

# 譜面ファイルの仕様

このドキュメントは、holorhysmの譜面ファイルの仕様をまとめたものです。

## 例

```
/** @type {Object} - 全体 */
{
  /** @type {Number|String} - BPMを表す文字列 */
  "bpm": 141,
  /** @type {String} - 拍子を表す文字列 */
  "beats": "x < 68 ? [4,4] : [2,4]",
  /** @type {String} - ソフラン倍率を表す文字列 */
  "soflan": "x < 68 ? 1.00 : 1.1",
  /** @type {Number} - 1小節目の開始位置(秒) */
  "offset": 0.96,
  /** @type {Object[]} - ノーツ一覧(順不同) */
  "notes": [
    /** @type {Object} - ノーツ */
    {
      /** @type {String} - ノーツの種類。push, hover, isolateのいずれか */
      "type": "push",
      /** @type {Number[]} - ノーツの端の位置、-3〜3。[左端, 右端] */
      "where": [ 0.00, 2.00],
      /** @type {Number[]} - ノーツのタイミング。[小節, 分子, 分母] */
      "when": [ 2, 0, 4],
      /** @type {Number} - ノーツの単体ソフラン倍率。 */
      "speed": 1.00,
    },
    // (以下省略)
  ],
  /** @type {Object[]} - デコレーター一覧(順不同) */
  "decorator": [
    /** @type {Object} - デコレーター */
    {
      /** @type {String|Array} - ノーツの色。グラデの場合はArrayで指定 */
      "color": "rgb(255 127 127 / .5)",
      /** @type {Object} - 開始位置 */
      "start": {
        /** @type {Number[]} - 端の位置、-3〜3。[左端, 右端] */
        "where": [-3.00, 3.00],
        /** @type {Number[]} - 開始位置のタイミング。[小節, 分子, 分母] */
        "when": [ 2, 0, 4],
      },
      /** @type {Object} - 終了位置 */
      "end": {
        /** @type {Number[]} - 端の位置、-3〜3。[左端, 右端] */
        "where": [ 0.00, 0.00],
        /** @type {Number[]} - 開始位置のタイミング。[小節, 分子, 分母] */
        "when": [ 2, 2, 4],
      },
      /** @type {Object} - 接続イージング */
      "easing": {
        /** @type {String} - 左側の接続イージング */
        "left": "Cubic_Out",
        /** @type {String} - 右側の接続イージング */
        "right": "Linear_InOut",
      },
      /** @type {Number} - デコレーターの単体ソフラン倍率 */
    }
  ]
}
```

```

        "speed": 1.00,
    },
    // (以下省略)
],
/** @type {Object[]} - カスタム関数一覧(順不同) */
"function": [
    {
        "when": [ 2, 0, 4],
        "main": "console.log(_.DB)",
        "worker": "console.log($.chart.notes)",
    },
    // (以下省略)
],
}

```

## 各項目の解説

### chart.bpm

```

// BPM変化が一切ない場合、Stringで数字を記述します。
"bpm": "141",
// 一応Numberでも動きますが非推奨です。
"bpm": 141,
// BPM変化がある場合、JavaScriptの式を記述します。条件(三項)演算子を使うのが比較的簡単です。
"bpm": "x < 30 ? 121 : 242",

```

- 必須のプロパティです。
- 楽曲のBPMを表す、**JavaScriptの式(expression)として有効な文字列**です。
  - 式の中で、**xは小節番号を表します**。
    - 実装: `Function("x", `return ${"ここに入れる文字列を指定"}`)(小節番号)`
- 許容される型は `{String|Number}` です。
  - 2024-06-12 Update : *Numberでも動くことに気づきました。非推奨としたうえで記載しておきます。*

### chart.beats

```

// 拍子変化がない場合、Stringで配列を記述します。
"beats": "[4, 4]",
// 拍子変化がある場合、JavaScriptの式を記述します。条件(三項)演算子を使うのが比較的簡単です。
"beats": "x < 68 ? [4, 4] : [2, 4]",

