

# Save Design スクリプトリファレンス

## 目次

1. [インターフェース](#)
    - 1.1 [IAfterInitializeCallback](#)
    - 1.2 [IAfterInitializeRollback](#)
    - 1.3 [IAfterLoadCallback](#)
    - 1.4 [IAfterLoadRollback](#)
    - 1.5 [IBeforeSaveCallback](#)
    - 1.6 [IBeforeSaveRollback](#)
    - 1.7 [ISaveDesignConfig](#)
  2. [属性](#)
    - 2.1 [EncryptorAttribute](#)
    - 2.2 [SaveDesignRootAttribute](#)
    - 2.3 [SharedDataAttribute](#)
    - 2.4 [SlotDataAttribute](#)
    - 2.5 [SlotMetaDataAttribute](#)
    - 2.6 [TempDataAttribute](#)
  3. [列挙型](#)
    - 3.1 [ExceptionPolicy](#)
    - 3.2 [SerializerType](#)
    - 3.3 [TempDataResetTiming](#)
  4. [クラス](#)
    - 4.1 [SaveDesignRoot](#) 属性を付与されたクラス
  5. [サードパーティライセンス](#)
-

# 1. インターフェース

## 1.1 IAfterInitializeCallback

説明

データの初期化時に一度だけ何らかの処理を実行したい場合はこのインターフェースを使用します。

呼び出されるケース

- データを初期化したとき
- データを読み込んだとき、ゲームバージョンの違いでセーブファイルに対象のデータが存在しなかったとき

呼び出されないケース

- データを読み込んだとき、既存セーブデータにそのデータが含まれており、正常に復元された場合

このインターフェースは `SharedData` 属性、`SlotData` 属性のいずれかを付与したクラスに実装する必要があり、`SlotMetaData` 属性、`TempData` 属性のみを付与したクラスやどのデータ属性も付与していないクラスに実装した場合は無視されます。

Public 関数

| 関数名                               | 説明               |
|-----------------------------------|------------------|
| <a href="#">OnAfterInitialize</a> | データ初期化時に呼び出されます。 |

OnAfterInitialize

- `public void OnAfterInitialize ();`

説明

データの初期化時に一度だけ呼び出されます。

```
using System;
using SaveDesign.Runtime;

[SharedData, Serializable]
public class ExampleClass : IAfterInitializeCallback
{
    public int money;

    void IAfterInitializeCallback.OnAfterInitialize()
    {
        money = 100;
    }
}
```

## 1.2 IAfterInitializeRollback<T>

説明

初期化処理中に例外が発生し、[IAfterInitializeCallback](#) で実装したコールバックによって発生した副作用をロールバックする場合はこのインターフェースを使用します。

呼び出されるケース

- 初期化処理中に例外が発生したとき、[IAfterInitializeCallback.OnAfterInitialize\(\)](#) が呼ばれていた

呼び出されないケース

- [IAfterInitializeCallback](#)を実装していない
- 初期化処理中に例外が発生したとき、[IAfterInitializeCallback.OnAfterInitialize\(\)](#) が呼ばれていなかった

Public 関数

| 関数名                                       | 説明                        |
|---|---------------------------|
| <a href="#">OnAfterInitializeRollback</a> | データ初期化処理のロールバック時に呼び出されます。 |

OnAfterInitializeRollback

- public void **OnAfterInitializeRollback** (T previousData);

説明

データ初期化処理のロールバック時に呼び出されます。

```
using System;
using SaveDesign.Runtime;
using UnityEngine;

[SharedData, Serializable]
public class ExampleClass : IAfterInitializeCallback, IAfterInitializeRollback<ExampleClass>
{
    [SerializeField] int frameRate;

    public void SetFrameRate(int frameRate)
    {
        this.frameRate = frameRate;
        Application.targetFrameRate = frameRate;
    }

    void IAfterInitializeCallback.OnAfterInitialize()
    {
        SetFrameRate(60);
    }

    void IAfterInitializeRollback.OnAfterInitializeRollback(ExampleClass previousData)
    {
        if (previousData != null) Application.targetFrameRate = previousData.frameRate;
        else Application.targetFrameRate = -1;
    }
}
```

