un nombre (à point flottant) représentant la température initiale en **Celsius**. La valeur par défaut est 20.

---

**Route.Pressure** *ValueInKPa*

***ValueInKPa*** : Un nombre (à point flottant) représentant la pression d'air initiale en **kPa** (1000 Pascal). La valeur par défaut est 101.325.

---

**Route.AmbientLight** *RedValue; GreenValue; BlueValue*

***RedValue***: Un nombre entier allant de 0 à 255 représentant la composante rouge de la lumière ambiante. La valeur par défaut est 160.

***GreenValue***: Un nombre entier allant de 0 à 255 représentant la composante verte de la lumière ambiante. La valeur par défaut est 160.
***BlueValue***: Un nombre entier allant de 0 à 255 représentant la composante bleu de la lumière ambiante. La valeur par défaut est 160.
Cette commande définit la couleur de la lumière ambiante qui doit être utilisé. Tous les polygones de la scène sont éclairés par la lumière ambiante indépendamment de la direction de la lumière.

---

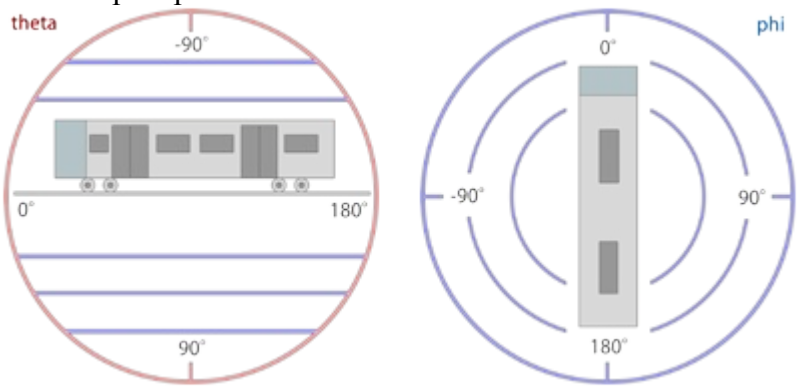
**`Route.DirectionalLight`** *`RedValue; GreenValue; BlueValue`*

***RedValue***: Un nombre entier allant de 0 à 255 représentant la composante rouge de la lumière directionnelle. La valeur par défaut est 160.
***GreenValue***: Un nombre entier allant de 0 à 255 représentant la composante verte de la lumière directionnelle. La valeur par défaut est 160.
***BlueValue***: Un nombre entier allant de 0 à 255 représentant la composante bleu de la lumière directionnelle. La valeur par défaut est 160.
Cette commande définit la lumière directionnelle qui doit être utilisé. Les polygones de la scène ne seront pleinement éclairé par la lumière directionnelle que si la direction de la lumière pointe sur les faces avant. Si pointant sur les faces arrière, aucune lumière ne sera prise en compte. *Route.LightDirection* devrait être définie pour spécifier la direction de la lumière.

---

**`Route.LightDirection`** *`Theta; Phi`*

***Theta***: Un nombre (à point flottant) représentant l'angle en **dégrées** qui contrôle la hauteur de la direction de la lumière. La valeur par défaut est 60.
***Phi***: Un nombre (à point flottant) représentant l'angle en **degrees** qui contrôle la rotation plane de la direction de la lumière. La valeur par défaut est d'environ -26,57.
Cette commande définit la direction de la lumière initiale pour la position 0 de la voie. C'est le sens ou la lumière brille, signifiant que le soleil est situé dans la direction opposée. Tout d'abord, *Theta* détermines la hauteur. Une valeur de 90 représente une direction vers le bas, tandis qu'une valeur de -90 représente une direction vers le haut. Avec ces deux extrêmes, la valeur de *Phi* n'est pas pertinent. Une valeur de 0 pour *Theta* représente une direction vers l'avant venant de l'horizon de derrière. Une fois la hauteur défini par *Theta*, *Phi* determines la rotation en plan. Une valeur de 0 pas de rotation, Une valeur de 90 rotation en direction de la droite et Une valeur de -90 rotation vers la gauche. Une orientation vers l'arrière peut être à la fois obtenues en définissant *Theta* et *Phi* à 180 et 0 ou 0 et 180, respectivement. Les valeurs entre les deux permettent un contrôle plus précis de la direction de la lumière exacte.




*Convertir une direction sphérique (theta,phi) dans une direction cartésienne (x,y,z) :*

*f*
```
x = cos(theta) * sin(phi)
y = -sin(theta)
z = cos(theta) * cos(phi)
```

*Conversion d'une direction cartésienne (x,y,z) dans une direction sphériques (theta,phi) pour $y^2 \neq 1$ :*

*f*
```
theta = -arctan(y/sqrt(x²+z²))
phi = arctan(z,x)
```
*Conversion d'une direction cartésienne (x,y,z) dans une direction sphériques (theta,phi) pour y²=1 :*

*f*
```
theta = -y * pi/2
phi = 0
```
Dans les formules ci-dessus, *cos(x)* représente le cosinus, *sin(x)* la condition sine, *arctan(x)* la tangente inverse, *arctan(x,y)* la tangente à deux arguments inverses et *sqrt(x)* la racine carrée. Les formules peuvent être utilisées pour convertir entre les coordonnées sphériques et cartésiennes si vous travaillez avec l'un qui semble plus intuitif que de travailler avec l'autre. Les formules servent aussi à la documentation officielle sur la façon dont la direction de la lumière est définie mathématiquement (en utilisant les radians pour les fonctions trigonométriques).

---

```
Route.DeveloperID
```
Cette commande est ignorée par openBVE.

## ■ 6. Le Train namespace

Les commandes à partir de ce "namespace" définit les associations route-train.

---

```
Train.Folder FolderName
Train.File FolderName
```
***FolderName***: Le nom de dossier du train à utiliser par défaut sur cet itinéraire.

---

```
Train.Run(RailTypeIndex).Set RunSoundIndex
Train.Rail(RailTypeIndex).Set RunSoundIndex
```
***RailTypeIndex***: Un entier non négatif représentant le type de rail tel que défini par Structure.Rail et utilisé par Track.RailType.
***RunSoundIndex***: Un nombre entier non négatif représentant le son du roulement du train associer au type de rail.
Le développeur du train offre un répertoire de sons de roulement différents. Utilisez cette commande pour associer ces sons de roulement pour les types de rails que vous utilisez dans votre itinéraire. Pour les sons de roulement corrects devant être joué sur n'importe lequel de vos types de rails, vous devez coordonner vos efforts avec le développeur du train. S'il vous plaît voir la page sur standards pour plus d'informations.

---

```
Train.Flange(RailTypeIndex).Set FlangeSoundIndex
```
***RailTypeIndex***: Un nombre entier non négatif représentant le type de rail tel que défini par Structure.Rail et utilisé par Track.RailType.
***RunSoundIndex***: Un entier non négatif représentant le son bridé du train associer au type de rail.
Le développeur du train offre un répertoire de sons bridés différents. Utilisez cette commande pour associer ces sons bridés aux types de rails que vous utilisez dans votre itinéraire. Pour les sons bridés corrects devant être joué sur n'importe lequel de vos types de rails, vous devez coordonner vos efforts avec le développeur du train. S'il vous plaît voir la page sur standards pour plus d'informations.

---

**Train.Timetable**(*TimetableIndex*)**.Day**.Load FileName

***TimetableIndex*** : Un nombre entier non négatif représentant l'index timetable (tableau des horaires).

***FileName*** : Le nom du fichier pour la version de jour de timetable, relatif au dossier train (1er choix), ou relatif au dossier **Object** (2eme choix).

Utilisez cette commande pour charger la version de jour de timetable. L'index de l'horaire montre la station de départ particulière qui peut être défini par les commandes Track.Sta.

---

**Train.Timetable**(*TimetableIndex*)**.Night**.Load FileName

***TimetableIndex*** : Un nombre entier non négatif représentant l'index timetable.

***FileName*** : Le nom de fichier pour la version de nuit de timetable, relatif au dossier train (1er choix), ou relatif au dossier **Object** (2eme choix).

Utilisez cette commande pour charger la version de nuit de timetable. L'index de l'horaire montre la station de départ particulière qui peut être défini par les commandes Track.Sta.

---

**Train.Gauge** *ValueInMillimeters*

***ValueInMillimeters*** : Un nombre (à point flottant) représentant l'écartement des rails gauge, measuré en **millimètres** (0.001 mètres). La valeur par défaut est 1435.

☀ *Note:*
Train.Gauge est identique à Route.Gauge.

---

**Train.Interval** $Interval_0$; $Interval_1$; ...; $Interval_{n-1}$

***ValueInSeconds*** : Un nombre (à point flottant) qui représente l'intervalle de temps entre le train du joueur et le train qui précède, mesuré en **secondes**. La valeur par défaut est 0.

Idem à Route.RunInterval. Pour plus d'informations voir à Route.RunInterval.

---

**Train.Velocity** *Speed*

***Speed***: Un nombre positif (à point flottant) qui représente la vitesse maximale des trains précédents peuvent avoir, **par défaut** measuré en **kilometres par heure**, ou 0 pour une vitesse infinie. La valeur par défaut est 0.

Cette commande définit la vitesse maximale des trains précédents peuvent avoir. La vitesse réelle peut être réduit par les commandes de Track.Limit. Le train du joueur n'est pas affecté par ce paramètre et peut se déplacer dans les limites définies par Track.Limit.

**Train.Acceleration**

Cette commande est ignorée par openBVE.

**Train.Station**

Cette commande est ignorée par openBVE.

## ■ 7. La Structure namespace

Les commandes dans Structure namespace définissent les objets à utiliser dans d'autres commandes. Généralement, les commandes comme Track.Rail, Track.FreeObj et ainsi de créer des objets référencés par *StructureIndex*. Ce *StructureIndex* est spécifique à cette commande, vous pouvez donc définir un ensemble d'objets spécifiques à Track.Rail, Track.FreeObj et ainsi de suite.

