GTS - Ginren Traffic System

銀河連邦

2025/06/04

Table of Contents

# はじめに

この文書は、GTS - Ginren Traffic System（以降、GTSとします。）で適用することのできる拡張モデルパック（以下、拡張パックあるいはパックとします）の作成方法（仕様）、及び交通信号制御機等を始めとする各種データの仕様について記載しています。

この文書はユーザーガイドライン・マニュアルとしては作成されていません。パックを作成するつもりがなく、単にGTSを利用したいだけの場合は、こちらではなく別途用意しているユーザー向けマニュアルをお読みください。

開発に伴いなるべく仕様書も最新に整えるようにするつもりですが、仕様書と現況に差異が生じる場合があります。また、バージョンは最新版を想定していますが、過去のバージョンから修正・削除されたものなどについてはその旨を明記するとします。

画像が不足していますがおいおい追加します。また、この文書はマークダウンを自動変換して作成しているため、ところどころお見苦しい点がありますがご了承ください。

# パックについて

GTS内で追加される各交通機器（一部例外を除きます）において、ユーザーが独自に作成したモデル等を適用することができます。その際、決められた形式に従って「パック」を作成する必要があります。このパックは、RealTrainMod（RTM）のモデルパックに相当するもので、こちらの作成経験がある場合は概ねその知識が活かせると思います。ただし互換性はありません（ファイル形式などは同一ですが）。

中身はZIPファイルなので誰でも作成できます。スマートフォンでも頑張れば作れます。

## パックでは何が変更できるのか

GTSにパックを適用すると、例えば以下のような変更が可能となります。

* ほぼすべての交通機器（信号機、制御機、押ボタン箱など）のモデル（見た目）を変更することができます。
* 押ボタン箱等の場合、必要に応じて自前の音声を流すことができます。
* （現時点未対応：上級者向け）Javaのクラスを追加することで、GTSそのものの仕組みを一部変更することができます。

## パックに含むことのできるファイル

GTSが認識できるファイルとしては、以下のものに限られます。

### モデリングファイル（3Dオブジェクト）

現在、MQOファイル（Metasequoiaで作成できるもの）のみ対応しています。Metasequoia 4.xの形式に準拠しており、読み込みとしては「マテリアル」「オブジェクト」「テクスチャUV」「スムージング角度」のパラメータを読み込むことができます。

MQOファイルをお持ちでない方（UnityだとFBX、他にはobjなど）は、お手数ですがMQOファイルに変換の上ご利用ください。

### テクスチャファイル

GTSでは、png形式を推奨していますが、jpg形式も読み取ることができます。というより、内部ではImageIOを呼び出しているだけなので、これが対応している形式であれば何でも読み取ることができます。

### 音声ファイル

Minecraftの制約上、OGGファイルのみに対応しています。MP3やWAV等をお持ちの方は、Audacity等をダウンロードしてご利用いただくと変換が可能です。多分、コーデックは何でも対応しているんじゃないかと思っていますが、読み取れなかったらコーデック部分を見直してみてください。

### JSONファイル

各モデルの設定ファイルを記載するためのものとなります。JSONが分からない方はそもそも話になりませんので、検索して学習して下さい。ここでは説明は割愛します。

### gts.txt

テキストファイルとして含めることのできる唯一のファイルで、大文字小文字を含めてこのファイル名で統一する必要があります。メタデータを格納する場所ですが、詳しくは後述します。

これ以外のファイルもパックに含めることはできますが、GTSで読み込む際には無視されます。逆にいえば、不要なファイルが入っていてもGTSは問題なくパックを認識します。

# パックの作り方

注意：ここで紹介しているGTS Pack Makerは、2025年6月9日時点で開発中です。

GTSでは、パックを少しでも作り易く、管理しやすくするためにGTS Pack Makerというパック管理ツールを使用することができます。こちらに沿って作っていただければ、確実に仕様に即したものができあがります。このツールは、ダウンロードしたjarファイルをダブルクリックすることで実行できます。詳しい使い方については、別途用意しているツール用マニュアルをご覧ください。

ここでは、GTSパックとして認められるために必要な手順をざっくり掲載します。

## ワーキングディレクトリを作成する

パックのファイル群を格納するため、どこでもいいのでワーキングディレクトリを作成しましょう。パックに使用するファイルは全てこのディレクトリの中に置くようにしてください。このディレクトリの中であれば、いくらでも階層に分けて整理して問題ありません。好きなところにファイルを置いてください。

なお、ここに置くファイルは原則半角英数字とアンダーバーのみで名前を決めることを強くお勧めします。マルチバイト文字（全角）を入れると想定しない動きを取る場合があります（特に音声ファイルはほぼ100％読み込みに失敗します）。

## モデルを作る

お好きなモデリングソフトウェアを使用して、3Dモデルを作成してください。なお、3Dモデルは比較的自由に作成頂けますが、適用する交通機器によっては追加の制約が課せられることがあります。共通で推奨すべき項目について、以下に記します。

### ファイル形式

ファイル形式は前章の通り、MQOファイルのみを使用することができます。どのようなモデリングソフトウェアを使用していただいても構いませんが、必ずMQOファイルでの出力を行ってください。以下、MQOファイルであること、Metasequoiaを使用していること前提で記載します。