1.3 IAfterLoadCallback

説明

データが読み込まれた後に何らかの処理を実行したい場合はこのインターフェースを使用します。

このインターフェースは `SharedData` 属性、`SlotData` 属性、`SlotMetaData` 属性のいずれかを付与したクラスに実装する必要があり、`TempData` 属性のみを付与したクラスやどのデータ属性も付与していないクラスに実装した場合は無視されます。

Public 関数

| 関数名                         | 説明                    |
|-----------------------------|-----------------------|
| <a href="#">OnAfterLoad</a> | データが読み込まれた直後に呼び出されます。 |

OnAfterLoad

- public void **OnAfterLoad** ();

説明

データが読み込まれた後に呼び出されます。

```
using System;
using SaveDesign.Runtime;

[SharedData, Serializable]
public class ExampleClass : IAfterLoadCallback
{
    public int numberOfStartups;

    void IAfterLoadCallback.OnAfterLoad()
    {
        numberOfStartups++;
    }
}
```

1.4 IAfterLoadRollback<T>

説明

読み込み処理中に例外が発生し、[IAfterLoadCallback](#) で実装したコールバックによって発生した副作用をロールバックする場合はこのインターフェースを使用します。

Public 関数

| 関数名                                 | 説明                         |
|-------------------------------------|----------------------------|
| <a href="#">OnAfterLoadRollback</a> | データ読み込み処理のロールバック時に呼び出されます。 |

OnAfterLoadRollback

- public void **OnAfterLoadRollback** (T previousData);

説明

データ読み込み処理のロールバック時に呼び出されます。

```
using System;
using SaveDesign.Runtime;
using UnityEngine;

[SharedData, Serializable]
public class ExampleClass : IAfterLoadCallback, IAfterLoadRollback<ExampleClass>
```

```

{
    [SerializedField] int frameRate;

    public void SetFrameRate(int frameRate)
    {
        this.frameRate = frameRate;
        Application.targetFrameRate = frameRate;
    }

    void IAfterLoadCallback.OnAfterLoad()
    {
        SetFrameRate(frameRate);
    }

    void IAfterLoadRollback.OnAfterLoadRollback(ExampleClass previousData)
    {
        if (previousData != null) Application.targetFrameRate = previousData.frameRate;
        else Application.targetFrameRate = -1;
    }
}

```

## 1.5 IBeforeSaveCallback

### 説明

データが保存される前に何らかの処理を実行したい場合はこのインターフェースを使用します。

このインターフェースは `SharedData` 属性、`SlotData` 属性、`SlotMetaData` 属性のいずれかを付与したクラスに実装する必要があり、`TempData` 属性のみを付与したクラスやどのデータ属性も付与していないクラスに実装した場合は無視されます。

### Public 関数

| 関数名                          | 説明                   |
|------------------------------|----------------------|
| <a href="#">OnBeforeSave</a> | データが保存される直前に呼び出されます。 |

### OnBeforeSave

- public void **OnBeforeSave** ();

### 説明

データが保存される前に呼び出されます。

```

using System;
using SaveDesign.Runtime;

[SharedData, Serializable]
public class ExampleClass : IBeforeSaveCallback
{
    static readonly DateTime s_epoch = new(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);

    // JsonUtilityの場合、DateTime 型のシリアルライズに非対応のため long 型で保存する。

```

```

public long saveDateTime;

public DateTime SaveDateTime => s_epoch.AddMilliseconds(saveDateTime).ToLocalTime();

void IBeforeSaveCallback.OnBeforeSave()
{
    // データが保存される直前に DateTime 型を long 型に変換して書き込む
    saveDateTime = (long)(DateTime.Now.ToUniversalTime() - s_epoch).TotalMilliseconds;
}
}

```

## 1.6 IBeforeSaveRollback

### 説明

書き込み処理中に例外が発生し、[IBeforeSaveCallback](#) で実装したコールバックによって発生した副作用をロールバックする場合はこのインターフェースを使用します。

#### Public 関数

| 関数名                                  | 説明                         |
|--------------------------------------|----------------------------|
| <a href="#">OnBeforeSaveRollback</a> | データ書き込み処理のロールバック時に呼び出されます。 |

#### OnBeforeSaveRollback

- public void **OnBeforeSaveRollback** ();