```

- 必須のプロパティです。
- 楽曲の拍子を表す、**JavaScriptの式(expression)として有効な文字列**です。
  - 式の中で、**xは小節番号を表します**。
    - 実装: `Function("x", `return ${"ここに入れる文字列を指定"}`)(小節番号)`
- 式の**返り値は2つの数字を持つ配列**である必要があります。
  - 配列の0番目が分子、1番目が分母を示します。
    - 例1: 4分の4拍子 → `[4, 4]`
    - 例2: 8分の7拍子 → `[7, 8]`
  - 配列の1番目を `0` にするとバグの原因になります。
- 許容される型は `{String}` です。

### chart.soflan

```
// ソフランがない場合、"1"を記述します
"soflan": "1",
// 一応Numberでも動きますが非推奨です。
"bpm": 1,
// ソフランがある場合、JavaScriptの式を記述します。条件(三項)演算子を使うのが比較的簡単です。
"soflan": "x < 68 ? 1.00 : 1.1",
```

- 必須のプロパティです。
- 譜面のスクロール速度を表す、**JavaScriptの式(expression)**として有効な文字列です。
  - 式の中で、**xは小節番号を表します。**
    - 実装: `Function("x", `return ${"ここに入れる文字列を指定"}`)(小節番号)`
- 許容される型は `{String|Number}` です。
  - 2024-06-13 Update : `Number`でも動くことに気づきました。非推奨としたうえで記載しておきます。

## chart.offset

```
// 1小節目の開始位置をNumberで記述します。単位は「秒」です。
"offset": 0.96,
```

- 必須のプロパティです。
- 1小節目の開始位置を表す数字です。単位は「秒」です。
- 1小節目の開始位置が0未満であってははいけません。
  - 一応動くとは思いますが、他の仕様との兼ね合いでバグが発生しかねないので避けてください。
- 許容される型は `{Number}` です。

## chart.notes

```
// ノーツをすべて格納した配列です。
"notes": [
  {
    "type": "push",
    "where": [ 0.00, 2.00],
    "when": [ 2, 0, 4],
    "speed": 1.00,
  },
  // (以下省略……)
],
```

- 必須のプロパティです。
- 譜面のノーツを表すオブジェクトをすべて格納した配列です。
- 少なくとも1つ以上のオブジェクトが含まれている必要があります。
  - ノーツがないとスコア計算でゼロ除算エラーが発生します。
- 許容される型は `{Object[]}` です。

### chart.notes[].type

```
// ノーツの種類を指定します。push, hover, isolateのいずれかを指定します。
"type": "push",
```

- 必須のプロパティです。

- ノーツの種類を表す文字列です。
- 以下のうちいずれかの文字列を指定する必要があります。
  - "push" : 赤色のノーツです。タップに反応します。
  - "hover" : 緑色のノーツです。押している間に反応します。
  - "isolate" : 青色のノーツです。離している間、もしくは擦り動作をしたときに反応します。
- 許容される型は `{String}` です。

## chart.notes[].where

```
// ノーツの左端・右端の位置を指定します。
"where": [ 0.00, 2.00],
```

- **必須**のプロパティです。
- ノーツの左端・右端の位置を数値2つの配列で指定します。
  - 配列の0番目で左端、1番目で右端を指定します。
- レーンの境目が整数で、一番左のレーンの左端が -3、一番右のレーンの右端が 3 を表します。
- 許容される型は `{Number[]}` です。

## chart.notes[].when

```
// ノーツの位置を3つの数字で指定します。
"when": [ 2, 0, 4],
// ほとんどの譜面で最大桁数が[3桁, 2桁, 2桁]になるので、スペースの数を合わせると見やすいです。
"when": [123, 3, 16],
```

- **必須**のプロパティです。
- ノーツを押すべきタイミングを3つの数字の配列で指定します。
  - 配列の0番目の値には、その**ノーツが置かれる小節番号**を指定します。
    - **自然数**である必要があります。
  - 配列の1番目の値には、その**ノーツが小節の先頭から何拍目に置かれるか**を指定します。
    - **0以上の数**である必要があります。
      - **非負整数**にすることを推奨します。
  - 配列の2番目の値には、**1番目の拍を何分音符でカウントするか**を指定します。
    - **自然数**である必要があります。
  - 例 : [2, 3, 8] → 2小節目の先頭から8分音符3つ分後ろのタイミング
- 許容される型は `{Number[]}` です。