### サイズ

サイズに関しては、この後のステップでJSONファイルを作成する際に指定できますので、ここでは特に意識する必要はありません。好きなサイズで作成してください。ただし、極端に小さなサイズで作成すると、リサイズ時の計算に誤差が生じてモデルの形状が破綻する恐れがあります。最大・最小頂点が小数点第3位くらいまでに収まるようにすレ場問題ないので、たいていの場合問題にはなりません。

### 座標（位置）

基本的に設置時は強制的に原点中心に移動しますのでどの位置に作成していただいても構いませんが、モデリングの際は原点中心で行うことをお勧めします。というのも、ミラーリングなどを使用する場合はおのずと原点中心のほうがやりやすいと思います。

意図的にモデルをずらしたい場合はコンフィグファイルでデフォルトの描画位置を指定することができます。

### テクスチャ

テクスチャはUVを使用してマッピングしてください。UVマッピングは0～1の間で座標指定されるので、あまり細かく指定すると破綻するかもしれません（小数点第6位くらいまでは平気です）。平面や曲面などといった指定もありますがこちらは未対応です。必ずUVで設定してください。また、**すべての面に対して**UVを割り当ててください。UVが貼られていない面が存在する場合、描画に失敗することがあります。

GTSで使用できるテクスチャは原則1枚（信号機などを除く）なので、すべての面のテクスチャは1つのテクスチャに収める必要があります。テクスチャのファイルパスはコンフィグファイルで定めますので、モデリングの際にはどこにいようと構いません。

### 面

4.x以降のMetasequoiaでは、多角形の面を扱うことができます。GTSでは、**三角面あるいは四角面**のみで作成することを強く推奨します。設計上は多角形の面も扱うことができるようにしていますが、動作確認を全く行っていないため多角形で面を作成したファイルを使用した場合についてはサポート対象外とさせていただきます。

デフォルトでは面は表面しか描画されませんので、表裏で面を別々に登録したい場合はなんかこういい感じに重ねてください。今後対応するかは未定です。

### オブジェクト

交通信号機、交通信号柱、アームなどはこのオブジェクトごとに描画の有無を判断します。オブジェクト名を変更して、適切にコンフィグを書けるように整理してください。Metasequoia4.xからは階層化もできますが、階層化してもGTS読み込み時には（多分）すべて無視します。

オブジェクト名はいらない事故を避けるため、半角英数字で構成することを強くお勧めします。また、交通信号制御機はこのオブジェクト名を基準に光らせる現示を決定するため、各モデルパックではオブジェクト名を統一しておくとユーザーが使いやすくなります。

参考までに、交通信号機の場合、当社では以下のような命名規則を用いています。

| オブジェクト名 | 内容 |
| --- | --- |
| body | 現示と関係ない、常に描画されるもの。 |
| (g|y|r)[250|300] | 各色のレンズ部分。ここが現示指定時に点灯する。例：g300→300mmの青 |
| (g|y|r)[normal|dl|dy|over]h | 各色の庇。左から順に、ノーマル ダブルフード（左穴） ダブルフード（右穴） 筒型 |

制御機のサイクル情報を一緒に提供しない場合は、各現示がどのオブジェクトに対応しているかの説明や資料を同梱することでユーザーが理解しやすくなります。

ここまでできたら、モデルファイルをワーキングディレクトリ内に放り込みましょう。modelsというディレクトリを作ってその中に入れると整理しやすいです。

## テクスチャを放り込む

モデルには最低1枚のテクスチャが必要となります。テクスチャを作成し（モデルの作成時に使用したテクスチャで構いません）、pngあるいはjpg（非推奨）形式にして、ワーキングディレクトリに放り込みます。

テクスチャのサイズは問いませんが、目安としては1024×1024以下となります。もちろん高画質なテクスチャを使用することもできますが、比例してメモリ使用量も上がるので注意してください。

Minecraftは描画にOpenGLを使用しており、その制約上テクスチャは**2の累乗**ピクセル四方で作成するとパフォーマンスが向上します。そうでなくても動きますができる限りこのサイズで作成することをお勧めします。よくわからない方は以下のどれかで作成してください。

* 16x16　…　Minecraftのデフォルトテクスチャがこのサイズ。ドット感強め。
* 32x32　…　解像度が圧倒的に足りないので、単色のモデルとか向き。
* 64x64　…　1つのテクスチャを全面で利用する場合はここが最小限。
* 128x128　…　遠目で見る分には問題ないが、近くに来るとすぐ粗が目立つ。
* 256x256　…　普通にプレイする分には特に問題ない。細部はやや粗が目立つ。
* 512x512　…　ここまで解像度があれば基本的にスクショにも耐えうる。
* 1024x1024　…　細かなレンズの模様や小さな文字などを再現したい場合に使用。メモリの使用量が若干多い。
* 2048x2048　…　これ以上は4Kとかで間近で見るとかしない限りオーバースペックとなる。