La syntaxe des commandes dans Structure namespace est la suivante :

```
Structure.Command(StructureIndex).Load FileName
```

*StructureIndex* est un entier non négatif. *FileName* est le fichier objet à charger, relatif au dossier **Object**. *Command* est l'une des commandes suivantes :

| ***Commande*:** | **Remarques** |
|---|---|
| Ground | Définit les objets pour Cycle.Ground et Track.Ground. |
| Rail | Définit les objets pour Track.Rail, Track.RailStart et Track.RailType. |
| WallL | Définit les objets à gauche pour Track.Wall. |
| WallR | Définit les objets à droite pour Track.Wall. |
| DikeL | Définit les objets à gauche pour Track.Dike. |
| DikeR | Définit les objets à droite pour Track.Dike. |
| FormL | Définit les bords gauche pour les plates-formes Track.Form. |
| FormR | Définit les bords droit pour les plates-formes Track.Form. |
| FormCL | Définit les plateformes gauches transformable pour Track.Form. Pas d'objets ANIMATED pris en charge. |
| FormCR | Définit les plateformes droites transformable pour Track.Form. Pas d'objets ANIMATED pris en charge. |
| RoofL | Définit les bords de toit gauche pour Track.Form. |
| RoofR | Définit les bords de toit droit pour Track.Form. |
| RoofCL | Définit les bords de toit gauche transformable pour Track.Form. Pas d'objets ANIMATED pris en charge. |
| RoofCR | Définit les bords de toit droit transformable pour Track.Form. Pas d'objets ANIMATED pris en charge. |
| CrackL | Définit les objets transformables gauche pour Track.Crack. Pas d'objets ANIMATED pris en charge. |

| | |
|---|---|
| CrackR | Définit les objets transformables droit pour Track.Crack. Pas d'objets ANIMATED pris en charge. |
| FreeObj | Définit les objets pour Track.FreeObj. |
| Beacon | Définit les objets pour Track.Beacon. (balise) |

Généralement, les objets supportés sont B3D, CSV, X and ANIMATED. Cependant, les commandes FormCL, FormCR, RoofCL, RoofCR, CrackL et CrackR n'acceptent seulement les objets B3D, CSV et X.
➟ Pour plus d'informations sur forms, roofs et cracks...

De plus, il ya la commande Structure.Pole, qui a une syntaxe légèrement différente :

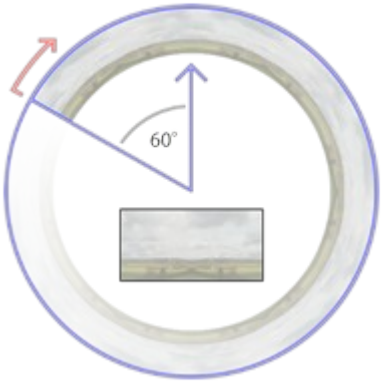
```
Structure.Pole(NumberOfAdditionalRails; PoleStructureIndex).Load
FileName
```

***NumberOfAdditionalRails***: Un entier non négatif représentant le nombre de rails supplémentaires couverts par le poteau. 0 crée un poteau pour un rail, 1 pour deux rails, etc
***PoleStructureIndex***: Un entier non négatif représentant l'index de la structure du poteau.
***FileName***: Le fichier de l'objet à charger, relatif au dossier **Object**.

S'il vous plaît noter que tous les objets sauf FreeObj sont insérés au début d'un bloc et devrait donc s'étendre de 0 à *BlockLength* (par défaut 25m) sur l'axe z. Pour plus d'informations sur l'utilisation, voir les commandes respectives de Track namespace.

## ■ 8. La Texture namespace

Les commandes à partir de ce namespace défini quelles sont les images de fond à utiliser et comment elles sont alignés.




L'image de fond est affiché comme un mur cylindrique autour de la caméra dont le départ (vu de dessus) est de 60 degrés vers la gauche de la direction vers l'avant initiales (à la position 10 heures). De là, l'image de fond enveloppe autour du cylindredans le sens des aiguilles d'une montre avec un nombre de répétition spécifié via Texture.Background(*BackgroundTextureIndex*).X, qui, par défaut crée 6 répétitions dans un cercle complet.

Les 3 / 4 supérieurs de l'image est affichée au-dessus de l'horizon, tandis que le 1 / 4 inférieur est affiché en dessous de l'horizon. Via Texture.Background(*BackgroundTextureIndex*).Aspect, vous pouvez choisir une hauteur fixe du cylindre ou garder le ratio d'aspect de la texture. Si l'image doit avoir une hauteur fixe, le cylindre a une hauteur de 1/2 de son rayon, ce qui correspond à environ 20 degrés d'inclinaison vers le haut de

l'image, et d'environ -7 degrés vers le bas de l'image. Si le ratio d'aspect de l'image est préservée, cela prend non seulement la largeur et la hauteur de l'image en compte, mais aussi le nombre de répétitions.

Peu importe le nombre de répétitions que vous avez choisie, vous devez vous assurer que les bords gauche et droit des textures s'intègrent parfaitement ensemble. S'il vous plaît prendre également en compte que les plafonds haut et bas sont créés avec la partie du haut et du bas de 10% de l'image. Vous devriez éviter les pics de montagne et les phénomènes extrêmes semblables dans le haut des 10% de l'image afin que de tels extrêmes ne fuit pas dans le plafond.

L'image chargée dans Texture.Background(0) est affiché au début de la route, sauf si une commande Track.Back au début de la route demande une image différente.

---

**Texture.Background(*BackgroundTextureIndex*).Load *FileName***

***FileName*** : Le fichier de texture à charger, relatif au dossier **Object**.
Cette commande charge une image de fond pour être utilisé plus tard par Track.Back.

---

**Texture.Background(*BackgroundTextureIndex*).X *RepetitionCount***

***RepetitionCount*** : Le nombre de fois que l'image de fond est répétée dans un cercle complet. La valeur par défaut est 6.

---

**Texture.Background(*BackgroundTextureIndex*).Aspect *Mode***

***Mode***: Le mode de manipulation du ratio de l'aspect à utiliser. La valeur par défaut est 0.

▸ Options pour *Mode* :

**0** : Utilise une hauteur fixe pour le cylindre.
**1** : Préserve le ratio de l'aspect de l'image.

## ■ 9. The Cycle namespace

---

**Cycle.Ground(*GroundStructureIndex*).Params $GroundStructureIndex_0$; $GroundStructureIndex_1$; $GroundStructureIndex_2$; ...; $GroundStructureIndex_{n-1}$**

***GroundStructureIndex***: Un entier non négatif indiquant l'indice de la structure du sol pour lesquels un cycle est à définir.
$GroundStructureIndex_i$: Un entier non négatif indiquant une structure du sol qui a été préalablement chargée via Structure.Ground.
Lorsque on utilise habituellement Track.Ground(*GroundStructureIndex*), le même objet de structure du sol sera placé plusieurs fois dans chaque bloc. Avec Cycle.Ground, vous pouvez modifier ce comportement et le changer automatiquement par une série d'objets différents en utilisant Track.Ground(*GroundStructureIndex*). Le cycle se répète indéfiniment, où $GroundStructureIndex_0$ commence une position de voie à 0. Techniquement, le *i* dans $GroundStructureIndex_i$ choisi pour une position de voie particulière *p* correspond à `Mod[p /`

`BlockLength, n]`.

Considérons la définition suivante :

```
With Structure
.Ground(0) grass.csv
.Ground(1) river.csv
.Ground(2) asphalt.csv
```

Les deux codes suivants vont produire le même résultat :

```
With Track
0, .Ground 0
25, .Ground 1
50, .Ground 2
75, .Ground 0
100, .Ground 1
125, .Ground 2
; et ainsi de suite...
```

```
With Cycle
.Ground(0) 0; 1; 2
With Track
0, .Ground 0
```

## ■ 10. Le Signal namespace

Les commandes à partir de ce namespace définir les signaux personnalisés.

---

```
Signal(SignalIndex).Load AnimatedObjectFile
```

***SignalIndex***: Un entier non négatif représentant l'index du signal.
***AnimatedObjectFile***: Une référence à un fichier objet animé, relatif au dossier **Object**.
Utilisez cette commande pour charger directement les signaux à partir d'objets animés. Le *SignalIndex* peut être référencés plus tard par les commandes de Track.SigF pour placer le signal.

---

```
Signal(SignalIndex).Load SignalFileWithoutExtension;
GlowFileWithoutExtension
```

***SignalIndex***: Un entier non négatif représentant l'index du signal.
***SignalFileWithoutExtension***: Une référence à un fichier objet B3D/CSV/X représentant le signal, relatif au dossier **Object**, mais sans l'extension du fichier. **Doit être spécifié.**
***GlowFileWithoutExtension***: Une référence facultative à un fichier objet B3D/CSV/X représentant la lueur de distance, relatif au dossier **Object**, mais sans l'extension du fichier.
Utilisez cette commande pour charger les signaux d'une série de textures individuelles appliquées sur un objet commun. openBVE cherche les objets X, CSVet B3D dans cet ordre exact. Les textures sont tenus d'avoir le même nom que le signal ou la lueur, plus un index d'aspect non-négatif, plus une extension de fichier pour les textures. Le *SignalIndex* peut être référencés plus tard dans les commandes Track.SigF pour placer le signal.

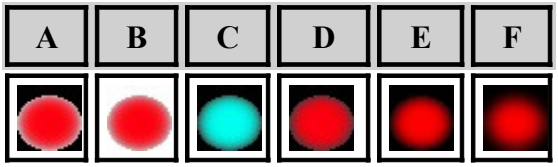
Pour *SignalFileWithoutExtension*, il devrait y avoir les fichiers suivants présents (exemple) :

*SignalFileWithoutExtension*.x
*SignalFileWithoutExtension*0.bmp
*SignalFileWithoutExtension*1.bmp
*SignalFileWithoutExtension*2.bmp
*SignalFileWithoutExtension**n**.bmp

Les index d'aspect de 0 à *n* représentent successivement des aspects plus permissif, où 0 est rouge. Les signaux intégré, par exemple, utiliser les indices 0 (●), 1 (●●), 2 (●), 3 (●●), 4 (●) et 5 (●●). Vous pouvez les utiliser autant que nécessaire.