### 説明

データ書き込み処理のロールバック時に呼び出されます。

## 1.7 ISaveDesignConfig

### 説明

データの保存に関する設定を提供するインターフェースです。

#### Public 関数

| 関数名                                      | 説明                               |
|--|----------------------------------|
| <a href="#">GetSaveDataDirectoryPath</a> | ファイルを保存するディレクトリパスを取得する。          |
| <a href="#">GetSharedDataFileName</a>    | 共有データを保存するファイル名を取得する。            |
| <a href="#">GetSlotDataFileName</a>      | セーブスロットごとに分けて保存するデータのファイル名を取得する。 |
| <a href="#">GetFileExtension</a>         | 保存するファイルの拡張子を取得する。               |
| <a href="#">GetExceptionPolicy</a>       | 例外の扱い方を取得する。                     |

#### GetSaveDataDirectoryPath

- public string **GetSaveDataDirectoryPath** ();

## 説明

ファイルを保存するディレクトリパスを取得する。

特にこだわりが無ければ `Application.persistentDataPath` + ディレクトリ名 を返すよう実装することを推奨します。

Android など一部のプラットフォームは、`Application.persistentDataPath` 配下のディレクトリパスを返さなければ読み書きができません。

---

## GetSharedDataFileName

- `public string GetSharedDataFileName ();`

## 説明

共有データを保存するファイル名を取得する。

---

## GetSlotDataFileName

- `public string GetSlotDataFileName ();`

## 説明

スロット固有のデータを保存するファイル名を取得する。

---

## GetFileExtension

- `public string GetFileExtension ();`

## 説明

保存するファイルの拡張子を取得する。

`null` や空文字を返すと拡張子のないファイルが生成されます。

---

## GetExceptionPolicy

- `public ExceptionPolicy GetExceptionPolicy ();`

## 説明

初期化、読み込み、保存処理中に例外が発生した場合の動作の種類を取得する。

---

## 2. 属性

### 2.1 EncryptorAttribute

#### 説明

データの読み書き時に暗号化処理・複合化処理を組み込む場合に使用する。

```
using SaveDesign.Runtime;

[Encryptor]
public static class CustomEncryptor
{
    public static void Encrypt(ref byte[] data)
    {
        ...
    }

    public static void Decrypt(ref byte[] data)
    {
        ...
    }
}
```

#### コンストラクタ

- public EncryptorAttribute ();

### 2.2 SaveDesignRootAttribute

#### 説明

セーブデータを管理する中核クラスに付与します。

この属性が付与されたクラスには、初期化、保存、読み込み、削除などを行うためのエントリポイントが自動生成されます。

詳細は [SaveDesignRoot 属性を付与されたクラス](#) をご確認ください。

#### コンストラクタ

- public SaveDesignRootAttribute ([SerializerType](#) serializerType);

| パラメーター名                        | 説明  |
|--------------------------------|---|
| <a href="#">serializerType</a> | 使用するシリアライザーの種類。(デフォルト値: SerializerType.JsonUtility) |

```
using System;
using SaveDesign.Runtime;

[SaveDesignRoot]
internal partial class ExampleClass { }

// 下記のようにアクセスできる
```



```
ExampleClass.Shared
ExampleClass.Slot
ExampleClass.Load
```

### serializerType

#### 説明

使用するシリアライザーを設定できます。

詳細は [SerializerType](#) セクションをご確認ください。

## 2.3 SharedDataAttribute

#### 説明

すべてのセーブスロットで共有されるデータを定義する属性です。

プレイヤー全体の進行状況やグローバル設定などに適しています。

#### コンストラクタ

- `public SharedDataAttribute ();`
- `public SharedDataAttribute (string path);`
- `public SharedDataAttribute (params Type[] dependsOnTypes);`
- `public SharedDataAttribute (string path, params Type[] dependsOnTypes);`

| パラメーター名                        | 説明                 |
|--------------------------------|--------------------|
| <a href="#">path</a>           | データにアクセスするための階層パス。 |
| <a href="#">dependsOnTypes</a> | 読み書き時に依存するデータのリスト。 |

```
using System;
using SaveDesign.Runtime;

[SharedData, Serializable]
public class ExampleClass
{
    public string value;
}

// 下記のようにアクセスできる
SD.Shared.ExampleClass.value = "shared data example.";

var value = SD.Shared.ExampleClass.value;
```