### なぜJPGは非推奨？

JPG（JPEG）ファイルは、テクスチャ1枚当たりの容量を大きく減らすことができるため、一見メモリの節約をできると思うかもしれません。ですが、OpenGLは結局展開すると（誤解を恐れずにざっくりいえば）ピクセル単位のデータに変換します。なのでPNGだろうがJPGだろうが使用するメモリの量は（サイズが同じなら）同等となります。JPGは容量を小さくする関係上非可逆圧縮という、画質が低下する処理を行っているため、テクスチャの画質が粗くなります。ならば初めからPNGつかっておけばいいよね、という話です。

余談ですがBMPも多分使えます。ただWindows専用でレガシーな保存方法のため使用しない方が吉です。

## （任意）音声ファイルを作成する

押ボタン箱、音響装置（現在開発予定段階）は、その機器が音声を出すことができます。その再生する音声ファイルを用意しましょう。信号機や制御機の場合はこのステップは不要です。

音声ファイルはoggのみ対応しているので、この形式に変換してください。Audacityとか使用するのがおすすめです。

作成した後は、音声ファイルをワーキングディレクトリに放り込んでください。

## コンフィグファイルを作成する

各モデルとテクスチャ、音声ファイルを紐づける設定ファイルとして、コンフィグファイルを作成します。ここが一番パック制作初心者には難しいかと思われます。

JSON書式を使用して、JSONファイルを作成します。拡張子は小文字で.jsonとする必要があります。これ以外の拡張子はJSON書式で書いても読み込まれないので注意してください。

コンフィグファイルの中に記載するJSONキーと対応する値に関しては、次章で解説します。コンフィグファイルを作成できたらワーキングディレクトリに放り込んでください。

以上のステップを、必要な数だけ繰り返してください。

## （推奨）gts.txtを作成する

このファイルは、パックのメタ情報を入力するために作成されます。キーと値を=で結び、各プロパティを指定します（JavaのPropertiesと似た形式です）。ただしコメントは付与できません。

このファイルは任意で追加できますが、個人利用にとどまる場合などを除き入れておくことをお勧めします。特に、nameとauthorは入れておくことで、バージョンアップに伴いパックが正常に読み込まれなくなることを防止します。

### 例

name=GTSTestPack  
author=Gingarenpo  
  
meta1=Hoge  
meta2=Fuga

nameに関しては、同一環境内においてかぶらない名称を付与してください。この名称がワールドの各交通機器に保存されるため、この名前を変更するとMissing状態になりダミーファイルに化けますのでご注意ください。このキーの指定がない、あるいはそもそもgts.txtを作成していない場合はデフォルトでZIPファイルから拡張子を除いた名前が使用されることになります。バージョンアップしたときに名前変わったりすることがあるので、nameは必ず入れておいてください。

またnameはリソースパックをダミーで用意する都合上、ResourceLocationの制約に伴って大文字箱文字に変換され、スペースは無効となります。なので、小文字の羅列で記載することをお勧めします。数字もOKですが先頭に追加することはできません。

authorは現状使用されませんが将来的に使用する計画がありますので、可能なら入れておくことをお勧めします。こちらは全角表記でも構いません。

それ以外の情報も登録でき、そちらはメタデータとして扱われることになります。もしかしたらどこかで参照できるようになるかもしれませんが、現時点では特に使用箇所はありません。

## ZIP圧縮する

必要なファイルをワーキングディレクトリに全て放り込めたら、**ワーキングディレクトリに入ってから**ファイル・ディレクトリを全選択してZIPファイルに圧縮してください。**ワーキングディレクトリを**圧縮しないでください。

ZIP圧縮できたら、一度中身を確認してください。Windowsの場合はZIPファイルをダブルクリックすることで中身を見ることができます。

ZIPファイルを開いて最初に出てきた（ZIPファイル直下）ファイルやディレクトリが**ワーキングディレクトリの内部と**一致していることを確認してください。もし、ワーキングディレクトリそのものが入っている場合は、正しくZIP圧縮できていませんので、やり直してください。

ないとは思いますが暗号化すると開けなくなり落ちるので暗号化はしないでください。7-Zipを使用する場合、7z形式での圧縮は行わないでください。

これでパックが完成しました。あとは、このパックをお使いのMinecraftディレクトリのmods/GTSに入れて動作確認をしたり、配布したり煮たり焼いたりしてください。

# コンフィグの共通設定

コンフィグファイルは、原則以下のようなキーと値が含まれている必要があります。ただし、各交通機器によって使用が異なるものも存在しますので、その点については各交通機器の仕様をご覧ください。

なお、ファイルパスと記されているものは、ZIPファイル直下を基準とした相対パスで指定します。models/test\_model.mqoのような形になります。先頭にスラッシュを付けないでください。

