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 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
OnBeforeSave	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;
    }
}
```

1.2 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
OnAfterInitialize	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;
    }
}
```

1.3 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
OnAfterLoad	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 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
GetSaveDataDirectoryPath	ファイルを保存するディレクトリパスを取得する。
GetSharedDataFileName	共有データを保存するファイル名を取得する。
GetSlotDataFileName	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。
GetFileExtension	保存するファイルの拡張子を取得する。

GetSaveDataDirectoryPath

- public void **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 void **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 void **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 void **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。

---

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

serializerType	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
path	データにアクセスするための階層パス。
dependsOnTypes	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
path	データにアクセスするための階層パス。
dependsOnTypes	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 **SlotDataAttribute** ();

```
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
path	データにアクセスするための階層パス。
resetTiming	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`)
dependsOnTypes	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 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 parital 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]

3.2 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
config	データの保存に関する設定
currentSlotIndex	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
Initialize	データを初期化する。
Load	データを読み込む。
Save	データを保存する。
Delete	データを削除する。
Shared	共有データにアクセスするためのエントリポイント。
Slot	セーブスロットごと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。
Temp	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。

条件を満たした場合、 `Initialize` , `Load` , `Save` , `Delete` の4つのエントリポイントには `UniTask` か `Awaitable` がベースの **非同期関数**が生成されます。

`UniTask` の場合、プロジェクトに`UniTask`を導入し、スクリプティングシンボル `SAVE_DESIGN_SUPPORT_UNITASK` を定義することで、UniTask ベースの非同期関数が生成されます。

`Awaitable` の場合、バージョンが `Unity 2023.1` 以降のプロジェクトであれば自動的に Awaitable ベースの非同期関数が生成されます。
ただし、 `UniTask` ベースの非同期関数を生成する条件を満たしていた場合はそちらが優先されます。

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** ();

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。

```
SD.Initialize.Shared();             // 同期
await SD.Initialize.Async.Shared(); // 非同期


[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** ();

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。

```
SD.Initialize.Slot();             // 同期
await SD.Initialize.Async.Slot(); // 非同期

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ポイントです。

Static 関数

- public static bool **Shared** ();

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。

```
SD.Load.Shared();             // 同期
await SD.Load.Async.Shared(); // 非同期


[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** (string **identifier**);

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。

```
SD.Load.Slot(0);             // 同期
await SD.Load.Async.Slot("auto"); // 非同期


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** (string **identifier**, out ? **meta**);

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。

また、非同期関数は生成されません。

```
// 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。

メタ情報は `Slot` の保存時に自動的に保存されます。

Static 関数

- public static bool **Shared** ();

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。

```
SD.Save.Shared();             // 同期
await SD.Save.Async.Shared(); // 非同期


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

- public static bool **Slot** (int **slotIndex**);
- public static bool **Slot** (string **identifier**);

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。

```
SD.Save.Slot(0);             // 同期
await SD.Save.Async.Slot("auto"); // 非同期


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。

メタ情報は `Slot` の削除時に自動的に削除されます。

Static 関数

- public static bool **Shared** ();

説明

共有データを削除する。

```
SD.Delete.Shared();             // 同期
await SD.Delete.Async.Shared(); // 非同期
```

- public static bool **Slot** (int **slotIndex**);
- public static bool **Slot** (string **identifier**);

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。

```
SD.Delete.Slot(0);             // 同期
await SD.Delete.Async.Slot("auto"); // 非同期
```

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;
```

Private partial 関数

関数名	説明
OnGameDataError	初期化や読み書き処理中に発生した例外を受け取る。

OnGameDataError

説明

初期化や読み書き処理中に発生した例外を受け取る。

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

[SaveDesignRoot]
internal partial class SD
{
    static partial void OnGameDataError(Exception e)
    {
        Debug.LogException(e);
    }
}
```

4.2 Encryptor

internal static partial class **Encryptor**

説明

暗号化処理を組み込みたい場合は、このクラスの部分メソッドを実装してください。

Private static partial 関数

関数名	説明
Encrypt	データを暗号化する。
Decrypt	データを複合化する。

Encrypt

- static partial void **Encrypt** (ref byte[] **data**);

説明

データの暗号化を組み込むための部分関数。

引数の `data` に対して暗号化後の `byte[]` を代入することで返す。

```
namespace SaveDesign.Runtime
{
    internal static partial class Encryptor
    {
        static partial void Encrypt(ref byte[] data)
        {
            ...
        }
    }
}
```

Decrypt

- static partial void **Decrypt** (ref byte[] **data**);

説明

データの複合化を組み込むための部分関数。

引数の `data` に対して暗号化後の `byte[]` を代入することで返す。

```
namespace SaveDesign.Runtime
{
    internal static partial class Encryptor
    {
        static partial void Decrypt(ref byte[] data)
        {
            ...
        }
```

```
        }
    }
```

---

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
- UniTask — 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。