### path

#### 説明

データにアクセスするための階層パスです。

階層を分けてデータの整理ができます。

スラッシュ区切りのパスを設定すると複数階層に分けられます。

また、階層パスは `static partial` クラスで生成されるため自由に拡張できます。

```
using System;
using SaveDesign.Runtime;

[SharedData("Path1/Path2"), Serializable]
public class ExampleClass { }

// 下記のようにアクセスできる
SD.Shared.Path1.Path2.ExampleClass
```

---

## dependsOnTypes

### 説明

データの読み書き時に依存する他のデータを設定できます。

これを設定することで、依存するすべてのデータの読み書きを終えてからこのデータの読み書きが実行されます。

ただし、依存するすべてのデータが同じ種類のデータでなければいけません。

また、循環するような依存関係は設定はできません。

```
using System;
using SaveDesign.Runtime;

[SharedData, Serializable]
public class A
{
    public bool flag;
}

[SharedData(typeof(A)), Serializable]
public class B : IAfterLoadCallback
{
    public int value;

    void IAfterLoadCallback.OnAfterLoad()
    {
        // 読み込み後の処理でデータAに依存している
        if (SD.Shared.A.flag) value += 50;
    }
}

[SlotData, Serializable]
public class SlotData { }

// 依存先のデータと種類が異なるためエラーになる。
[SharedData(typeof(SlotData)), Serializable]
public class C { }
```

---

## 2.4 SlotDataAttribute

説明

セーブスロットごとに分けて保存されるデータを定義する属性です。

キャラクターの状態、所持アイテム、進行チャプターなど個別のゲーム進行に関わるデータに適しています。

コンストラクタ

- public SlotDataAttribute ();
- public SlotDataAttribute (string path);
- public SlotDataAttribute (params Type[] dependsOnTypes);
- public SlotDataAttribute (string path, params Type[] dependsOnTypes);

| パラメーター名                        | 説明                 |
|--------------------------------|--------------------|
| <a href="#">path</a>           | データにアクセスするための階層パス。 |
| <a href="#">dependsOnTypes</a> | 読み書き時に依存するデータのリスト。 |

```
using System;
using SaveDesign.Runtime;

[SlotData, Serializable]
public class ExampleClass
{
    public string value;
}

// 下記のようにアクセスできる
SD.Slot.ExampleClass.value = "slot data example.";

var value = SD.Slot.ExampleClass.value;
```

path

説明

データにアクセスするための階層パスです。

階層を分けてデータの整理ができます。

スラッシュ区切りのパスを設定すると複数階層に分けられます。

また、階層パスは `static partial` クラスで生成されるため自由に拡張できます。

```
using System;
using SaveDesign.Runtime;

[SlotData("Path1/Path2"), Serializable]
public class ExampleClass { }
```

```
// 下記のようにアクセスできる
SD.Slot.Path1.Path2.ExampleClass
```

## dependsOnTypes

### 説明

データの読み書き時に依存する他のデータを設定できます。

これを設定することで、依存するすべてのデータの読み書きを終えてからこのデータの読み書きが実行されます。

ただし、依存するすべてのデータが同じ種類のデータでなければいけません。

また、循環するような依存関係は設定はできません。

```
using System;
using SaveDesign.Runtime;

[SlotData, Serializable]
public class A
{
    public bool flag;
}

[SlotData(typeof(A)), Serializable]
public class B : IAfterLoadCallback
{
    public int value;

    void IAfterLoadCallback.OnAfterLoad()
    {
        // 読み込み後の処理でデータAに依存している
        if (SD.Slot.A.flag) value += 50;
    }
}

[SharedData, Serializable]
public class SharedData { }

// 依存先のデータと種類が異なるためエラーになる。
[SlotData(typeof(SharedData)), Serializable]
public class C { }
```