## 各キーと値の対応表

| 設定項目名称 | 概要 | 備考 |
| --- | --- | --- |
| type | このコンフィグがどの交通機器の設定ファイルなのかを指定するもの。 | 文字列で指定。指定可能なものはタイプ一覧を参照 |
| textures | このモデルが使用するテクスチャのファイルパスを指定する。中身はオブジェクトで、さらにその中にあるbaseキーにファイルパスを指定する。 | 交通機器によってbase以外の指定を行う必要があるものもある |
| id | このコンフィグの固有IDを文字列で指定する。 | 半角英数字で、**同一パック内で重複しないこと**。 |
| model | モデルのファイルパスを指定する。 |  |
| size（任意） | このモデルがワールドに設置されるときの大きさを、浮動小数で指定する。 | 単位はブロック。1.0と指定すると1ブロック分の大きさになる。このサイズは、最も長い辺がそのサイズになるように調整される。デフォルトは1.0のことが多いが指定することをお勧め |
| opacity（任意） | 現在未使用。何を入れても無視される。 | 将来的に透明度の対応をするために予約しているもの |
| originalPosition（任意） | 通常、モデルは原点基準で描画されるものをXYZ軸方向それぞれにどれだけずらすかを指定する。配列でXYZそれぞれを浮動小数で指定する。例：[0.0, 0.5, 0.0] | デフォルトはすべて0 |

順番は問いません。ただし、任意と書かれているもの以外の全てのキーを必ず含むようにしてください。そうしないと読み込まれない場合があります。

## type一覧

typeに入れるべき内容は以下となります。これ以外を入れると読み込まれません。

「未対応」と書いているものは実装していないので無視されます。

| 項目名 | 交通機器 |
| --- | --- |
| light | 交通信号機 |
| controller | 交通信号制御機 |
| button | 押ボタン箱 |
| pole | ポール |
| arm | アーム |
| sensor | 感知器（現時点で未対応） |
| sound | 音響装置（現時点で未対応） |

# 交通信号制御機

最終更新日時：2025年6月9日

追加バージョン：2.0 alpha1

交通信号制御機（以下、制御機と記します）は、以下のクラスにより構成されています。

* GTSBlockTrafficController → ブロックインスタンス
* GTSTileEntityTrafficController → 制御機のTileEntity
* GTSTileEntityRendererTrafficController → 制御機のレンダラー
* GTSTrafficControllerConfig → 制御機の個別情報（パック）

~~Version 2.0-alpha2の時点では、制御機のモデル変更には対応していません。~~ Version 2.0-alpha3より対応しました。

## パックJSON（コンフィグ）の仕様

現時点で、制御機は共通の設定項目以外の独自設定項目は存在しません。

## 制御方式と用語

制御機は、サイクル フェーズ チャンネル の3つの要素を用いて、信号機の制御を模倣します。この概念について、以下のような「押ボタン式横断路」の動作を整理したものを用いて説明します。

| No. | 概要 | 車灯現示 | 歩灯現示 |
| --- | --- | --- | --- |
| 1 | 初期現示 | 青 | 赤 |
| 2 | 押ボタンが押された時 | 黄 | 赤 |
| 3 | 一定時間後 | 赤 | 赤 |
| 4 | 歩行者が横断可能 | 赤 | 青 |
| 5 |  | 赤 | 青点滅 |
| 6 | この後1に戻る | 赤 | 赤 |

### サイクル

サイクルとは、交通信号機の現示をひとまとめにしたものを指します。上記の表そのものがサイクルとなります。

制御機は、1つ以上のサイクルを紐づけることができます。

制御機は、1Tickごとに初回に紐づけられているサイクルを順に探索し、起動できるサイクルが発見された場合はそのサイクルを起動します。もし、現段階で起動できるサイクルが存在しなかった場合は、次回の判定に持ち越されます。

サイクルは一般的に**排他**であるべきであり、***同時間帯同条件で起動できるサイクルを2つ以上同一の制御機に登録するのは推奨されません***。この場合、どのサイクルが選ばれるかは不定です。

### フェーズ

フェーズとは、ある時点での各灯器における現示の状態を指します。上記の表で言えば、行に相当します。

サイクルは、1つ以上のフェーズを紐づけることができます。逆にいえば、最低1つはサイクルにフェーズを紐づける必要があります。空のフェーズが指定された場合、サイクルの動作が行われないどころか落ちる危険性があります。

制御機は、サイクルの起動中に1Tickごとにフェーズの終了条件をチェックします。フェーズが終了条件を満たす場合は、そのフェーズを終了し、次のフェーズに移動します。もし、現在起動しているサイクルにこれ以上続行するフェーズがない場合は、サイクルを終了します。

### チャンネル

チャンネルとは、あるフェーズにおける各灯器の現示を、種類別に分類したものを指します。上記の表で言えば、1列（車灯、歩灯）に相当します。

フェーズに対して直接現示を指定すると、1フェーズにつき1つしか現示が登録できなくなり、信号制御の意味が9割9分なくなります。そのため、チャンネルという概念を利用して、フェーズ内で異なる現示を実現することができます。

フェーズは、1つ以上のチャンネルを紐づけることができます。チャンネルを1つも登録しないということも可能ですが、そのようなフェーズはまったく意味をなさないので、特殊用途を除き通常は1つ以上を行うことになります。

上記のような概念を組み合わせることによって、現実世界で実施されている制御を完全に模倣することができます。

## 制御情報のJSONファイルについて（中級者向け）

制御機は、ArrayListによってサイクル・フェーズ・チャンネルを管理していますが、これを利用者はGUIからJSONファイルを用いてロード・セーブすることができます。拡張子が.gtsとなっていますが、中身は純粋なJSONファイルとなっています。