Toutes les faces dans l'objet seront appliquée l'élément de texture actif. Cela signifie que vous ne pouvez pas utiliser toute autre texture dans l'objet, mais toujours attribuer des coordonnées de texture appropriée. Pour l'objet lueur, les règles ci-dessus s'appliquent également. L'objet lueur est généralement un rectangle placé nettement en avant du signal, mais vous pouvez également utiliser des formes différentes

Les textures lueur méritent une attention particulière. Toutes les textures lueur sont pré-traitées de la façon suivante :




La texture avec laquelle vous commencez doit avoir une forme forte, généralement ovale. La forme doit être complètement saturé dans le centre et se fondre vers un blanc pur à sa périphérie. Les alentours de la forme peut être soit en noir pur (A) ou un blanc pur (B).

Lorsque openBVE charges la texture lueur, il remplacera tous les pixels noirs purs avec des pixels blancs purs, arrivant ainsi à (B). De là, l'image est inversée (C), puis la teinte décalée de 180 degrés (D). Par rapport à (B), cela a pour effet global l'inversion de la brillance de l'image, les pixels complètement saturée seront laissé inchangé (par exemple le noyau), alors que les pixels brillants (comme le rebord extérieur de la forme) devient sombre, et vice versa. Ensuite,  le gamma de l'image est corrigé pour encore assombrir les parties sombres (E), et finalement, l'image es légèrement flouté (F).

La texture résultante est toujours un mélange additif. Cela signifie qu'au lieu de dessiner directement la texture sur l'écran, les pixels de la texture sont ajoutés aux pixels de l'écran. Ici, l'ajout de noir (0) ne change pas les pixels de l'écran, l'ajout d'un canal de couleur totalement saturée (1) se traduira par un canal de couleur totalement saturée, par exemple ajouté du blanc produit du blanc. Gardez à l'esprit que lors de la conception des textures, que vous devrez suivre les règles inverses, par exemple, la conception de l'image comme décrit dans (A) ou (B), tout en ayant à l'esprit comment elle sera traité plus tard.

## ■ 11. Le Track namespace

Les commandes à partir de ce namespace définit le tracé de la voie. Les commandes à partir de ce namespace doit apparaître après les commandes de l'un des autres namespaces, et ils forment habituellement la plus grande partie du fichier route.

ⓘ Utilisation des positions des voies

Toutes les commandes à partir de Track namespace doivent être associés à la position des voies.
Une fois la position de la voie définie, toutes les commandes suivantes sont associées à cette

position de la voie jusqu'à qu'une nouvelle position de voie soit définie. Si vous n'avez pas explicitement définie une position de voie avant que la première commande de Track namespace soit utilisé, 0 sera supposée. Même si vous n'avez pas besoin d'utiliser des positions de voie en ordre ascendant, la série de commandes qui est associées à la même position de voie seront triés en ordre croissant une fois le fichier chargé. Bien que les positions de voie peuvent être chacune un nombre non-négatifs (à point flottant), de nombreuses commandes dans Track namespace ne peuvent être appliqués au début d'un bloc, qui est de 25m par défaut. Pour la situation par défaut, cela signifie que certaines commandes ne doivent être utilisés à une position de voie de 0, 25, 50, 75, 100, 125 et ainsi de suite. Toutes les commandes pour lesquelles cette restriction s'applique sont marqués comme tels.



---

### `Track.RailStart RailIndex; X; Y; RailType`

***RailIndex*** : Un entier positif (**>=1**) indiquant quel index de rail utiliser.
***X*** : Un nombre (à point flottant) qui représente la distance horizontale par rapport au rail du joueur, **par défaut** mesuré en mètres. Les valeurs négatives indiquent la gauche, les positives la droite.
***Y*** : Un nombre (à point flottant) qui représente la distance verticale distance par rapport au rail du joueur, **par défaut** mesuré en **mètres.** Les valeurs négatives indiquent en dessous, les positifs par dessus.
***RailType*** : Un entier non négatif référençant le type de rail utilisé tel que définies par une commande de Structure.Rail.

Cette commande crée un nouveau rail représenté par l'index *RailIndex*. Dés que cette commande est utilisée, un rail du même *RailIndex* doit ne pas avoir été utilisé jusqu'ici dans l'itinéraire, ou doit avoir été terminé par une commande Track.RailEnd. Si un rail de même *RailIndex* était déjà utilisé dans l'itinéraire, les valeurs par défaut de *X*, *Y* et *RailType* sont les dernières valeurs utilisées par le rail, sinon 0. Si le rail est à mettre à jour, utilisez la commande Track.Rail. S'il doit être terminée, utilisez la commande Track.RailEnd. Vous pouvez mettre fin à un rail de *RailIndex* donné et commencer un nouveau rail de même *RailIndex* à la même position de voie à condition que l'ancien rail est d'abord pris fin et que le nouveau rail commence après. Pour chaque bloc, une structure, déterminée par *RailType*, est automatiquement placée.

⚠ Cette commande doit être utilisé au début d'un bloc.

---

### `Track.Rail RailIndex; X; Y; RailType`

***RailIndex*** : Un entier positif (**>=1**) indiquant quel index de rail utiliser.
***X*** : Un nombre (à point flottant) qui représente la distance horizontale par rapport au rail du joueur, **par défaut** mesuré en mètres. Les valeurs négatives indiquent la gauche, les positives la droite.
***Y*** : Un nombre (à point flottant) qui représente la distance verticale distance par rapport au rail du joueur, **par défaut** mesuré en **mètres.** Les valeurs négatives indiquent en dessous, les positifs par dessus.
***RailType*** : Un entier non négatif référençant le type de rail utilisé tel que définies par une commande de Structure.Rail.

Cette commande crée un nouveau rail ou met à jour un rail existant. Le rail est représenté par l'index *RailIndex*. Si un rail de même *RailIndex* était déjà utilisé dans l'itinéraire, les valeurs par défaut de *X*, *Y* et *RailType* sont les dernières valeurs utilisées par le rail, sinon 0. Si le rail doit être terminée, utilisez la commande Track.RailEnd. Vous pouvez mettre fin à un rail de *RailIndex* donné et commencer un nouveau rail de même *RailIndex* à la

même position de voie à condition que l'ancien rail est d'abord pris fin et que le nouveau rail commence après. Dans chaque bloc, les valeurs *X* et *Y* sont répétées si une commande Track.Rail n'est pas utilisé pour ce bloc. En conséquence, l'actualisation des valeurs *X* ou *Y* affecte la disposition du rail du bloc précédent seulement. La modification de *RailType* aura une incidence sur le rail du point de l'endroit où cette commande est utilisée. Si cette commande est utilisée plusieurs fois sur la même position de voie pour le même rail, la première instance de la commande prend effet, tandis que les suivantes sont ignorées.

⚠ Cette commande ne peut pas être utilisé au début d'un bloc.

ⓘ Track.RailStart vs. Track.Rail

Si vous souhaitez démarrer un nouveau rail, vous pouvez soit utiliser Track.RailStart ou Track.Rail. Lorsque vous utilisez Track.RailStart, vous donnez le balisage qu'un nouveau rail est en train de démarré, ce qui n'est pas valide si le rail existe déjà. En utilisant un explicite Track.RailStart vous protégera de l'utilisation d'un *RailIndex* qui est déjà utilisé, auquel cas un message d'erreur est généré.

---

## `Track.RailType RailIndex; RailType`

***RailIndex***: Un entier non négatif indiquant l'index du rail à changer. Le rail du joueur peut être désigné par l'index **0**. La valeur par défaut est 0.
***RailType***: Un entier non négatif référençant le type de rail utilisé tel que définies par une commande de Structure.Rail. La valeur par défaut est 0.

Cette commande modifie le type de rail pour un rail existant, representé par *RailIndex*. Le rail doit avoir été commencé avec une commande Track.RailStart ou Track.Rail et ne doit pas avoir été terminé par une commande Track.RailEnd. La modification de *RailType* aura une incidence sur le rail à l'endroit où cette commande est utilisée.

⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

## `Track.RailEnd RailIndex; X; Y`

***RailIndex*** : Un entier positif (**>=1**) indiquant quel index de rail utiliser.
***X*** : Un nombre (à point flottant) qui représente la distance horizontale par rapport au rail du joueur, **par défaut** mesuré en mètres. Les valeurs négatives indiquent la gauche, les positives la droite.
***Y*** : Un nombre (à point flottant) qui représente la distance verticale distance par rapport au rail du joueur, **par défaut** mesuré en **mètres.** Les valeurs négatives indiquent en dessous, les positifs par dessus.

Cette commande termine un rail existant, représentée par l'index *RailIndex*. Les valeurs par défaut de *X* et *Y* sont les dernières utilisées par le rail. Une fois cette commande utilisée pour un *RailIndex* spécifique, le rail correspondant est considéré par la suite comme inexistant.

⚠ Cette commande ne peut être utilisé au début d'un bloc.

*Exemple de commandes Track.RailStart, Track.Rail, Track.RailType and Track.RailEnd*

```
With Track
1000, .RailStart 1; 3.8; 0.0; 0
1025, .RailType 1; 1
1050, .Rail 1; 1.9; 0.0; 0
1075, .RailEnd 1
```

Dans l'exemple précédent, le rail 1 commence avec une valeur x de 3.8 et porte un index de rail qui corresponds à un objet destiné à illustrer un rail rectiligne. Le rail conserve la valeur x de 3.8 jusqu'à la position de voie 1025, où le type de rail est changé pour correspondre à un objet destiné à représenter une courbe en forme de S. A la position de voie 1050, le rail est redéfinie pour avoir une valeur x de 1.9, après quoi le rail est encore droit jusqu'à 1075, où il se termine, ayant encore une valeur x de 1.9.

---

`Track.Accuracy Value`

***Value***: Un nombre non-négatifs (à point flottant) qui représente la précision de la voie. La valeur par défaut est de 2.

Cette commande définit la précision de la piste à partir de ce point. Les valeurs doivent être dans la plage de 0 à 4, où 0 signifie une précision parfaite (aucune imprécision du tout), 1 signifie une très bonne précision (lignes à grande vitesse), 2 signifie une bonne précision, 3 signifie une précision médiocre, et 4 signifie une mauvaise précision. Les valeurs intermédiaires sont également possibles. Actuellement, les valeurs inférieures à 0 sont fixés à 0, et les valeurs supérieures à 4 sont fixés à 4.

---

`Track.Adhesion Rate`

***Rate***: Un nombre non-négatifs (à point flottant) mesurée en pourcentage représentant de l'adhérence de la voie. La valeur par défaut est 100.