---

## 2.5 SlotMetaDataAttribute

### 説明

各セーブスロットに付随するメタ情報を定義する属性です。

実際のセーブデータとは分離され、セーブスロットの一覧表示などに活用できます。

ただし、メタ情報は他の種類のデータとは異なり、保存するたびに新しいデータとして作成されます。

そのため、読み込んだメタ情報の値を直接書き換えても保存されません。

これは、メタ情報が他の種類のデータから自動的に生成されることを保証する仕組みです。

---

### コンストラクタ

- public SlotMetaDataAttribute ();

```

using System;
using SaveDesign.Runtime;

[SlotMetaData, Serializable]
public class ExampleClass : IBeforeSaveCallback
{
    public string playerName;
    public int level;
    public int money;

    void IBeforeSaveCallback.OnBeforeSave()
    {
        var player = SD.Slot.Player;
        playerName = player.name;
        level = player.level;
        money = player.money;
    }
}

// 下記のように読み込むことができる
if (SD.Load.SlotMeta(slotIndex, out var meta))
{
    var info = meta.playerName + ", " + meta.level + ", " + meta.money;

    meta.playerName = "dummy name"; // ✕メタ情報の値を書き換えても保存されない
}

// 保存は SlotData を保存するときに一緒に保存される

```

## 2.6 TempDataAttribute

### 説明

保存されない一時的なデータです。ゲームセッション中にのみ有効なフラグや一時的な状態の保存に使用します。どのタイミングでリセットされるかは、[TempDataResetTiming](#) により制御できます。

### コンストラクタ

- public TempDataAttribute ();
- public TempDataAttribute (string path);
- public TempDataAttribute ([TempDataResetTiming](#) resetTiming);
- public TempDataAttribute (params Type[] dependsOnTypes);
- public TempDataAttribute (string path, [TempDataResetTiming](#) resetTiming);
- public TempDataAttribute (string path, params Type[] dependsOnTypes);
- public TempDataAttribute ([TempDataResetTiming](#) resetTiming, params Type[] \* dependsOnTypes\*);
- public TempDataAttribute (string path, [TempDataResetTiming](#) resetTiming, params Type[] dependsOnTypes);

| パラメーター名              | 説明                 |
|----------------------|--------------------|
| <a href="#">path</a> | データにアクセスするための階層パス。 |

|                                |  |
|--------------------------------|--|
| <a href="#">resetTiming</a>    | 一時データをリセットするタイミング。(デフォルト値: TempDataResetTiming.OnSharedDataLoad) |
| <a href="#">dependsOnTypes</a> | 読み書き時に依存するデータのリスト。   |

```
using SaveDesign.Runtime;

[TempData]
public class ExampleClass
{
    public string value;
}

// 下記のようにアクセスできる
SD.Temp.ExampleClass.value = "temp data example.";

var value = SD.Temp.ExampleClass.value;
```

### path

#### 説明

データにアクセスするための階層パスです。

階層を分けてデータの整理ができます。

スラッシュ区切りのパスを設定すると複数階層に分けられます。

また、階層パスは `static partial` クラスで生成されるため自由に拡張できます。

```
using SaveDesign.Runtime;

[TempData("Path1/Path2")]
public class ExampleClass { }

// 下記のようにアクセスできる
SD.Temp.Path1.Path2.ExampleClass
```

### resetTiming

#### 説明

一時データをリセットするタイミングを設定できます。

セーブスロットごとに分けて使用したい一時データがある場合などにこの値を設定することで、適切なタイミングで自動的にリセットされます。

この仕組みによって、**初期化忘れによるバグを未然に防ぎます**。

```
using SaveDesign.Runtime;

[TempData(TempDataResetTiming.OnSlotDataLoad)]
public class ExampleClass
{
    public string value = "reset";
}
```

```
// OnSlotDataLoad の場合、セーブスロットを読み込むとリセットされる  
SD.Temp.ExampleClass.value = "example";
```

```
var value = SD.Temp.ExampleClass.value; // example
```

```
if (SD.Load.Slot("identifier"))  
{  
    value = SD.Temp.ExampleClass.value; // reset  
}
```