JSONファイルは共有用ファイルとして読み書きができるようにしているにすぎないため、1からJSONファイルを作成するのはかなり高度な知識を要求されます。内部でやっていることは、GTSCycleクラスとGTSPhaseクラスのインスタンスをもとに、そのフィールドの内容を正確に保存しているものとなります。

当社では、プリセットとしていくつかJSONファイルを作成して配布していますので、よくわからない場合はそちらを使用してください。

## カスタム制御スクリプトについて（上級者向け）

既存の組み合わせなどでは表現しきれないような非常に高度な制御を実現する場合、Javascriptを使用してサイクルをプログラミングチックに表現することができます。正確には、起動条件や次のフェーズ等を外部から直接指定することができるものになります。

なお、この機能は開発用に使用していたものを一般開放したものであり、外部のスクリプトに頼る以上当社の責任範囲を超過した者も含まれます。利用は自己責任でお願いいたします。

### カスタムスクリプトの仕様

先述の通り、Javascriptを用いて記述します。必須となる関数を定義する必要があり、この関数が存在しないとエラーでサイクルの実行がされません。また、文法エラーなどが発生した場合もサイクルの実行がされません。

| 関数名 | 概要 | 戻り値 |
| --- | --- | --- |
| canStart(te, detected, world) | このサイクルを起動できるかどうかを返す。 | サイクルが起動可能ならばtrue、不可能ならfalse |
| getNextPhase(te, detected, world) | 現在のフェーズが終了した場合に呼び出され、次のフェーズの番号を返す。フェーズの登録順に0,1,2…と割り振られており、この対応する番号を返す。番号が存在しない場合は内部で0に変換される。これ以上続行できるフェーズが存在しない場合は、-1を返す。 | 次に実行すべきフェーズのインデックス番号（int）、続行すべきものがなければ-1 |
| getInitialPhase(te, detected, world) | このサイクルが起動したとき、始めに実行されるフェーズのインデックス番号を返す。 | 初回実行時のフェーズのインデックス番号（int） |

各引数に関しては以下の通りです。

| 引数名 | 渡されるJavaの型 | 概要 | 備考 |
| --- | --- | --- | --- |
| te | GTSTileEntityTrafficController | このサイクルを実行している制御機のTileEntity |  |
| detected | boolean | 検知信号（押ボタン箱や感知機等）を受信しているかどうか |  |
| world | World | 制御機が設置されているワールドのインスタンス | サーバー側、クライアント側両方来る可能性があるので特定のサイドに依存する内容は避けるべき |

また、bindingという変数を使用することができます。この変数はGTSで用意したいくつかのメソッドや変数等を利用することができる、便利なオブジェクトです。

| メソッド・フィールド名 | 渡されるJavaの型 | 概要 | 備考 |
| --- | --- | --- | --- |
| log(any) | - | Minecraftのログに出力します。デバッグ用途が想定されています。 | 内部的にはGTSのロガーのdebugメソッドを呼び出しているにすぎません。 |
| cycle | GTSCycle | このサイクル自身です。フィールドの変更などを行うことができます。 | 多分privateメソッドは呼べません。 |

### カスタムスクリプトの例

以下は、カスタムスクリプトを用いた最小限の例です。カスタムスクリプトを使うまででもないですが、概ね以下のような形で実装します。関数外に何か処理を書くとそれも実行されますが、実行のタイミングはサイクルが登録された直後なのでご注意ください。

// サイクルは例えば夜間のみ実行可能とする  
function canStart(te, detected, world) {  
 // 現在のtickを取得  
 tick = world.getWorldTime() % 24000; // 世界の累計Tick数が返るので1日で割る  
 return tick > 18000; // 夜間に相当するTickのみ実行可能とする  
}  
  
function getNextPhase(te, detected, world) {  
 // シーケンシャルに次に進めるだけ。最後まで到達した場合は終了。  
 cycle = binding.cycle;  
 if (cycle.getNowPhaseNumber() == cycle.getPhases().size() - 1) {  
 return -1; // 到達した場合は終了  
 }  
 return cycle.getNowPhaseNumber() + 1;  
}  
  
function getInitialPhase(te, detected, world) {  
 // シーケンシャルなので初期は0  
 return 0;  
}

### カスタムスクリプトの指定方法

GTSScriptCycleをtypeに指定し、scriptPathにパスを指定すればOKです。制御機の動作中は絶対に動かさないようにしてください。可能な限り、modsディレクトリの中にあるGTSディレクトリに保存することをお勧めします。 # 交通信号機

最終更新日時：2025年6月4日

追加バージョン：2.0 alpha1

交通信号機（以下、信号機とします）は、以下のクラスにより構成されています。

* GTSBlockTrafficLight → ブロックインスタンス
* GTSTileEntityTrafficLight → 信号機のTileEntity
* GTSTileEntityRendererTrafficLight → 信号機のレンダラー
* GTSTrafficLightConfig → 信号機の個別情報（パック）

## パックJSON（コンフィグ）の仕様

信号機は、共通の項目以外に以下の設定が必要です。

| 設定項目名称 | 概要 | 備考 |
| --- | --- | --- |
| light | この信号機において、点灯するオブジェクトの一覧。文字列の配列で指定します。 | ここを設定することで、フェーズ状態において無点灯のものをしっかり描画できるようになります。 |