Cette commande définit l'adhérence de la voie à partir de ce point. À titre de référence, la valeur de 135 représente des conditions sèches, 85 représente des conditions de gel et 50 représente des conditions de neige. Avec une valeur de 0, le train ne sera pas capable de se déplacer du tout.

• 11.2. Géométrie

---

`Track.Pitch Rate`

***Rate*** : Un nombre (à point flottant) mesurée par milliers représentant la hauteur de la voie. La valeur par défaut est 0.

Cette commande définit la pente de tous les rails à partir de ce point. Les valeurs négatives indiquent une pente descendante, une positive une pente ascendante. La valeur 0 représente le niveau de la voie. *Rate* peut être calculée à partir d'une différence de longueur *X* et d'une différence de hauteur *Y* de la façon suivante :

*Rate s'exprime à travers X et Y:*

*f* `Rate = 1000 * Y / X`

⚠ Cette commande ne peut pas être utilisé au début d'un bloc.

---

`Track.Curve Radius; CantInMillimeters`

***Radius*** : Un nombre (à point flottant) représentant le rayon de la courbe, **par défaut** mesuré en **metres**. La valeur par défaut est 0.

***CantInMillimeters***: Un nombre (à point flottant) qui représente le dévers d'une courbe incliné, **toujours** mesuré en **millimètres** (0.001 metres). La valeur par défaut est 0. Voir aussi Options.CantBehavior.

Cette commande définit le rayon de la courbe pour le rail du joueur à partir de ce point. Les valeurs négatives de *Radius* indique une courbe à gauche, les positives vers la droite. La valeur 0 représente une voie droite. Le

paramètre *CantInMillimeters* définit le dévers (surélévation) en millimètres. Si Options.CantBehavior est à 0 (par defaut), seule la valeur absolue de *CantInMillimeters* est considéré, et le dévers est toujours vers le centre de courbe (vers l'intérieur). Aussi, cant ne peut pas être appliqué sur une voie rectiligne. Si Options.CantBehavior est à 1, *CantInMillimeters* est signé, c'est à dire les valeurs négatives vers le bord extérieur et positives vers le bord intérieur de la voie courbes. Aussi, cant peut être appliqué sur une voie rectiligne, où les valeurs négatives vers le bord gauche et celles positives vers la droite.

⚠ Cette commande ne peut pas être utilisé au début d'un bloc.

---

```
Track.Turn Ratio
```

***Ratio*** : Un nombre (à point flottant) qui représente un tournant. La valeur par défaut est 0.

Cette commande crée un tournant basées sur un point au point d'insertion. *Ratio* indique le rapport entre la différence de longueur *Z* et décalage horizontal *X* de la façon suivante :

*f* `Ratio = X / Z`

Un ratio négatif représente un virage à gauche, positif un virage à droite. La valeur 0 représente une voie droite.

⚠ Cette commande ne peut pas être utilisé au début d'un bloc.

⚠ Cette commande est obsolète – utilisez plutôt Track.Curve.

---

```
Track.Height Y
```

***Y*** : Un nombre (à point flottant) qui représente la hauteur du rail du joueur, **par défaut** mesuré en mètres.

Cette commande définit la hauteur du rail du joueur au-dessus du sol au point d'insertion. Elle influence le placement des objets au sol définie par Structure.Ground et modifiés par Track.Ground. La hauteur est interpolé entre les commandes adjacentes de Track.Height. Par exemple, les deux codes suivants produisent des résultats équivalents :

*Exemple d'une commande Track.Height interpolée aux limites des 25m :*

```
1000, Track.Height 1
1075, Track.Height 4
```

*Exemple de Track.Height définit explicitement tous les 25m produisant le même résultat :*

```
1000, Track.Height 1
1025, Track.Height 2
1050, Track.Height 3
1075, Track.Height 4
```

⚠ Cette commande ne peut être utilisé au début d'un bloc.

• 11.3. Objects

---

```
Track.FreeObj RailIndex; FreeObjStructureIndex; X; Y; Yaw; Pitch; Roll
```

***RailIndex*** : Un entier non négatif représentant le rail sur lequel placer l'objet. La valeur par défaut est 0.

***FreeObjStructureIndex*** : Un entier non négatif qui représente l'objet à placer tel que défini par

Structure.FreeObj. La valeur par défaut est 0.
***X*** : Le décalage X par rapport au rail (rectiligne), **par défaut** mesurée en **mètres**. Les valeurs négatives représentent la gauche, les positives la droite. La valeur par défaut est 0.
***Y*** : Le décalage Y par rapport au rail (rectiligne), **par défaut** measurée en **mètres**. Les valeurs négatives représentent en dessous du sommet des rails, ceux positifs au-dessus. La valeur par défaut est 0.
***Yaw*** : L'angle en degrés par lequel l'objet est tourné dans le plan XZ dans le sens des aiguilles d'une montre vu de dessus. La valeur par défaut est 0.
***Pitch*** : L'angle en degrés par lequel l'objet est tourné dans le plan YZ dans le sens des aiguilles d'une montre vu de la gauche. La valeur par défaut est 0.
***Roll*** : L'angle en degrés par lequel l'objet est tourné dans le plan XY dans le sens des aiguilles d'une montre vu de derrière. La valeur par défaut est 0.
Cela place un objet "libre" une seule fois sur un rail spécifique. L'objet doit avoir été chargé par Structure.FreeObj(*FreeObjStructureIndex*) avant d'utiliser cette commande.

---

### `Track.Wall` *`RailIndex; Direction; WallStructureIndex`*

***RailIndex*** : Un entier non négatif représentant le rail sur lequel commencer ou mettre à jour le mur. La valeur par défaut est 0.
***Direction*** : Un entier indiquant quel mur utiliser comme décrit ci-dessous.
***WallStructureIndex*** : Un entier non négatif qui représente l'objet à placer tel que défini par Structure.WallL et Structure.WallR. La valeur par défaut est 0.

▸ Options pour *Direction*:

**-1** : L'objet WallL (mur gauche) est utilisé.
**0** : les deux objets WallL et WallR sont utilisés.
**1** : L'objet WallR (mur droit) est utilisé.

Cela démarre ou met à jour un mur sur un rail spécifié. L'objet doit avoir été chargé par Structure.WallL(*WallObjectIndex*) ou Structure.WallR(*WallObjectIndex*) avant d'utiliser cette commande. Les murs sont placés au début de chaque bloc jusqu'à une fin de mur correspondant à Track.WallEnd pour ce rail. S'il vous plaît noter que les murs sont réutilisés, si un rail est terminée par Track.RailEnd et qu'il redémarre par Track.RailStart ou Track.Rail à moins que le mur n'est été fini par l'intermédiaire de Track.WallEnd.

⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

### `Track.WallEnd` *`RailIndex`*

***RailIndex*** : Un entier non négatif représentant le rail sur lequel on met fin au mur existant.

Ceci termine un mur existant qui a précédemment été démarré par Track.Wall sur un rail spécifique. Le mur n'est pas placé depuis le bloc dans lequel cette commande est utilisée.

⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

### `Track.Dike` *`RailIndex; Direction; DikeStructureIndex`*

***RailIndex***: Un entier non négatif représentant le rail sur lequel commencer ou mettre à jour la digue La valeur par défaut est 0.
***Direction***: Un entier indiquant la digue à utiliser comme décrit ci-dessous.
***DikeStructureIndex***: Un entier non négatif qui représente l'objet à placer tel que défini par Structure.DikeL et Structure.DikeR. La valeur par défaut est 0.

▸ Options pour *Direction*:
**-1**: L'objet DikeL (digue gauche) est utilisé.
**0**: Les deux objets DikeL et DikeR sont utilisés .
**1**: L'objet DikeR (digue droite) est utilisé.
Cela démarre ou met à jour une digue sur un rail spécifié. L'objet doit avoir été chargé par Structure.DikeL(*DikeObjectIndex*) ou Structure.DikeR(*DikeObjectIndex*) avant d'utiliser cette commande. Les digues sont placées au début de chaque bloc jusqu'à une fin de digue correspondant à Track.DikeEnd pour ce rail. S'il vous plaît noter que les digues sont réutilisés, si un rail est terminée par Track.RailEnd et qu'il redémarre par Track.RailStart ou Track.Rail à moins que la digue n'est été fini par l'intermédiaire de Track.DikeEnd.
⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

```
Track.DikeEnd RailIndex
```
***RailIndex*** : Un entier non négatif représentant le rail sur lequel mettre fin à une digue existante.
Ceci termine une digue existante qui a précédemment été démarré par Track.Dikesur un rail spécifique . La digue n'est pas forcément placé dans le bloc pour lequel cette commande est utilisée.
⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

```
Track.Pole RailIndex; NumberOfAdditionalRails; Location;
Interval; PoleStructureIndex
```
***RailIndex*** : Un entier non négatif représentant le rail sur lequel commencer ou mettre à jour le poteau. La valeur par défaut est 0.
***NumberOfAdditionalRails*** : Un entier non négatif représentant la quantité de rails supplémentaires couverts pour ce poteau (c'est à dire ce rail, plus *NumberOfAdditionalRails* de rails). La valeur par défaut est 0.
***Location*** : Si *NumberOfAdditionalRails* est 0, le côté sur lequel le poteau est placé (voir ci-dessous), ou le décalage x dans un multiple de 3.8 mètres si *NumberOfAdditionalRails* est au moins égal à 1. La valeur par défaut est 0.
***Interval*** : Un entier multiple de la longueur du bloc spécifiant l'intervalle dans lequel sont placés les poteaux.
***PoleStructureIndex*** : Un entier non négatif qui représente l'objet à placer tel que défini par Structure.Pole. La valeur par défaut est 0.
Cela démarre ou met à jour un poteau sur un rail spécifié. L'objet doit avoir été chargé par Structure.Pole (*NumberOfAdditionalRails*; *PoleStructureIndex*) avant d'utiliser cette commande. Les poteaux sont placés au début de chaque bloc dont les positions sont un multiple entier de *Interval* (ui n'est pas nécessairement au même endroit cette commande est placé). Si *NumberOfAdditionalRails* est 0, *Location* indique le côté du rail sur lequel le poteau est placé. Si *Location* est inférieur ou égal à 0, le poteau est placé tel quel (ce qui correspond au côté gauche). Si *Location* est supérieure à 0, l'objet est mis en miroir sur l'axe des x et ensuite placé (ce qui correspond au côté droit). Si *NumberOfAdditionalRails* est supérieur ou égal à 1, *Location* spécifie le décalage x dans un multiple de 3.8 metres. S'il vous plaît noter que les poteaux sont reutilisé si un rail est terminée par Track.RailEnd et qu'il redémarre par Track.RailStart ou Track.Rail à moins que le poteau n'est été terminée par l'intermédiaire de Track.PoleEnd.
⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

```
Track.PoleEnd RailIndex
```

***RailIndex*** : Un entier non négatif représentant le rail sur lequel mettre fin à un poteau existant.
Ceci termine un poteau existant qui a été préalablement lancée via Track.Pole sur un rail spécifié. Le poteau n'est pas forcément placé dans le bloc pour lequel cette commande est utilisée.
⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

`Track.Crack RailIndex1; RailIndex2; CrackStructureIndex`

Description pas encore disponible.