## dependsOnTypes

### 説明

データの初期化処理で依存する他のデータを設定できます。

これを設定することで、依存するすべてのデータの初期化処理を終えてからこのデータの初期化処理が実行されます。

ただし、依存するすべてのデータが同じ種類のデータでなければならず、  
一時データの場合はリセットタイミングも依存先のデータと同じでなければいけません。

また、循環するような依存関係は設定はできません。

```
using SaveDesign.Runtime;  
  
[TempData]  
public class A  
{  
    public bool flag;  
}  
  
[TempData(typeof(A))]  
public class B  
{  
    public int value;  
  
    public B()  
    {  
        // 初期化時の処理でデータAの値に依存している  
        if (SD.Temp.A.flag) value += 50;  
    }  
}  
  
[SlotData, System.Serializable]  
public class SlotData { }  
  
// ✕ 依存先のデータと種類が異なるためエラーになる  
[TempData(typeof(SlotData))]  
public class C { }  
  
// ✕ 依存先のデータとリセットタイミングが異なるためエラーになる
```

```
[TempData(TempDataResetTiming.OnGameStart, typeof(A))]  
public class D { }
```

---



### 3. 列挙型

#### 3.1 ExceptionPolicy

説明

初期化処理や読み書き処理中に発生した例外の扱い方の種類。

変数

| 変数名            | 説明             |
|----------------|----------------|
| Throw          | そのままスローする。     |
| LogAndSuppress | ログに出力して握りつぶす。  |
| Suppress       | ログに出力せずに握りつぶす。 |

#### 3.2 SerializerType

説明

シリアライザーの種類。

これを `SaveDesignRoot` 属性に設定することで使用するシリアライザーの種類を切り替えられます。

```
using SaveDesign.Runtime;

[SaveDesignRoot(SerializerType.MessagePack)]
internal partial class SD { }
```

変数

| 変数名             | 説明                               |
|-----------------|----------------------------------|
| JsonUtility     | Unity 標準の JSON ライブラリ。            |
| MessagePack     | MessagePack for C# [MIT License] |
| Newtonsoft.Json | Newtonsoft.Json [MIT License]    |

#### 3.3 TempDataResetTiming

説明

一時データのリセットタイミングの種類。

変数

| 変数名              | 説明                         |
|------------------|----------------------------|
| OnGameStart      | ゲーム起動時に一度だけリセットする。         |
| OnSharedDataLoad | 共有データの初期化時または読み込み時にリセットする。 |

|                |                              |
|----------------|------------------------------|
| OnSlotDataLoad | セーブスロットの初期化時または読み込み時にリセットする。 |
| Manual         | 手動でリセットする。                   |

## 4. クラス

### 4.1 SaveDesignRoot 属性を付与されたクラス

#### 説明

#### Static 変数

| 変数名                              | 説明                          |
|----------------------------------|-----------------------------|
| <a href="#">config</a>           | データの保存に関する設定                |
| <a href="#">currentSlotIndex</a> | 現在読み込んでいるセーブスロット番号。(読み取り専用) |

#### config

public static [ISaveDesignConfig](#) config;

#### 説明

これに設定したディレクトリパスやファイル名を使用してデータを保存する。

データの読み書きをする前に必ず設定する必要があります。

詳細は [ISaveDesignConfig](#) セクションをご確認ください。

#### currentSlotIndex

- public static int **currentSlotIndex**;

#### 説明

現在読み込んでいるセーブスロットの番号を返します。(読み取り専用)

[SlotData](#) 属性が付与されたクラスが1つ以上ある場合に使用できます。

初期化や読み書きを行うことで適切な値に自動的に更新されます。

| 操作                      | currentSlotIndex の変化 |
|-------------------------|----------------------|
| Initialize.Slot()       | -1 に設定される            |
| Load.Slot(identifier)   | -1 に設定される            |
| Load.Slot(slotIndex)    | slotIndex が設定される     |
| Save.Slot(slotIndex)    | slotIndex が設定される     |
| Save.Slot(identifier)   | 変更なし (そのままの値を維持)     |
| Delete.Slot(slotIndex)  | 変更なし (そのままの値を維持)     |
| Delete.Slot(identifier) | 変更なし (そのままの値を維持)     |

```
var slotIndex = SD.currentSlotIndex; // -1

if (SD.Load.Slot(0))
{
    slotIndex = SD.currentSlotIndex; // 0
}

if (SD.Save.Slot(1))
{
    slotIndex = SD.currentSlotIndex; // 0
}

if (SD.Load.Slot("identifier"))
{
    slotIndex = SD.currentSlotIndex; // -1
}
```