### textures

信号機の点灯部分、非点灯部分を別のテクスチャとして描画することにより、よりリアリティの増した描画ができるようにしています。そのため、以下の追加項目の設定が必要です。

| 設定項目名称 | 概要 | 備考 |
| --- | --- | --- |
| light | 信号機の点灯部分に関するテクスチャのパス。 | 全て同一で構わない場合は、baseと同じパスを入れても構いません。 |
| noLight | 信号機の非点灯部分に関するテクスチャのパス。 | 全て同一で構わない場合は、baseと同じパスを入れても構いません。 |

### patterns

交通信号制御機のチャンネルに指定することができるものとして指定する必要があります。ここには、この交通信号機の現示パターンを配列で指定します。配列内には、そのパターンで点灯させるモデルのオブジェクト名を指定することになります。後方互換性のため現在未使用の項目もありますが、設定をしてください。

| 設定項目名称 | 概要 | 備考 |
| --- | --- | --- |
| name | パターン名 | 半角英数字64文字以内推奨。 |
| object | このパターンで点灯させるオブジェクト名を列挙する。 | 文字列の配列で指定します。 |
| ticks | 点滅周期（Tick）。指定した周期で点滅を繰り返すようになります。なお、点灯時間・非点灯時間はそれぞれここで指定したTickの半分の時間となります。 | 0を指定すると常時点灯になります。負の数を指定しないでください。バグります。 |

### 例

{  
 "model": "models/Light\_CommonSteel.mqo",  
 "textures": {  
 "base": "NSS1\_base.png",  
 "light": "NSS1\_light.png",  
 "noLight": "NSS1\_nolight.png"  
 },  
 "body": [  
 "body",  
 "g",  
 "overg",  
 "y",  
 "overy",  
 "r",  
 "overr",  
 "body\_back",  
 "g\_back",  
 "overg\_back",  
 "y\_back",  
 "overy\_back",  
 "r\_back",  
 "overr\_back"  
 ],  
 "patterns": [  
 {  
 "name": "green",  
 "objects": [  
 "g300"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "yellow",  
 "objects": [  
 "y300"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "red",  
 "objects": [  
 "r300"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "yellow\_flush",  
 "objects": [  
 "y300"  
 ],  
 "tick": 20  
 },  
 {  
 "name": "red\_flush",  
 "objects": [  
 "r300"  
 ],  
 "tick": 20  
 },  
 {  
 "name": "green",  
 "objects": [  
 "g300",  
 "g300\_back"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "yellow",  
 "objects": [  
 "y300",  
 "y300\_back"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "red",  
 "objects": [  
 "r300",  
 "r300\_back"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "yellow\_flush",  
 "objects": [  
 "y300",  
 "y300\_back"  
 ],  
 "tick": 20  
 },  
 {  
 "name": "red\_flush",  
 "objects": [  
 "r300",  
 "r300\_back"  
 ],  
 "tick": 20  
 },  
 {  
 "name": "green\_yellow",  
 "objects": [  
 "g300",  
 "y300\_back"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "green\_red",  
 "objects": [  
 "g300",  
 "r300\_back"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "yellow\_green",  
 "objects": [  
 "y300",  
 "g300\_back"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "red\_green",  
 "objects": [  
 "r300",  
 "g300\_back"  
 ],  
 "tick": 0  
 },  
 {  
 "name": "redF\_yellowF",  
 "objects": [  
 "r300",  
 "y300\_back"  
 ],  
 "tick": 20  
 },  
 {  
 "name": "yellowF\_redF",  
 "objects": [  
 "y300",  
 "r300\_back"  
 ],  
 "tick": 20  
 }  
 ],  
 "size": 1.5,  
 "opacity": 0.5,  
 "light": [  
 "g300",  
 "y300",  
 "r300",  
 "g300\_back",  
 "y300\_back",  
 "r300\_back"  
 ],  
 "id": "NSS1\_2H33GYR\_over"

なんかかぶっているものもありますが概ねこのような形で記載します。

## チャンネルについて

交通信号制御機の欄で詳しく解説しますが、信号機はチャンネルを持つことができます。そのチャンネル名をもとに、アタッチされた制御機から信号を受信し、現示を行います。 # 交通信号柱

最終更新日時：2025年6月4日

追加バージョン：2.0 alpha1

交通信号柱（以下、ポールとします）は、以下のクラスにより構成されています。

* GTSBlockTrafficPole → ブロックインスタンス
* GTSTileEntityTrafficPole → ポールのTileEntity
* GTSTileEntityRendererTrafficPole → ポールのレンダラー
* GTSTrafficPoleConfig → ポールの個別情報（パック）

## パックJSON（コンフィグ）の仕様

ポールは、共通の項目以外に以下の設定が必要です。

| 設定項目名称 | 概要 | 備考 |
| --- | --- | --- |
| normalObject | 一般的に設置されるポールのオブジェクトを記載します。 | objectと同一でいいです |
| upJointObject | 同一のポールが上部に存在する（接続されている）場合のオブジェクトを記載します。 | 任意ですが、同じ場合でもnormalObjectと一致するものをコピペしておくと安心です。 |
| bottomJointObject | 同一のポールが下部に存在する（接続されている）場合のオブジェクトを記載します。 | 任意ですが、同じ場合でもnormalObjectと一致するものをコピペしておくと安心です。 |
| fullJointObject | 同一のポールが上下に設置されていて挟まれている場合のオブジェクトを記載します。 | 任意ですが、同じ場合でもnormalObjectと一致するものをコピペしておくと安心です。 |