---

`Track.Ground CycleIndex`

***CycleIndex*** : Un entier non négatif représentant le cycle des objets de terrain a placer tel que défini par Structure.Ground ou Cycle.Ground.
Ce qui définit des objets de terrain à partir de ce point ci. Les objets de terrain sont toujours placés au début d'un bloc à une certaine hauteur en dessous du rail du joueur (telle que définie par Track.Height). Si aucun cycle a été défini pour *CycleIndex*, l'objet chargé dans Structure.Ground(*CycleIndex*) est placé. Sinon, le cycle des objets de terrain tel que défini par Cycle.Ground(*CycleIndex*) est utilisé.
⚠ Cette commande ne peut être utilisé au début d'un bloc.

• 11.4. Stations (*Gares*)

---

`Track.Sta Name; ArrivalTime; DepartureTime; PassAlarm; Doors; ForcedRedSignal; System; ArrivalSound; StopDuration; PassengerRatio; DepartureSound; TimetableIndex`

***Name*** : Le nom de la gare. Elle est affichée dans l'horaire et dans les messages, ne devrait donc pas être omis.
***ArrivalTime*** : L'horaire du trains du joueur prévu pour arriver à cette gare. Des valeurs spéciales peuvent aussi apparaître - voir ci-dessous.
***DepartureTime*** : L'horaire du trains du joueur prévu pour le départ de cette gare. Des valeurs spéciales peuvent aussi apparaître - voir ci-dessous.
***PassAlarm*** : Indique si le dispositif d'alarme de passage devrait rappeler au conducteur de s'arrêter à cette gare. La valeur par défaut est 0.
***Doors*** : Indique quelles portes doivent s'ouvrir à cette gare. La valeur par défaut est 0.
***ForcedRedSignal*** : Indique si le signal d'arrêt derrière la dernière gare doit être rouge à l'approche du train. La valeur par défaut est 0.
***System*** : Indique quel système intégré de sécurité devraient s'appliquer jusqu'à la prochaine gare. La valeur par défaut est 0.
***ArrivalSound*** : Le fichier son à jouer à l'arrivée du train, relatif au dossier **Sound**.
***StopDuration*** : Un nombre positif (à point flottant) indiquant la durée d'arrêt minimum en secondes, y compris l'ouverture / fermeture de la porte. La valeur par défaut est 15.
***PassengerRatio*** : Un nombre positif (à point flottant) indiquant la quantité relative de passagers dans le train à cette gare. Comme référence, 100 représente un train avec une quantité normale de passagers, tandis que 250 représente un train surpeuplé. Une valeurs entre 0 et 250 doit être utilisé. La valeur par défaut est 100.

***DepartureSound*** : Le fichier sonore à jouer avant le départ (l'heure de départ moins la durée du son moins le temps de fermeture des portes), relatif au dossier **Sound**.
***TimetableIndex*** : Un entier non négatif représentant l'horaire qui sera indiqué à cette gare tel que défini par Train.Timetable(*TimetableIndex*).

▸ Les options disponibles pour *ArrivalTime* :

*time* : Le train est prévu pour arriver à cet horaire particulier.
o*mitted* : Le train peut arriver à n'importe quelle heure.
**P** ou **L** : Tous les trains doivent passer cette gare.
**B** : Le trains du joueur doit passer cette gare, tandis que tous les autres trains doivent s'arrêter.
**S** : Le trains du joueur doit s'arrêter à cette gare, tandis que tous les autres trains doivent passer.
**S :***time* : Le trains du joueur doit arriver à cet horaire particulier, alors que tous les autres trains doivent passer.

▸ Les options disponibles pour *DepartureTime* :

*time* : Le train doit partir à cet horaire particulier.
o*mitted* : Le train peut partir à n'importe quelle heure.
**T** ou = : Il s'agit de la gare terminale. Si *ForcedRedSignal* est mis à 1, le signal de départ se tiendra à rouge indéfiniment.
**T :** *time* : Il s'agit de la gare terminale. Si *ForcedRedSignal* est mis à 1, le signal de départ passera toujours au vert avant l'heure spécifiée comme s'il s'agissait d'une gare régulière.
**C** : C'est une gare ou on utilise "change ends". Voir la description ci-dessous.
**C :***time*: C'est une gare ou on utilise "change ends". Le changement de gare aura lieu à l'heure indiquée, sauf si *StopDuration* interfères. Voir la description ci-dessous.

▸ Les options disponibles pour *PassAlarm* :

**0** : Le dispositif d'alarme de passage ne rappelle pas au conducteur de s'arrêter à cette gare.
**1** : Le dispositif d'alarme passage rappelle au conducteur de s'arrêter à cette gare.

▸ Les options disponibles pour *Doors* :

**L** ou **-1** : Les portes de gauche devrait s'ouvrir à cette gare.
**N** ou **0** : Les portes ne s'ouvrent pas à cette gare, c'est à dire que le train doit simplement attendre.
**R** ou **1** : Les portes de droite devrait s'ouvrir à cette gare.
**B** : Les portes de gauche et de droite s'ouvrent à cette gare.

▸ Les options disponibles pour *ForcedRedSignal* :

**0**: Les signaux ne sont pas affectés par cette gare.
**1**: Le signal immédiatement après l'arrêt à la dernière gare sera au rouge jusqu'à ce que le train arrive à la zone d'arrêt et l'heure de départ.

▸ Les options disponibles pour *System* :

**ATS** ou **0**: ATS doit être utilisé à partir de cette gare. La voie suivante n'est pas équipée de l'ATC.
**ATC** ou **1**: ATC doit être utilisé à partir de cette gare. La voie suivante est équipé d'un ATC.

Cette commande initialise une nouvelle gare. Elle doit être placé au début du quai de la gare. Afin de finaliser la création de la gare, utilisez la commande Track.Stop qui place des points d'arrêt suite à cette commande. Toutes les commandes suivantes à Track.Stop seront associée à cette gare. Au moins une commande Track.Stop doit exister si les trains doivent s'arrêter à cette gare.

Les gares peuvent être marqués comme "changing ends" dans l'heure de départ. À ces gares, quand l'heure de départ a été atteint, le train passera automatiquement à la gare suivante. Cette fonctionnalité est destinée à continuer un voyage en sens inverse sans avoir besoin d'accéder à des gares manuellement depuis le menu.

⚠ La fonctionnalité "changing ends" est uniquement disponible dans la version 1.2.11 de développement et au-dessus.

---

```
Track.Station Name; ArrivalTime; DepartureTime; ForcedRedSignal; System; DepartureSound
```

***Name*** : Le nom de la gare. Elle est affichée dans l'horaire et dans les messages, ne devrait donc pas être omis.
***ArrivalTime*** : L'horaire du trains du joueur prévu pour arriver à cette gare. Des valeurs spéciales peuvent aussi apparaître - voir ci-dessous.
***DepartureTime*** : L'horaire du trains du joueur prévu pour le départ de cette gare. Des valeurs spéciales peuvent aussi apparaître - voir ci-dessous.
***ForcedRedSignal*** : Indique si le signal d'arrêt derrière la dernière gare doit être rouge à l'approche du train. La valeur par défaut est 0.
***System*** : Indique quel système intégré de sécurité devraient s'appliquer jusqu'à la prochaine gare. La valeur par défaut est 0.
***DepartureSound***: Le fichier sonore à jouer avant le départ (l'heure de départ moins la durée du son moins le temps de fermeture des portes), relatif au dossier **Sound**.

▸ Les options disponibles pour *ArrivalTime* :

*time* : Le train est prévu pour arriver à cet horaire particulier.
o*mitted* : Le train peut arriver à n'importe quelle heure.
**P** ou **L** : Tous les trains doivent passer cette gare.
**B** : Le trains du joueur doit passer cette gare, tandis que tous les autres trains doivent s'arrêter.
**S** : Le trains du joueur doit s'arrêter à cette gare, tandis que tous les autres trains doivent passer.
**S :***time* : Le trains du joueur doit arriver à cet horaire particulier, alors que tous les autres trains doivent passer.

▸ Les options disponibles pour *DepartureTime* :

*time* : Le train doit partir à cet horaire particulier.
o*mitted* : Le train peut partir à n'importe quelle heure.
**T** ou = : Il s'agit de la gare terminale. Si *ForcedRedSignal* est mis à 1, le signal de départ se tiendra à rouge indéfiniment.
**T :** *time* : Il s'agit de la gare terminale. Si *ForcedRedSignal* est mis à 1, le signal de départ passera toujours au vert avant l'heure spécifiée comme s'il s'agissait d'une gare régulière.
**C** : C'est une gare ou on utilise "change ends". Voir la description ci-dessous.
**C :***time*: C'est une gare ou on utilise "change ends". Le changement de gare aura lieu à l'heure indiquée, sauf si *StopDuration* interfères. Voir la description ci-dessous.

▸ Les options disponibles pour *ForcedRedSignal* :

**0**: Les signaux ne sont pas affectés par cette gare.
**1**: Le signal immédiatement après l'arrêt à la dernière gare sera au rouge jusqu'à ce que le train arrive à la zone d'arrêt et l'heure de départ.