## chart.notes[].speed

```
// ノーツ単体のソフラン倍率を指定します。
"speed": 1.00,
```

- 任意のプロパティです。
  - 指定されていない場合のデフォルト値は 1 です。
- ノーツの流れてくる速度(倍率)を指定します。
  - **0でない数**である必要があります。
    - 0でも動作はしますが、**負荷の原因**になるので**原則禁止**とします。
- 許容される型は `{Number}` です。

## chart.decorator

```
// デコレーターをまとめた配列です。
"decorator": [
  {
    "color": "rgb(255 127 127 / .5)",
    "start": {
      "where": [-3.00, 3.00],
      "when": [ 2, 0, 4],
    },
    "end": {
      "where": [ 0.00, 0.00],
      "when": [ 2, 2, 4],
    },
    "easing": {
      "left": "Cubic_Out",
      "right": "Linear_InOut"
    },
    "speed": 1.00,
  },
  // (以下省略)
],
```

- 必須のプロパティです。
- 譜面のデコレーターを表すオブジェクトをすべて格納した配列です。
- 必ずしもオブジェクトが含まれている必要はありません。
  - デコレーターがない譜面をつくる場合は空配列を指定してください。
- 許容される型は `{Object[]}` です。

## chart.decorator[].color

```
// 単色のデコレーターの場合、デコレーターの色をCSSの<color>型として解釈できる文字列で指定します。
"color": "rgb(255 127 127 / .5)",
// グラデーションをする場合、CanvasGradient.prototype.addColorStop( )の引数になるような2次元配列を指
定します。
"color": [
  [0, "rgb(from white r g b / 1.0)"],
  [1, "rgb(from white r g b / 0.0)"],
],
```

- 必須のプロパティです。
- デコレーターの色を表す値を指定します。
- 許容される型は `{String!(Number!String)[[]]}` です。

## 単色デコレーターの場合

単色デコレーターの場合は、原則としてCSSの<color>型として解釈できる文字列を指定します。  
以下に、CSSの<color>型の指定方法を列挙します。

TL;DR.

- [このページ](#)にある一覧表にあるキーワードが使えます
- `"#FF0000"` みたいなやつも使えます
- [このページ](#)の `"oklch(27.1% 0.062 281.44)"` みたいなやつも使えます
- 疑似ロングノーツには `"pseudolong"` を指定するといいでしょう




```
// 単色のデコレーターの場合、デコレーターの色をCSSの<color>型として解釈できる文字列で指定します。  
"color": "rgb(255 127 127 / .5)",
```

## ver0.5.0 暫定仕様

ver0.5.0ではSafari 17.6が一部の構文に対応していないことに起因するバグを回避するために、color文字列の解析を自前で実装しています。

この関係で、一部の記法が使用不可となっています。

現在使える色指定の方法は、後述するもののうち、以下の方法のみとなります。

- `<named-color>`
- `<hex-color>`
- `<color-function>` のうち、`rgb()`、`hsl()`、`oklab()`、`oklch()`
  - 相対色構文には対応していますが、`from`のあとに続くcolorも同様にこの一覧にあるものしか対応していません
  - `rgb()` のLegacy構文(カンマ区切り)は対応していません
- キーワード `pseudolong`
  - `<named-color>` が使える場所ならどこでも使えます
- `<named-color>` 型のキーワード
  - `"red"`、`"aqua"`、`"violet"`、`"rebeccapurple"` などが挙げられます。
  - 完全な一覧は[MDN Web Docs \(ja-JP\)『<named-color>』](#)を参照してください。
  - 完全な透明を表す `"transparent"` を指定することもできます。
  - `"currentcolor"` も`<named-color>`型らしいですが、これは使わないでください。
    - 表示色がコード依存になるので……
- `<hex-color>` 型 16進数カラーコード
  - `"#891546"` (  ), `"#1154"` (  ) などが挙げられます。
    - `"#RRGGBB"`、`"#RRGGBBAA"`、`"#RGB"`、`"#RGBA"` の4パターンがあります。
  - 詳細な仕様は[MDN Web Docs \(ja-JP\)『<hex-color>』](#)を参照してください。
- `<color-function>` 型 色空間関数
  - `"rgb(128 128 128 / 0.5)"` (  ) などが挙げられます。
  - MDN Web Docs (ja-JP)に各関数の仕様が掲載されています。
    - [rgb\(\)](#)、[hwb\(\)](#)、[hsl\(\)](#)、[lab\(\)](#)、[lch\(\)](#)、[oklab\(\)](#)、[oklch\(\)](#)、[color\(\)](#)
    - とくに `oklch()` はグラデを自然にできたりするのでおすすめです。
      - [カラーピッカーツール](#) もあります。
      - 拙作に[p5.jsによる色空間変換ツール](#) もあります。
- 色空間関数 relative color syntax
  - [Chrome for Developers](#) の紹介記事が参考になります。
  - それぞれの色空間関数の最初の引数に `from <color>` を追加して使います。
  - 「基準の色に対してこのくらい変化させる」といったときに使えます。