なお、ポールではobject項目は使用しませんが、後方互換性のためにnormalObjectと同様のものを入れておくことを推奨します。

また、sizeは1.0想定で作成していますので、2.0などを入力すると破綻する可能性があります。少なくともジョイントは反応しません。

### 例

準備中

## モデルの作り方

ポールは、**必ず**原点を中心にして作成してください。ポールも設置時のプレイヤーの向きに影響され方向が変わるので、点対称であるべきです。

各ジョイントオブジェクトに対しても必ず全て原点で作成してください。また、サイズは全て統一してください（1つのオブジェクトだけ極端にでかい、とかを行うとそのオブジェクトに合わせて伸縮するのでカオスなことになります）。

適切に設定していれば、ポールを設置した際にモデルが切り替わり、より見た目がきれいに見えることでしょう。 # アーム

最終更新日時：2025年6月4日

追加バージョン：2.0 alpha2

アームは、以下のクラスにより構成されています。

* **GTSItemTrafficArm** → アームのアイテムインスタンス
* GTSTileEntityTrafficArm → アームのTileEntity
* GTSTrafficArmConfig → アームの個別情報（パック）

なお、他のクラスとの相違点として、以下があります。

* ブロックとして存在しません。アームは、ポールの一種となります。そのため、ブロックインスタンスではなく**アイテムインスタンス**となります。
* アームはポールと一緒に描画されるため、TileEntitySpecialRendererは存在しません。
* ブロックとして存在しない以上TileEntityの実態もないため、GTSTileEntityTrafficArmはダミーのTileEntityとなっています。すなわち、Minecraftに登録されているTileEntityではありません。

他と違いクラスが分かりにくいと思いますが、上記にしたがって閲覧いただければと思います。

## パックJSON（コンフィグ）の仕様

アームは、共通の項目以外に以下の設定が必要です。

| 設定項目名称 | 概要 | 備考 |
| --- | --- | --- |
| edgeObjects | ポール始点から描画されるアームのオブジェクトを記載します。 |  |
| baseObjects | 中間地点で描画されるアームのオブジェクトを記載します。 | 後述しますがここは必須で、割と重要なポイントです。 |
| endObjects | ゴールとなるポイントで描画されるアームのオブジェクトを記載します。 | たいていの場合はベースと同じかと思われます |
| drawStartPrimary | true / falseで指定します。trueにすると、長さが足りない場合でもedgeObjectsの描画を優先します。falseにすると、足りない場合に描画がされなくなります。 | trueを通常は指定してください。falseは特殊な用途でのみ使用します。 |

なお、アームではobject項目は使用しませんが、後方互換性のためにnormalObjectと同様のものを入れておくことを推奨します。

drawStartPrimaryに関しては、後方互換性のために入れているものとなります。通常はtrueを指定してください。コードを読んで何をやっているのか理解できる方はfalseに変更しても構いません。

### 例

{  
 "id": "Arm\_Normal2\_Ped",  
 "model": "models/Arm\_Ped2Normal.mqo",  
 "textures": {  
 "base": "arm.png"  
 },  
 "edgeObjects": ["start"],  
 "baseObjects": ["base"],  
 "endObjects": ["end"],  
 "size": 1,  
 "drawStartPrimary": false  
}

## モデルの作り方

アームはその性質上、モデルの作り方に若干の癖があります。これを守ることでよりスムーズに描画がされるので、できるだけ従ってください。コードを読んで描画内容を理解できる方は色々カスタマイズされても構いません。

### 描画の仕組み

アームの描画は、以下のようにして行われます。

1. ポールの開始地点ブロックの中心を原点とし、接続先の方向にsizeで指定した分だけの長さでedgeObjectsを描画します。
2. edgeObjectsの描画終了地点から、接続先地点のブロックからsizeで指定した分の長さを引いた長さにbaseObjectsを**スケールして**描画します。
3. 最後に、接続先地点のブロックからsizeで指定した分の長さを引いた地点から接続先地点までを、endObjectsによって描画します。

特に重要なのが2で、baseObjectsはスケーリングされて描画されます。そのため、しっかり意識して作成しないと変な伸び方をしたりします。

### baseObjectsの制約

baseObjectsは、**原点を中心としてX軸方向にスケーリングされます**。したがって、アームの作成時は必ずモデルを原点に配置し（とくにX座標）、X軸方向にアームを伸ばしてください。この時のサイズはsizeになりますのでそれも意識してください。

### その他の制約

ポールと同様に、全て原点を中心に作成してください。# 押ボタン箱

最終更新日時：2025年6月4日

追加バージョン：2.0 alpha2

押ボタン箱は、以下のクラスで構成されています。

* GTSBlockTrafficButton → ブロックインスタンス
* GTSTileEntityTrafficButton → 押ボタン箱のTileEntity
* GTSTileEntityRendererTrafficButton → 押ボタン箱のレンダラー
* GTSTrafficButtonConfig → 押ボタン箱の個別情報（パック）