▸ Les options disponibles pour *System* :

**ATS** ou **0**: ATS doit être utilisé à partir de cette gare. La voie suivante n'est pas équipée de l'ATC.
**ATC** ou **1**: ATC doit être utilisé à partir de cette gare. La voie suivante est équipé d'un ATC.

Cette commande initialise une nouvelle gare. Préférez l'utilisation de la commande Track.Sta, qui comprend plus d'options. Pour les options Track.Sta qui ne sont pas offerts par Track.Station, les valeurs suivantes s'appliquent :

| | |
|---|---|
| *PassAlarm* | 0 (pas utilisé) |
| *Doors* | B (Toutes les portes sont ouvertes) |
| *ArrivalSound* | Pas joué |
| *StopDuration* | 15 |

| | |
|---|---|
| *PassengerRatio* | 100 |
| *TimetableIndex* | Non affecté |

La commande doit être placée au début du quai de la gare. Afin de finaliser la création de la gare, utilisez la commande Track.Stop qui place des points d'arrêt suite à cette commande. Toutes les commandes suivantes Track.Stop seront associée à cette gare. Au moins une commande Track.Stop doit exister si les trains doivent s'arrêter à cette gare.

Les gares peuvent être marqués comme "changing ends" dans l'heure de départ. À ces gares, quand l'heure de départ a été atteint, le train passera automatiquement à la gare suivante. Cette fonctionnalité est destinée à continuer un voyage en sens inverse sans avoir besoin d'accéder à des gares manuellement depuis le menu.

⚠ La fonctionnalité "changing ends" est uniquement disponible dans la version 1.2.11 de développement et au-dessus.

---

**Track.Stop** *Direction; BackwardTolerance; ForwardTolerance; Cars*

***Direction***: De quel côté de placer un poteau d'arrêt par défaut. La valeur par défaut est 0.
***BackwardTolerance*** : Un nombre positif (à point flottant) indiquant la tolérance admise dans le sens arrière, **par défaut** mesuré en **mètres**. La valeur par défaut est 5.
***ForwardTolerance*** : Un nombre positif (à point flottant) indiquant la tolérance admise dans le sens avant, **par défaut** mesuré en **mètres**. La valeur par défaut est 5.
***Cars*** : Un entier non négatif indiquant pour combien de voitures ce point d'arrêt s'applique, ou 0 pour toutes les voitures. La valeur par défaut est 0.

▸ Les options disponibles pour *Direction* :

**-1**: Un poteau d'arrêt est créé sur le côté gauche.
**0**: Pas de poteau d'arrêt n'est créé.
**1**: Un poteau d'arrêt est créé sur le côté droit.

Cette commande met un point d'arrêt pour la dernière gare créée. S'il ya plus d'un arrêt défini pour une gare, le train doit s'arrêter à la première commande Track.Stop pour laquelle Cars est supérieur ou égal au nombre de voitures du train, où une valeur pour Cars de 0 indique un point d'arrêt indépendamment de la quantité de voitures utilisé en tant que dernier point d'arrêt d'une gare.

*Exemple d'une station avec plusieurs points d'arrêt :*

▶
```
With Track
0100, .Sta STATION
0178, .Stop 1;;;4 ,; pour 4 ou moins de voitures
0212, .Stop 1;;;6 ,; pour 5 ou 6 voitures
0246, .Stop 1;;;8 ,; pour 7 ou 8 voitures
0280, .Stop 1;;;0 ,; pour 9 ou plus  de voitures
```

---

**Track.Form** *$RailIndex_1$; $RailIndex_2$; RoofStructureIndex; FormStructureIndex*

Description pas encore disponible.

● 11.5. Signalisation et limitation de vitesse

## `Track.Limit` *Speed; Post; Cource*

***Speed***: Un nombre positif (à point flottant) qui représente la vitesse, **par défaut** mesuré en **km/h**, ou 0 indique aucune limitation de vitesse. La valeur par défaut est 0.
***Post***: Le côté sur lequel placer par défaut un poteau de limitation de vitesse de style japonais. La valeur par défaut est 0.
***Cource***: L'indication de direction. La valeur par défaut est 0.


▸ Options pour *Post* :

**-1** : Le poteau est placé sur le côté gauche de la voie.
**0** : Aucun poteau ne sera placé.
**1** : Le poteau est placé sur le côté droit de la voie.

▸ Options pour *Cource*:

**-1** : Le poteau s'applique pour une voie reliée à gauche.
**0** : Le poteau n'indique pas de direction particulière.
**1** : Le poteau s'applique pour une voie reliée à droite.

Cette commande définit la nouvelle limitation de vitesse à partir de ce point. Si la nouvelle limitation de vitesse est inférieure à la limitation de vitesse actuelle, la limitation de vitesse entrera en vigueur immédiatement. Si la limitation de vitesse est supérieure à la limitation de vitesse actuelle, la limitation de vitesse entrera seulement en vigueur une fois que le train tout entier aura dépassé ce point. En réglant *Speed* à `0`, il n'y a pas de limitation de vitesse. En fixant *Post* soit à `-1` ou à `1`, un poteau de vitesse par défaut de style japonais est placé sur le côté respectif de la voie. Le réglage de *Course* soit à `-1` ou à `1` inclus une indication de direction, qui est habituellement utilisé sur les commutateurs de chemins de fer afin d'indiquer que la limitation de vitesse ne s'applique que si la direction respective est prise. Si *Speed* est réglée à `0`, le réglage de *Course* n'a pas d'effet.

---

## `Track.Section` $a_0$; $a_1$; $a_2$; ...; $a_n$

$\mathbf{a}_i$ : Un nombre non négatif spécifiant l'un des aspects de la section.

Cette commande démarre une section, la partie fonctionnelle de la signalisation, qui sera utilisé en conjonction avec Track.SigF, ce qui crée une représentation visuelle d'une section (un signal). Les paramètres $a_i$ précisent les aspects que la section peut supporter. Un aspect de 0 corresponds à une section rouge qui ne doit pas être dépassé par un train. Le terme $a_0$ est obligatoire.

ⓘ Comportement par défaut en opposition à simplifiée d'une section

Il existe deux modes de comportement différents sur la façon d'interpréter les paramètres $a_i$. Le mode peut être réglé par Options.SectionBehavior. Les éléments suivants sont des descriptions séparées pour le comportement par défaut et simplifié.

**Le comportement par défaut :**
Les termes $a_i$ spécifie l'aspect que la section doit avoir selon le nombre de sections à venir qui sont vides jusqu'à ce qu'un rouge soit rencontré. L'ordre des termes est pertinents. Le même aspect peut se produire

plusieurs fois.
▸ Signification des termes $a_i$ :

$\mathbf{a_0}$ : L'aspect montrant quand cette section occupé par un train ou d'une autre manière reste au rouge.
$\mathbf{a_1}$ : L'aspect montrant quand cette section est vide, mais la section qui suit est au rouge.
$\mathbf{a_2}$: L'aspect montrant quand cette section et la section suivante sont vide, mais celle qui suit immédiatement celui-ci est au rouge.
$\mathbf{a_n}$: L'aspect montrant quand *n* sections sont vides avant qu'un signal rouge ne soit rencontré.

Dans le cas ou plusieurs sections suivantes sont vides que celle indiquée par les termes $a_i$, la section aura l'aspect $a_n$.

**Le comportement simplifié :**
Les termes $a_i$ specifie le repertoire que l'aspects de la section peut avoir. Une section portera le plus petit $a_i$ qui est plus grand que l'aspect actuel de la prochaine section. Si aucun exemple $a_i$ existe, la section portera l'aspect $a_n$. L'ordre de $a_i$ n'est pas pertinent. Si le même aspect apparaît plusieurs fois, cela n'a aucun effet.

*Exemple d'une commande Track.Section en conjonction avec une commande Track.SigF :*

▶
```
With Track
1000, .Section 0;2;4, .SigF 3;0;-3;-1
```

---

**Track.SigF** *SignalIndex; Section; X; Y; Yaw; Pitch; Roll*

***SignalIndex*** : Un entier non négatif représentant le signal qui doit être placé tel que défini par Signal(*SignalIndex*).Load.
***Section***: Un entier non négatif représentant la section à laquelle ce signal est attaché, with 0 étant la section courante, 1 la section suivante, 2 la section après celle-ci, et ainsi de suite.
***X*** : La coordonnée X pour placer l'objet du signal, **par défaut** mesuré en **mètres**. La valeur par défaut est 0.
***Y*** : La coordonnée Y pour placer l'objet du signal, **par défaut mesuré en mètres**. La valeur par défaut est 0.
***Yaw*** : L'angle en degrés par lequel l'objet est tourné dans le plan XZ dans le sens des aiguilles d'une montre vu de dessus. La valeur par défaut est 0.
***Pitch*** : L'angle en degrés par lequel l'objet est tourné dans le plan YZ dans le sens des aiguilles d'une montre vu de la gauche. La valeur par défaut est 0.
***Roll*** : L'angle en degrés par lequel l'objet est tourné dans le plan XY dans le sens des aiguilles d'une montre vu par derrière. La valeur par défaut est 0.
Cette commande crée un signal sans fonction, qui est, une représentation visuelle d'une section telle que définie par Track.Section. Le réglage de *Y* à un nombre négatif réinitialise la coordonnée y à 4.8 mètres et attache un signal par défaut. Voir aussi Track.Section.

Si aucun objet n'a été définie par Signal(*SignalIndex*), l'un des signaux japonais par défaut est utilisée :
▸ Signaux par défaut pour *SignalIndex*:

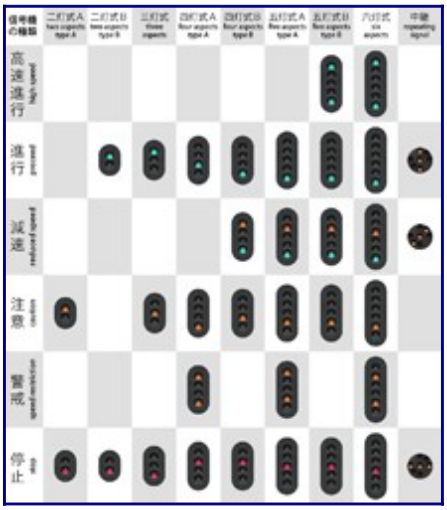
**3** : Un signal trois éléments ayant les éléments ●rouge, ●jaune et ●vert.
**4** : Un signal quatre éléments (type A) ayant les éléments ●rouge, ●●jaune-jaune, ●jaune et ●vert.
**5** : Un signal cinq éléments (type A) ayant les éléments ●rouge, ●●jaune-jaune , ●jaune, ●●jaune-vert et ●vert.
**6**: Un signal répété équivalent à celui créé par Track.Relay.