また、特別なキーワードとして `"pseudolong"` を使用することができます。

- holorhysmのシステムは、`"pseudolong"` を「hoverノーツの色の透明度を50%にした色」として解釈します。
  - ノーツのスキン変更に伴ってhoverノーツの色が変わっても、hoverノーツにあった色になります。
  - デコレーターとhoverノーツを使用した疑似ロングノーツに使う想定です。
  - 相対色構文の基準色に指定することもできます。
    - 例: `rgb(from pseudolong r g b / .5)`
    - ~~できるようにします~~

## グラデーションをかける場合

デコレーターに(下から上に向けて)グラデーションをかけることもできます。

グラデーションをかける場合は、`chart.decorator[].color` に [CanvasGradient.prototype.addColorStop\(\)](#) の引数を2次元配列で指定します。

```
// グラデーションをする場合、CanvasGradient.prototype.addColorStop() の引数になるような2次元配列を指定します。
```

```
"color": [
  [0, "rgb(from white r g b / 1.0)"],
  [1, "rgb(from white r g b / 0.0)"],
],
```

- `color` は配列です。
- `color` の各要素 `colorStop` は配列です。
- `colorStop` の0番目の要素は `offset` で、経路点がどの位置にあるかを指定します。
  - **0以上1以下のNumber型の値**を指定します。
  - デコレーターの開始地点が0、終了地点が1です。
- `colorStop` の1番目の要素は `color` で、経路点における色を指定します。
  - **CSSの<color>型として解釈できる文字列**を指定します。
  - 上の『[単色デコレーターの場合](#)』を参照してください。

例

(←開始位置)  (終了位置→)

```
"color": [
  [0.0, "oklch(62.8% 0.15 0 / 1)"],
  [0.2, "oklch(62.8% 0.15 120 / 1)"],
  [0.4, "oklch(62.8% 0.15 240 / 1)"],
  [0.6, "oklch(62.8% 0.15 360 / 1)"],
  [0.6, "rgb(128 128 128 / 1)"],
  [1.0, "rgb(128 128 128 / 0)"],
],
```

#### ①2色でグラデ

```
"color": [
  [0.0, "red"], // "この段階ではこの色にする"を指定することでグラデをつくる
  [1.0, "green"], // 0が先頭, 1が終端
],
```

↓

(←開始位置)  (終了位置→)

#### ②3色でグラデ

```
"color": [
  [0.0, "red"],
  [0.5, "yellow"], // 中間点は0.5
  [1.0, "blue"],
],
```

↓

(←開始位置)  (終了位置→)

#### ③途中で急に色を変える

```

"color": [
  [0.0, "red"],
  [0.5, "yellow"],
  [0.5, "red"], // 指定する位置を限界まで近づけると一気に変わったように見せる……みたいな
  [1.0, "blue"],
],

```

↓

(←開始位置)  (終了位置→)

## chart.decorator[].(start|end)

```

// デコレーターの開始位置を指定します。
"start": {
  "where": [-3.00, 3.00],
  "when": [ 2, 0, 4],
},
// デコレーターの終了位置を指定します。
"end": {
  "where": [ 0.00, 0.00],
  "when": [ 2, 2, 4],
},

```

- どちらも**必須**のプロパティです。
- デコレーターの開始位置・終了位置についての情報をまとめたオブジェクトです。
- 許容される型はどちらも `{Object}` です。