## パックJSON（コンフィグ）の仕様

### 音声ファイルについて

押ボタン箱は、audiosという追加オブジェクトを保有します。ここには、各対応する音声ファイルのパスを挿入します。

現時点では、以下の2つが存在します。

* base　…　一定間隔で常時鳴り続ける音声。ただし、設定で鳴らさないようにすることもできます。音響用押ボタン箱の「ピピッ」みたいな音を想定しています。
* detected　…　押ボタン箱が押された際に1度だけ鳴る音声。

今後追加予定のものもあるため、その場合はそちらも順次実装後に掲載します。

### テクスチャについて

押ボタン箱のtexturesは、baseの他にdetectedというキーでテクスチャを指定します。このテクスチャは押ボタン箱が押されている状態のときに使用されます。例えば、「おしてください」と「おまちください」を分けることができます。

### 例

準備中

## 押ボタンの挙動について

押ボタンはアタッチされている制御機に対して検知信号を送信する機器の役割を果たします。一度押して制御機の検知信号がオン（true）になったら、制御機が検知信号を解除するか、（設定していれば）最大待ち時間Tickまで待機しない限り押ボタンを押せなくなりますのでご注意ください。この場合、撤去して再度設置しなおせばまた押せるようになります。# 地名板

最終更新日時：2025年6月9日

追加バージョン：2.0 alpha3

## 注意

2.0 alpha3の時点では大変不安定となっており、テクスチャが正しく反映されなかったり、Minecraftがクラッシュする可能性があります。使用する場合はワールドのバックアップを強くお勧めします。

地名板は、以下のクラスで構成されています。

* GTSBlockTrafficSign → ブロックインスタンス
* GTSTileEntityTrafficSign → 地名板のTileEntity
* GTSTileEntityRendererTrafficSign → 地名板のレンダラー
* GTSTrafficSignConfig → ※ダミーの設定ファイルクラス

## パックJSON（コンフィグ）の仕様

地名板はコンフィグファイル不要です。現時点では自動生成されます。将来的にはテクスチャをもとに自由に選べるようにする予定ですが、まだ仕様がきちんと策定されていないのでどうなるか未定です。

## 自動生成される地名板について

使用方法はユーザーマニュアルに記載しているのでここでは内部仕様について記します（不安定な原因も）。

地名板は、Minecraftの起動時に「生成中」と表示されたテクスチャがダミーで作成されます。その後、TileEntityから内容を正しく読み込めれば正常に反映されます。

テクスチャは内部のジェネレーターで自動生成しており、国交省のレイアウトに準じています。このテクスチャ生成処理が結構重く、開発環境でも0.1秒近くかかりその間Minecraftが応答なしになります。

そのため、テクスチャ生成処理を並列処理化して、極力Minecraftを殺さないように心がけています。しかし、この代償としてわずかなタイミングでテクスチャがnullになる場合があります。なるべくエラーチェックを行うように心がけていますが、漏れがある場合ここで描画に失敗しMinecraftがクラッシュする可能性があります。次回起動時には正しいものが反映されているかロールバックされているかでワールドの破損は起こりませんが、もしこのような場合はお手数ですがご連絡いただけると助かります。

なお、2.0 alpha2までのバージョンは、こちら側のコーディングミスにより高確率で落ちます。2.0 alpha3ではこのミスは修正しているので、特に理由のない限りは最新版をご利用ください。# パックのサポートと権利について

## パックの権利

GTSで使用できるパックについては、**原則パックの作成者が**著作権を持ちます。これは、裏を返せば**そのパックが当社製のものでない限り、当社は一切の責任を負わない**ことを指します。GTSのコア部分に関してバグが発生した場合はサポートを行いますが、パックを導入したことでワールドの破損があった場合などは、パックの作成者にお問い合わせください。

なお、パックの作成者自身が「作り方がわからない」等といった相談を当社にされるのは問題ありません。

## パックの使い道

上述の通り、パックは作成者が全ての権利を保有するため、作成者の意向に従ってください。作成者は、そのパックを他ユーザーに向けて配布する場合は、その利用方法などについて周知を心がけてください。何度も繰り返しになりますが、拡張パックの内容そのものに関するお問い合わせは当社は一切受け付けません。

## パックの中身の権利問題

パックの中身に著作権や商標権、肖像権などに関わるような内容などは入れないでください。それらを使用し、また利用して発生したいかなる責任は、パックの作成者に帰属します。

# パックの宣伝をご希望の方へ

当社では、申請のあったパックを公式サイトやYouTubeなどで宣伝・紹介することがあります。その際は、当社までご連絡ください。中身を確認し、問題ないことを確認したうえで宣伝などを行うか判断します。

# その他

この仕様書は、頻繁に更新される可能性があります。更新の際、再度自動生成プログラムをかけ自動的に更新されたバージョンを公開しますが、細かい更新履歴はGitHub内のコミット差分をご覧ください。メンテの問題からこちらに更新履歴の記載は避けます。

© 2025- G-Factory. All rights reserved.