## エントリポイント

`SaveDesignRoot` 属性を付与されたクラスは初期化や読み書きなどを実行するためのエントリポイントが自動生成されます。

また、エントリポイントは `static partial` クラスで生成されるため、自由に拡張することができます。

生成されるエントリポイントは以下の通りです。

| 名前                         | 説明                                      |
|----------------------------|---|
| <a href="#">Initialize</a> | データを初期化する。                              |
| <a href="#">Load</a>       | データを読み込む。                               |
| <a href="#">Save</a>       | データを保存する。                               |
| <a href="#">Delete</a>     | データを削除する。                               |
| <a href="#">Shared</a>     | 共有データにアクセスするためのエントリポイント。                |
| <a href="#">Slot</a>       | セーブスロットごとに分けて保存するデータにアクセスするためのエントリポイント。 |
| <a href="#">Temp</a>       | 保存されない一時データにアクセスするためのエントリポイント。          |

## Initialize

### 説明

データの初期化関数へのエントリポイントです。

## Static 関数

- `public static void Shared ();`
- `public static void Shared (ExceptionPolicy exceptionPolicy);`

### 説明

共有データを初期化する。

共有データの場合はゲーム起動時に一度だけ読み込みを実行して、失敗した場合に初期化する処理を実装することを推奨します。

```
[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
static void InitSaveDesignConfig()
{
    // 保存関連の設定
    SD.config = Resources.Load<SaveDesignConfig>("SaveDesignConfig");

    // 共有データの読み込みに失敗したら初期化する
    if (!SD.Load.Shared()) SD.Initialize.Shared();
}
```

- public static void **Slot** ();
- public static void **Slot** ([ExceptionPolicy](#) exceptionPolicy);

#### 説明

セーブスロットごとに分けて保存するデータを初期化する。

新しくゲームを始めるときに実行します。

```
public void NewGame()
{
    SD.Slot.Player.money = 100; // ✖ データが初期化されていないためエラー

    SD.Initialize.Slot();

    SD.Slot.Player.money = 100; // ✔ OK
}
```

## Load

#### 説明

データの読み込み関数へのエントリーポイントです。

#### Static 関数

- public static bool **Shared** ();
- public static bool **Shared** ([ExceptionPolicy](#) exceptionPolicy);

#### 説明

共有データを読み込む。

共有データの場合はゲーム起動時に一度だけ読み込みを実行して、失敗した場合に初期化する処理を実装することを推奨します。

```
[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
static void InitSaveDesignConfig()
{
    // 保存関連の設定
    SD.config = Resources.Load<SaveDesignConfig>("SaveDesignConfig");

    // 共有データの読み込みに失敗したら初期化する
    if (!SD.Load.Shared()) SD.Initialize.Shared();
}
```

- public static bool **Slot** (int slotIndex);
- public static bool **Slot** (int slotIndex, [ExceptionPolicy](#) exceptionPolicy);
- public static bool **Slot** (string identifier);
- public static bool **Slot** (string identifier, [ExceptionPolicy](#) exceptionPolicy);

#### 説明

セーブスロットごとに分けて保存するデータを**スロット番号**、もしくは**識別子**を指定して読み込む。

セーブスロットに保存したデータを引き継いでゲームを開始する場合に実行します。

また、スロット番号による読み込み関数は**セーブスロットのデータを読み込む**ときに使用し、識別子による読み込み関数はオートセーブやチェックポイントなど **セーブスロットとは関係のないデータを読み込む**ときに使用することを推奨します。

```
public void LoadGame(int slotIndex)
{
    if (SD.Load.Slot(slotIndex))
    {
        // データが読み込めたら次のシーンへ遷移する
        SceneManager.LoadScene("Next Scene");
    }
}
```

- public static bool **SlotMeta** (int slotIndex, out ? meta);
- public static bool **SlotMeta** (int slotIndex, out ? meta, [ExceptionPolicy](#) exceptionPolicy);
- public static bool **SlotMeta** (string identifier, out ? meta);
- public static bool **SlotMeta** (string identifier, out ? meta, [ExceptionPolicy](#) exceptionPolicy);