## chart.decorator[].(start|end).where

```

// 開始位置・終了位置の左右方向の座標(どこに降ってくるか)を指定します。
"where": [-3.00, 3.00],

```

- **必須**のプロパティです。
- デコレーターの(開始位置|終了位置)の左端・右端の位置を数値2つの配列で指定します。
  - 配列の0番目で左端、1番目で右端を指定します。
- レーンの境目が整数で、一番左のレーンの左端が -3、一番右のレーンの右端が 3 を表します。
- 許容される型は `{Number[]}` です。

## chart.decorator[].(start|end).when

```

// 開始位置・終了位置がいつ判定線のところまで降ってくるかを指定します。
"when": [ 0.00, 0.00],

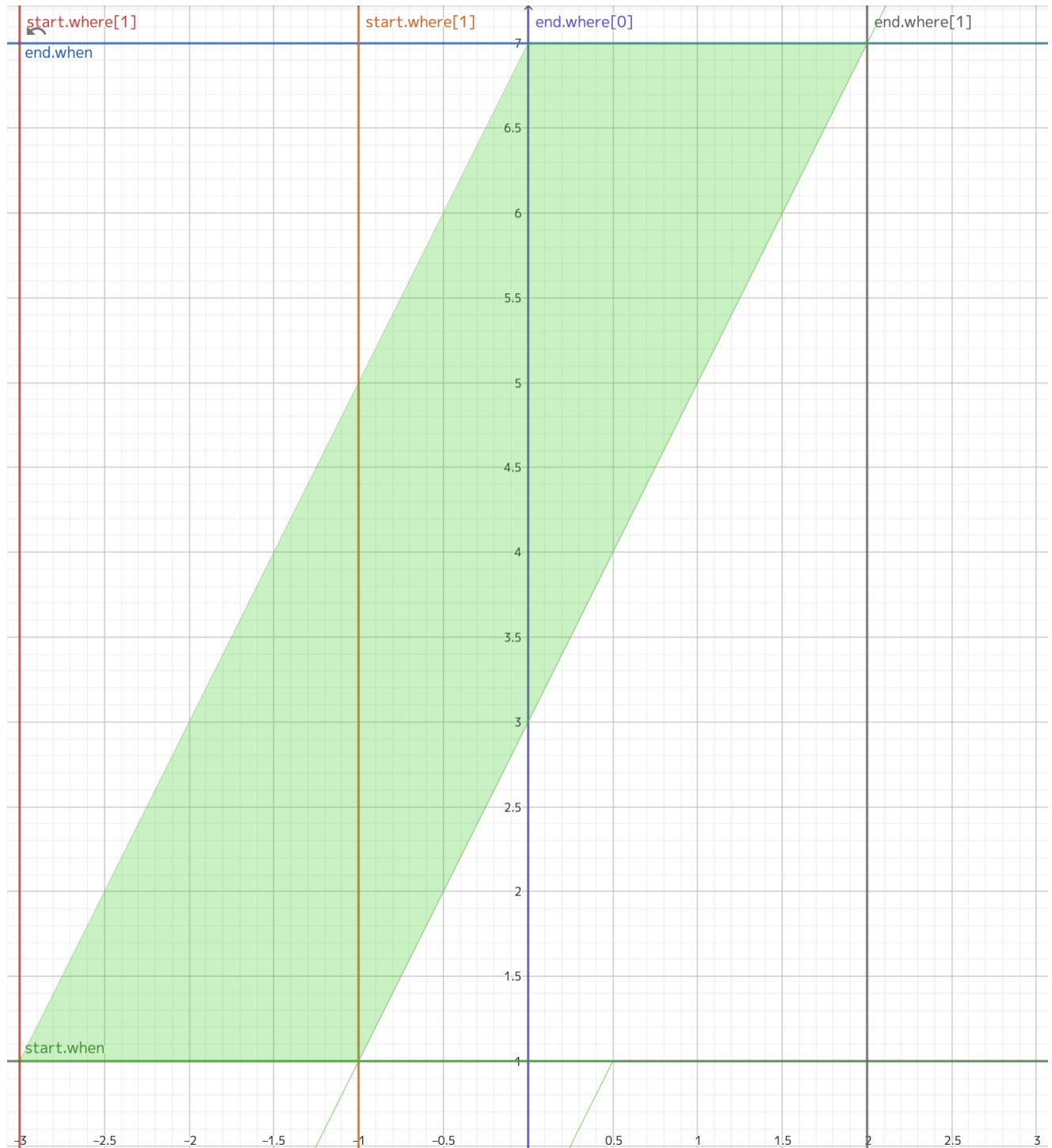
```