---

```
Track.Signal Aspects; Unused; X; Y; Yaw; Pitch; Roll
Track.Sig Aspects; Unused; X; Y; Yaw; Pitch; Roll
```

***Aspects*** : Le nombre d'éléments de ce signal. La valeur par défaut est -2.
***Unused*** : *Cet argument n'est pas utilisé par openBVE.*
***X*** : La coordonnée X pour placer l'objet du signal, **par défaut** mesuré en **mètres**. La valeur par défaut est 0.
***Y*** : La coordonnée Y pour placer l'objet du signal, **par défaut mesuré en mètres**. La valeur par défaut est 0.
***Yaw*** : L'angle en degrés par lequel l'objet est tourné dans le plan XZ dans le sens des aiguilles d'une montre vu de dessus. La valeur par défaut est 0.
***Pitch*** : L'angle en degrés par lequel l'objet est tourné dans le plan YZ dans le sens des aiguilles d'une montre vu par la gauche. La valeur par défaut est 0.
***Roll*** : L'angle en degrés par lequel l'objet est tourné dans le plan XY dans le sens des aiguilles d'une montre vu par derrière . La valeur par défaut est 0.

▸ Options pour *Type* :



**2** : Un signal à deux éléments (type A) ayant les éléments ●rouge et ●jaune.
**-2** : Un signal à deux éléments (type B) ayant les éléments ●rouge et ●vert.
**3** : Un signal à trois éléments ayant les éléments ●rouge, ●jaune et ●vert.
**4** : Un signal à quatre éléments (type A) ayant les éléments ●rouge, ●●jaune-jaune, ●jaune et ●vert.
**-4** : Un signal à quatre éléments (type B) ayant les éléments ●rouge, ●jaune, ●●jaune-vert et ●vert.
**5** : Un signal à cinq éléments (type A) ayant les éléments ●rouge, ●●jaune-jaune, ●jaune, ●●jaune-vert et ●vert.
**-5** : Un signal à cinq éléments (type B) ayant les éléments ●rouge, ●jaune, ●●jaune-vert, ●vert et ●●vert-vert.
**6** : Un signal à six éléments ayant les éléments ●rouge, ●●jaune-jaune, ●jaune, ●●jaune-vert, ●vert et ●●vert-vert.

Cette commande crée un signal fonctionnel. Vous pouvez choisir parmi les options disponibles pour *Aspect* pour créer un quelconque signal japonais par défaut. Le réglage de *X* à 0 crée un signal fonctionnel mais invisible semblable à Track.Section. Le réglage de *X* à un nombre non-nul et *Y* à un nombre négatif réinitialise la coordonnée y à 4.8 mètres attache un signal par défaut.

*Exemple d'un signal de quatre élément de type B sans poste à x=-3 et y=5 :*

▶
```
1000, Track.Signal -4;;-3;5
```

*Exemple d'un signal de quatre élément de type B incluant un poste à x=-3 et y=4.8 :*

▶
```
1000, Track.Signal -4;;-3;-1
```

Track.Signal est similaire à l'utilisation de Track.Section et Track.SigF en une seule commande.

---

```
Track.Relay X; Y; Yaw; Pitch; Roll
```

***X*** : La coordonnée X pour placer l'objet du signal, **par défaut** mesuré en **mètres**. La valeur par défaut est 0.
***Y*** : La coordonnée Y pour placer l'objet du signal, **par défaut mesuré en mètres**. La valeur par défaut est 0.
***Yaw*** : L'angle en degrés par lequel l'objet est tourné dans le plan XZ dans le sens des aiguilles d'une montre vu de dessus. La valeur par défaut est 0.
***Pitch*** : L'angle en degrés par lequel l'objet est tourné dans le plan YZ dans le sens des aiguilles d'une montre vu par la gauche. La valeur par défaut est 0.
***Roll*** : L'angle en degrés par lequel l'objet est tourné dans le plan XY dans le sens des aiguilles d'une montre vu par derrière . La valeur par défaut est 0.

Cette commande crée un signal répétant japonais par défaut. Le signal répétant répète l'état du signal à venir. Mise à zéro de X ne crée pas un signal répétant, mais forces la commande à être ignoré. Le réglage de X à un nombre non-nul et Y à un nombre négatif réinitialise la coordonnée y à 4,8 et attache un poste de signaux par défaut.

● 11.6. Systèmes de sécurité

---

```
Track.Beacon Type; BeaconStructureIndex; Section; Data; X; Y;
Yaw; Pitch; Roll
```

***Type*** : Un entier non négatif représentant le type de balise qui sera transmise au train.
***BeaconStructureIndex*** : Un entier non négatif représentant l'objet qui sera placé tel que défini par Structure.Beacon, ou -1 pour ne pas placer d'objet.
***Section*** : Un entier représentant la section dans laquelle la balise est fixé, à savoir 0 pour la section courante, 1 pour la section suivante, 2 pour la section suivante, etc., ou -1 pour la prochaine section rouge.
***Data*** : Un entier représentant des données arbitraires spécifiques au type de balise qui sera transmise au train.
***X***: La coordonée X à laquelle placer l'objet, **par défaut** mesuré en **metres**. La valeur par défaut est 0.
***Y***: La coordonée Y à laquelle placer l'objet, **par défaut** mesuré en **metres**. La valeur par défaut est 0.
***Yaw*** : L'angle en degrés par lequel l'objet est tourné dans le plan XZ dans le sens des aiguilles d'une montre vu de dessus. La valeur par défaut est 0.
***Pitch*** : L'angle en degrés par lequel l'objet est tourné dans le plan YZ dans le sens des aiguilles d'une montre vu par la gauche. La valeur par défaut est 0.
***Roll*** : L'angle en degrés par lequel l'objet est tourné dans le plan XY dans le sens des aiguilles d'une montre vu par derrière . La valeur par défaut est 0.

Cette commande place une balise (transpondeur). L'objet doit avoir été chargé par Structure.Beacon (*BeaconStructureIndex*) avant d'utiliser cette commande. Quand le train passe devant la balise, le type de balise et les différentes données sont transmises au train, y compris l'état de la section référencée.

Il est à noter que les systèmes intégrés de sécurité reçoivent également des données à partir de ces balises, que Track.Beacon(*Type*) est à peu près équivalent à Track.Transponder(*Type*). S'il vous plaît voir la page sur les normes de beacon pour plus d'informations.

---

```
Track.Transponder Type; Signal; SwitchSystem; X; Y; Yaw; Pitch;
Roll
Track.Tr Type; Signal; SwitchSystem; X; Y; Yaw; Pitch; Roll
```

***Type*** : Le type du transpondeur. La valeur par défaut est 0.

***Signal*** : Le signal de référence du transpondeur. La valeur par défaut est 0.
***SwitchSystem*** : Pour passer automatiquement le système de sécurité. La valeur par défaut est 0.
***X***: La coordonée X à laquelle placer l'objet, **par défaut** mesuré en mètres. La valeur par défaut est 0.
***Y***: La coordonée Y à laquelle placer l'objet, **par défaut** mesuré en mètres. La valeur par défaut est 0.
***Yaw*** : L'angle en degrés par lequel l'objet est tourné dans le plan XZ dans le sens des aiguilles d'une montre vu de dessus. La valeur par défaut est 0.
***Pitch*** : L'angle en degrés par lequel l'objet est tourné dans le plan YZ dans le sens des aiguilles d'une montre vu par la gauche. La valeur par défaut est 0.
***Roll*** : L'angle en degrés par lequel l'objet est tourné dans le plan XY dans le sens des aiguilles d'une montre vu par derrière . La valeur par défaut est 0.

▸ Options pour *Type* :


**0** : Un transpondeur de type S utilisé par ATS-S. Habituellement placé à 600m en face d'un signal.
**1** : Un transpondeur de type SN utilisé par ATS-SN. Habituellement placé à 20m en face d'un signal.
**2** : Un transpondeur de départ accidentel. Habituellement placé juste après une gare avec arrêt.
**3** : Un transpondeur ATS-P de renouvellement. Habituellement placé à 600m, 280m, 180m, 130m, 85m ou 50m en face d'un signal, selon les circonstances.
**4** : Un transpondeur ATS-P d'arrêt immediat. Habituellement placé à 25m ou 30m en face d'un signal, en fonction des circonstances.

▸ Options pour *Signal* :

**0** : Le prochain signal est référencé.
**1** : Le signal immédiatement après le prochain signal est référencé.
***N*** : Le *n*'ième signal derrière le prochain signal est référencé.

▸ Options pour *SwitchSystem* :

**-1** : Le transpondeur ne bascule pas le train entre ATS-SN et ATS-P.
**0** : Le transpondeur bascule automatiquement le train vers ATS-SN pour le transpondeur de types *0* et *1*, et vers ATS-P pour les types *3* et *4*.

Cette commande place un transpondeur, généralement pour la sécurité intégrée des systèmes ATS-SN or ATS-P. Pour plus d'informations sur ces systèmes et leurs transpondeurs, voir la documentation de l'utilisateur sur ATS.

Il est à noter que de coutume le plugins du système de sécurité recoit également des données provenant de ces transpondeurs Track.Transponder(*Type*) qui est à peu près équivalent à Track.Beacon(*Type*). S'il vous plaît consultez la page sur les normes beacon pour plus d'informations.

➟ Allez ici pour en savoir plus sur ATS-SN et ATS-P.
➟ Il ya un tutoriel disponible pour la bonne utilisation de l'ATS-SN et ATS-P dans les fichiers route, y compris tous les cinq transpondeurs.

---

## `Track.AtsSn`

Cette commande place d'un transpondeur de type S pour le système intégré de sécurité ATS-SN, référençant le prochain signal, et basculant automatiquement vers ATS-SN. La commande est equivalente à **Track.Tr 0;0;0**. Voir là pour plus d'informations.