#### 説明

各セーブスロットに付随するメタ情報を**スロット番号**、もしくは**識別子**を指定して読み込む。

セーブデータのロード画面で各セーブスロットの情報を表示するときに実行します。

スロット番号と識別子の使い分け方は、セーブスロットごとに分けて保存するデータのときと同じです。

```
// セーブスロットのUIリスト
[SerializeField] SaveSlotUI[] slots;

public void DisplaySaveSlots()
{
    for(int i = 0; i < slots.Length; i++)
    {
        if (SD.Load.SlotMeta(i, out var meta))
        {
            // メタ情報があれば保存されたデータがあるということ
            slots[i].UpdateUI(i, meta);
        }
        else
        {
            // メタ情報がなければ、そのセーブスロットは空である
            slots[i].UpdateUI(i, "no data");
        }
    }
}
```

```
}  
}
```

---

## Save

### 説明

データの保存関数へのエントリポイントです。

---

### Static 関数

- public static bool **Shared** ();
- public static bool **Shared** ([ExceptionPolicy](#) exceptionPolicy);

### 説明

共有データを保存する。

ゲーム設定を変更した後や、ゲーム終了時に呼び出すことを推奨します。

```
public void SaveSetting()  
{  
    if (SD.Save.Shared())  
    {  
        ...  
    }  
}
```

- 
- public static bool **Slot** (int slotIndex);
  - public static bool **Slot** (int slotIndex, [ExceptionPolicy](#) exceptionPolicy);
  - public static bool **Slot** (string identifier);
  - public static bool **Slot** (string identifier, [ExceptionPolicy](#) exceptionPolicy);

### 説明

セーブスロットごとに分けて保存するデータを**スロット番号**、もしくは**識別子**を指定して保存する。

スロット番号による保存は**セーブスロットのデータを保存する**ときに使用し、識別子による保存はオートセーブやチェックポイントなど **セーブスロットとは関係のないデータを保存する**ときに使用することを推奨します。

メタ情報も保存されます。

```
public void SaveGame(int slotIndex)  
{  
    if (SD.Save.Slot(slotIndex))  
    {  
        ...  
    }  
}
```

---

## Delete

### 説明

データの削除関数へのエントリポイントです。

---

### Static 関数

- public static bool **Shared** ();
- public static bool **Shared** ([ExceptionPolicy](#) exceptionPolicy);

#### 説明

共有データを削除する。

```
SD.Delete.Shared();
```

- public static bool **Slot** (int slotIndex);
- public static bool **Slot** (int slotIndex, [ExceptionPolicy](#) exceptionPolicy);
- public static bool **Slot** (string identifier);
- public static bool **Slot** (string identifier, [ExceptionPolicy](#) exceptionPolicy);

#### 説明

セーブスロットごとに分けて保存するデータをスロット番号、もしくは識別子を指定して削除する。

メタ情報も削除されます。

```
SD.Delete.Slot(0);
```

## Shared

#### 説明

共有データへのエントリポイントです。

`static partial` クラスで生成されるため、自由に拡張できます。

```
using System;
using SaveDesign.Runtime;

[SharedData, Serializable]
public class ExampleClass
{
    public int value;
}

// 下記のようにアクセスできる
SD.Shared.ExampleClass.value = 10;
```

## Slot

#### 説明

セーブスロットごとに分けて保存するデータへのエントリポイントです。

`static partial` クラスで生成されるため、自由に拡張できます。

```
using System;
using SaveDesign.Runtime;

[SlotData, Serializable]
public class ExampleClass
```

```
{
    public int value;
}

// 下記のようにアクセスできる
SD.Slot.ExampleClass.value = 10;
```

---

## Temp

### 説明

保存されない一時データへのエントリポイントです。

`static partial` クラスで生成されるため、自由に拡張できます。

```
using SaveDesign.Runtime;

[TempData]
public class ExampleClass
{
    public int value;
}

// 下記のようにアクセスできる
SD.Temp.ExampleClass.value = 10;
```

---

## 5. サードパーティ ライセンス

本パッケージは、以下のライブラリを参照するコードを生成する可能性があります：

- [MessagePack for C#](#) — MIT License
- [Newtonsoft.Json](#) — MIT License

これらのライブラリはパッケージに含まれていません。ライセンスの詳細については、`Third-Party Notices.txt` を参照してください。