- **必須**のプロパティです。
- デコレーターの(開始位置|終了位置)のタイミングを3つの数字の配列で指定します。
  - 配列の0番目の値には、その**ノーツが置かれる小節番号**を指定します。
    - **自然数**である必要があります。
  - 配列の1番目の値には、その**ノーツが小節の先頭から何拍目に置かれるか**を指定します。
    - **0以上の数**である必要があります。
      - **非負整数**にすることを推奨します。



- 配列の2番目の値には、**1番目の拍を何分音符でカウントするか**を指定します。
  - **自然数**である必要があります。
  - 例：[2, 3, 8] → 2小節目の先頭から8分音符3つ分後ろのタイミング
- 許容される型は {Number[]} です。

## 参考画像



## chart.decorator[].easing

// 右側・左側それぞれの接続にかけるイーザングを指定します。

```
"easing": {
  "left": "Cubic_Out",
```

```
    "right": "Linear_InOut",
  },
```

- **必須**のプロパティです。
- デコレーターの左右それぞれの接続にかけるイージングを指定するオブジェクトです。
- 許容される型は `{Object}` です。

## `chart.decorator[].easing.(left|right)`

```
// 右側・左側それぞれの接続にかけるイージングを指定します。
"left": "Cubic_Out",
"right": "Linear_InOut",
```

- どちらも**必須**のプロパティです。
- デコレーターの(左|右)側の2点の接続にかけるイージングを指定します。
- 許容される型は `{String}` です。

## `chart.decorator[].easing.(left|right)`に指定する文字列のルール

`chart.decorator[].easing.left`、`chart.decorator[].easing.right` には、以下のEBNF(拡張バックス・ナウア記法)で示される `<holorhysm-decorator-easing>` を満たす文字列を指定する必要があります。

```
<holorhysm-decorator-easing> = <function>, "_", <type>
<function>  = "Linear" | "Sine" | "Quad" | "Cubic" | "Quart" | "Expo" | "Circ"
<type>      = "In" | "Out" | "InOut"
```

- `<function>` は、線をどのような関数で描画するか指定します。
  - `Linear` : 直線
  - `Sine` : 正弦関数(サインカーブ)
  - `Quad` : `Quad` : 2次関数(放物線)
  - `Cubic` : 3次関数
  - `Quart` : 4次関数
  - `Expo` : 指数関数
  - `Circ` : 円周関数(円)
- `<type>` は加減速を指定します。
  - `In` : はじめは緩やかな変化、おわりにかけて変化が大きくなる(加速)
  - `Out` : はじめは大きな変化、おわりにかけて変化が緩やかになる(減速)
  - `InOut` : はじめとおわりは緩やかな変化、中盤に変化が大きくなる(加減速)
- `"Linear_In"` ・ `"Linear_Out"` ・ `"Linear_InOut"` はどれも直線を指定する `<holorhysm-decorator-easing>` の文字列です。

## `chart.decorator[].speed`

```
// デコレーター単体のソフラン倍率を指定します
"speed": 1.00,
```

- 任意のプロパティです。
  - 指定されていない場合のデフォルト値は `1` です。
- デコレーターの流れてくる速度(倍率)を指定します。
  - **0でない数**である必要があります。

- 0でも動作はしますが、負荷の原因になるので原則禁止とします。
- 許容される型は `{Number}` です。

## chart.function

```
// セッション中に実行するカスタム関数をまとめた配列です。
"function": [
  {
    "when": [ 2, 0, 4],
    "main": "console.log(_.DB)",
    "worker": "console.log($.chart.notes)",
  },
  // (以下省略)
],
```

- **必須**のプロパティです。
- セッション中の特定タイミングで実行する「カスタム関数」を表すオブジェクトをすべて格納した配列です。
- 必ずしもオブジェクトが含まれている必要はありません。
  - カスタム関数がない譜面をつくる場合は空配列を指定してください。
- 許容される型は `{Object[]}` です。