---

### `Track.AtsP`

Cette commande place un modèle de transpondeur de renouvellement pour le système intégré de sécurité ATS-P, référençant le prochain signal, et basculant automatiquement vers ATS-P. La commande est equivalente à **Track.Tr 3;0;0**. Voir là pour plus d'informations.

---

### `Track.Pattern Type; Speed`

***Type*** : Le type de limitation de vitesse.
***Speed*** : Un nombre non-négatifs (à point flottant) qui représente la limitation de vitesse, **par défaut** mesuré en **km/h**.

▸ Options pour *Type* :

**0**: Une limitation temporaire de vitesse.
**1**: Une limitation permanente de vitesse.

Cette commande définit une limitation de vitesse pour la sécurité intégrée du système ATS-P.

Une limitation temporaire de vitesse (*Type*=0) doit être inséré à l'endroit où la limitation de vitesse doit s'appliquer. ATS-P sera au courant de cette restriction de vitesse à l'avance et freinera le train afin que le train respecte la limitation de vitesse à ce point. Une fois le point est adopté, la limitation de vitesse ne s'applique plus.

Une limitation permanente de vitesse (*Type*=1) doit être inséré à l'endroit où la limitation de vitesse doit s'appliquer, cependant, ATS-P ne connaît pas cette limite à l'avance et freinera seulement le train à partir de ce point. Pour un degré supérieur de réalisme, insérer les limitations de vitesse permanentes au même point que les transpondeurs ATS-P. Une limitation de vitesse permanente, comme son nom l'indique, est mémorisé par ATS-P et n'est libéré que par une limitation de vitesse permanentes ultérieures.

---

### `Track.PLimit Speed`

***Speed***: Un nombre non-négatifs (à point flottant) qui représente la limitation de vitesse permanente pour ATS-P, **par défaut** mesuré en **km/h**.
Cette commande est équivalente à **Track.Pattern 1;*Speed***. Voir là pour plus d'informations.

● 11.7. Divers

---

### `Track.Back BackgroundTextureIndex`

***BackgroundTextureIndex***: Un entier non négatif représentant l'image de fond qui sera affiché tel que défini par Texture.Background(*BackgroundTextureIndex*).
Cette commande définit l'image de fond qui s'affichera à partir de maintenant.
⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

```
Track.Fog StartingDistance; EndingDistance; RedValue; GreenValue; BlueValue
```

***StartingDistance***: Un nombre (à point flottant) indiquant le début du brouillard, **par défaut** mesuré en mètres. La valeur par défaut est 0.
***EndingDistance***: Un nombre (à point flottant) indiquant la fin du brouillard, **par défaut** mesuré en mètres. La valeur par défaut est 0.
***RedValue***: Un nombre entier allant de 0 à 255 représentant la composante rouge du brouillard. La valeur par défaut est 128.
***GreenValue***: Un nombre entier allant de 0 à 255 représentant la composante verte du brouillard. La valeur par défaut est 128.
***BlueValue***: Un nombre entier allant de 0 à 255 représentant la composante bleu du brouillard. La valeur par défaut est 128.
Cette commande définit le brouillard à ce point , ou désactive le brouillard. Si le brouillard doit être activé, *StartingDistance* doit être inférieure à *EndingDistance*. Si le brouillard doit être désactivé, *StartingDistance* doit être supérieur ou égal à *EndingDistance*.

Le brouillard affecte la coloration des objets. Les objets à une distance du départ apparaissent tels quels, les objets après cette distance apparaissent dans la couleur du brouillard, et les objets entre les deux mélanges linéairement entre celles-ci. L'image de fond est affectée par le brouillard aussi. Pour les calculs du brouillard, l'image de fond est supposé être à 600 mètres de distance de la caméra, indépendamment de la distance de vue réelles.

En fonction de Options.FogBehavior, il ya deux options sur la manière que cette commande affecte le brouillard dés ce point. Dans le mode block-wise, le brouillard actuel mélanges pour les nouveaux paramètres du début de ce bloc à la fin de ce bloc. Le nouveau réglage est conservé pour les blocs suivants. C'est le comportement par défaut. En mode d'interpolation, chaque commande Track.Fog définit un point de contrôle pour le brouillard, où tous les paramètres (les distances et les couleurs) sont interpolées linéairement entre les points de contrôle.
⚠ Cette commande ne peut être utilisé au début d'un bloc.

---

```
Track.Brightness Value
```

*Value*: Un entier non-négatif dans la gamme de 0 à 255. La valeur par défaut est 255.
Cette commande marque un point qui affecte la luminosité dans la cabine. *Value* est mesurée de 0 (noir) à 255 (la lumière), et c'est une interpolation linéaire entre les commandes Track.Brightness successives pour n'importe quel point donné sur la voie. Cette commande doit être utilisé pour les tunnels, les ponts, les toits des gares, ou toute autre chose qui aurait une incidence sur la luminosité perçue à l'intérieur de la cabine.

*Exemple:*

```
With Track
1200, .Brightness 255 ,; avant le début du pont
1205, .Brightness 128 ,; directement sous le pont ici
1210, .Brightness 255 ,; dès que le pont se termine
```

---

```
Track.Marker FileName; Distance
```

***FileName***: Le nom du fichier pour l'image du marqueur, relatif au dossier **Object**.
***Distance***: Un nombre non-nul (à point flottant) indiquant la durée pour laquelle l'image du marqueur est

affiché, **par défaut** mesuré en **mètres**.

▸ Comportement pour *Distance* :

*Valeur négative* : L'image du marqueur commence à s'afficher à la commande Track.Marker, et se termine -*Distance* mètres après la commande Track.Marker.
*Valeur positive* : L'image du marqueur commence à s'afficher à *Distance* mètres avant la commande Track.Marker et se termine à la commande Track.Marker.

Cette commande affiche une image du marqueur qu'on appelle, et qui est affiché dans le coin en haut à droite de l'écran. Vous pouvez utiliser ces images à des fins de consultation ou d'information. La couleur RVB de 64,64,64 dans l'image est transparente.

---

```
Track.PointOfInterest RailIndex; X; Y; Yaw; Pitch; Roll; Text
Track.POI RailIndex; X; Y; Yaw; Pitch; Roll; Text
```

***RailIndex***: Un entier non négatif représentant le rail pour le point de vue intéressant.
***X***: Un nombre (à point flottant) qui représente le décalage horizontal par rapport au rail, **par défaut mesuré en mètres**. Les valeurs négatives indiquent la gauche, les positives la droite.
***Y***: Un nombre (à point flottant) qui représente le décalage vertical par rapport au rail, **par défaut mesuré en mètres**. Les valeurs négatives indiquent dessous, les positifs dessus.
***Yaw*** : L'angle en degrés par lequel l'objet est tourné dans le plan XZ dans le sens des aiguilles d'une montre vu de dessus. La valeur par défaut est 0.
***Pitch*** : L'angle en degrés par lequel l'objet est tourné dans le plan YZ dans le sens des aiguilles d'une montre vu par la gauche. La valeur par défaut est 0.
***Roll*** : L'angle en degrés par lequel l'objet est tourné dans le plan XY dans le sens des aiguilles d'une montre vu par derrière . La valeur par défaut est 0.
***Text***: Une représentation textuelle du point de vue intéressant.

Cette commande crée un point de vue intéressant auquel l'utilisateur peut sauter à en appuyant sur la touche CAMERA_POI_PREVIOUS (NUM 1) ou CAMERA_POI_NEXT (NUM 7). La caméra sera placée à l'emplacement spécifié à l'orientation spécifiée. Si *Text* n'est pas vide, un message apparaîtra brièvement montrant le texte.

---

```
Track.PreTrain Time
```

***Time***: L'heure à laquelle le train qui précède est à cette position de voie.

Cette commande crée une association position-temps pour un train invisibles précédents afin d'influencer la signalisation. Contrairement à un vrai train qui précède créé par Route.RunInterval, le train invisibles précédant créées par Track.PreTrain est un moyen de scripts où le train précédent est invisible à un moment donné. Les associations position-temps doivent être en ordre croissant, qui est, à une position de voie plus tard, le temps associé doit également être plus tard. Avant le premier temps scénarisé, le train invisibles qui précède réside à la première position scénarisé. Entre le premier et le dernier temps scénarisé, le train invisibles précédents se déplace (linéaire) entre les points scénarisés. Après le dernier temps scénarisé, le train invisibles précédent est retiré et supprime ainsi la signalisation.

---

```
Track.Announce FileName; Speed
```

***FileName*** : Le nom du fichier son à jouer, relatif au dossier **Sound**.
***Speed*** : La vitesse de référence en km/h pour une vitesse dépendante du sons, ou 0 joue le son indépendamment de la vitesse. La valeur par défaut est 0.

Cette commande lit une annonce ou tout autre type de son dans la cabine une fois que le train du joueur traverse le point où cette commande est utilisée. Si *Speed* est réglée à 0 (par défaut), le son est joué tel quel. Mais si *Speed* est donnée, le son est joué au ton d'origine de la vitesse spécifiée, et le ton est modulée

proportionnellement pour d'autres vitesses, utile pour les sons custom flange, les sons pointwork, etc.

---

`Track.Doppler FileName; X; Y`

***FileName***: Le nom du fichier pour le son à jouer, relatif au dossier **Sound**.
***X***: Un nombre (à point flottant) qui représente le décalage horizontal par rapport au rail, **par défaut mesuré en mètres**. Les valeurs négatives indiquent la gauche, les positives la droite.
***Y***: Un nombre (à point flottant) qui représente le décalage vertical par rapport au rail, **par défaut mesuré en mètres**. Les valeurs négatives indiquent dessous, les positifs dessus.

Cette commande place un effet sonore d'ambiance à l'emplacement spécifié. Le son sera joué en boucle pendant toute la durée de la simulation et emploie l'effet Doppler. (Remarque: Tous les sons dans openBVE emploient l'effet Doppler)

---

`Track.Buffer`

Cette commande place un heurtoir. Le train peut entrée en collision avec le heurtoir à la fois dans les directions avant et arrière. Placez cette commande au début et à la fin de la route. Un objet n'est pas automatiquement créé, utilisez donc Track.FreeObj pour créer une représentation visuelle du heurtoir si nécessaire.