## chart.function[].when

```
// カスタム関数を実行するタイミングを指定します。
"when": [ 2, 0, 4],
```

- **必須**のプロパティです。
- カスタム関数を実行するタイミングを3つの数字の配列で指定します。
  - 配列の0番目の値には、その**ノーツが置かれる小節番号**を指定します。
    - **自然数**である必要があります。
  - 配列の1番目の値には、その**ノーツが小節の先頭から何拍目に置かれるか**を指定します。
    - **0以上の数**である必要があります。
      - **非負整数**にすることを推奨します。
  - 配列の2番目の値には、**1番目の拍を何分音符でカウントするか**を指定します。
    - **自然数**である必要があります。
  - 例：[2, 3, 8] → 2小節目の先頭から8分音符3つ分後ろのタイミング
- 許容される型は `{Number[]}` です。

## chart.function[].main

```
// メインスレッドで実行する関数の内容を記述します。
"main": "console.log(_.DB)",
```

- **必須**のプロパティです。
- **メインスレッドで実行する関数のコード**を、JavaScriptの(関数の中身の)コードとして有効なStringで指定します。
  - 技術的補足： `Function("_", ここに入れる文字列を指定)(後述する'_');`
- 許容される型は `{String}` です。

## \_ (on custom function "main")

`chart.function[]`.`main` で実行する関数には、引数として `_` という名前のオブジェクトが渡されます。以下は、`_` の中身の説明です。

- `_`
  - `.audioUnit` - 楽曲を再生しているAudioUnit (実装は `/scripts/modules/audioUnit.js` を参照)
  - `.canvas` - 譜面描画先のHTMLCanvasElement
  - `.DB` - 譜面DBファイルのうち、現在プレイ中の楽曲/譜面の部分の情報を持つObject
    - `.music` - 譜面DBファイルのうち、現在プレイ中の楽曲の部分の情報を持つObject
    - `.chart` - 譜面DBファイルのうち、現在プレイ中の譜面の部分の情報を持つObject
  - `.chart` - 基本的にはプレイ中の譜面ファイルのObjectだが、追加で以下のプロパティを持つ
    - `.barStartAtPx[]` - ( {Number[]} )i小節目が1小節目先頭から何pxスクロールした位置からはじまるか
    - `.barStartAtSec[]` - ( {Number[]} )i小節目が1小節目先頭から何秒経過したタイミングからはじまるか
  - `.worker` - 描画・計算処理を実行してくれているWorker
  - `.containerDiv` - 現在のセッション画面のHTMLDivElement

技術的補足：`_` の中身はシャローコピーになっているはずなので、代入すれば下の方にも反映されます

## chart.function[].worker

```
// ワークスレッドで実行する関数の内容を記述します。  
"worker": "console.log($.chart.notes)",
```

- 必須のプロパティです。
- ワークスレッドで実行する関数のコードを、JavaScriptの(関数の中身の)コードとして有効なStringで指定します。
  - 技術的補足：`Function("_", ここに入れる文字列を指定)(後述する'_');`
- 許容される型は `{String}` です。

## \$ (on custom function "worker")

`chart.function[]`.`worker` で実行する関数には、引数として `_` という名前のオブジェクトが渡されます。以下は、`_` の中身の説明です。

- `$`
  - `.chart` - 譜面ファイルのObjectですが、以下のような相違点があります
    - `.barStartAtPx[]`, `.barStartAtSec[]` を持つ(main側と同様)
    - `.beats`, `.bpm`, `.soflan` がFunctionになっている(引数は小節番号)
    - `.notes[]` に判定済みのノーツはない (`.decorator[]`, `.function[]` も同様)
  - `.canvas` - 次のフレームでの状態を描画するためのOffscreenCanvas。
  - `.nowPlaying_sec` - 今楽曲の何秒の位置を再生している(と推測される)か
  - `.LocalStorageEX_Ref` - Worker起動時点のLocalStorageEX
    - holorrhysmではLocalStorageにObjectを入れるためにLocalStorageEXでラップしています
  - `.tryJudgingNote(note, time)` - ノーツが指定時刻に叩かれたものとして判定処理を試みる
    - `note` は `$.chart.notes[]` にあるもの、`time` は `audioUnit.currentTime` (秒)基準
  - `.judgeWidth` - 判定幅(±0ms)
    - `.judgeWidth["sync+"]`, `.sync`, `.connect`, `.jamming` があります