DataSaving System

Complete data persistence solution with compression, encryption, and multi-slot support

Quick Setup

1.
Add SetupWizard to your scene
Attach the `SetupWizard` component to a GameObject in your scene. It will automatically initialize the system on Awake.

2.
Configure settings
Set compression type (None/GZIP), encryption (None/AES), and enable auto-slot generation for quick start.

3.
Start saving data
Use the simple one-liner API to save and load your data immediately.

Basic Usage

```
// Save data
DataManager.Save("playerName", "John");
DataManager.Save("playerLevel", 42);
DataManager.Save("playerPosition", transform.position);

// Load data
string name = DataManager.Load<string>("playerName");
int level = DataManager.Load<int>("playerLevel");
Vector3 position = DataManager.Load<Vector3>("playerPosition");
```

Features

GZIP Compression

Reduce file sizes with built-in GZIP compression for efficient storage.

AES Encryption

Secure your save data with industry-standard AES encryption.

Multi-Slot Support

Create, manage, and switch between multiple save slots seamlessly.

Local File Storage

Save data directly to disk with automatic file management.

Built-in Caching

Faster load times through intelligent caching mechanisms.

Async Support

Choose between synchronous and asynchronous operations based on your needs.

Custom Type Converters

Built-in support for Unity types that Newtonsoft.Json doesn't handle natively:

`Vector2`
`Vector3`
`Vector4`
`Vector2Int`
`Vector3Int`
`Quaternion`
`Color`
`Color32`
`Rect`
`RectInt`
`Bounds`
`BoundsInt`
`Matrix4x4`
`Ray`
`Ray2D`
`Plane`
`AnimationCurve`
`Gradient`

API Reference

Core Operations

```
DataManager.Save(string key, object value)
```

Saves a value synchronously with the specified key.

```
DataManager.Save("score", 1500);
DataManager.Save("settings", gameSettings);
```

`DataManager.SaveAsync(string key, object value)`

Saves a value asynchronously with the specified key.

```
await DataManager.SaveAsync("largeData", bigObject);
DataManager.SaveAsync("config", config); // Fire and forget
```

`DataManager.Load<T>(string key)`

Loads a value synchronously. Returns default(T) if key not found.

```
int score = DataManager.Load<int>("score");
MyClass data = DataManager.Load<MyClass>("userData");
```

`DataManager.LoadAsync<T>(string key)`

Loads a value asynchronously. Returns default(T) if key not found.

```
int score = await DataManager.LoadAsync<int>("score");
var data = await DataManager.LoadAsync<MyClass>("userData");
```

Save Slot Management

`DataManager.CreateSaveSlot(string name)`

Creates a new save slot with the specified name.

`DataManager.SetActiveSlot(string name)`

Sets the active save slot. All save/load operations will use this slot.

`DataManager.DeleteSaveSlot(string name)`

Deletes the specified save slot and its data.

`DataManager.RenameSaveSlot(string oldName, string newName)`

Renames an existing save slot.

Utility Methods

`DataManager.GetSaveSlots()`

Returns a list of all available save slots.

`DataManager.DoesSlotExist(string name)`

Checks if a save slot with the given name exists.

`DataManager.DeleteAllSaveSlots()`

Deletes all save slots and their data. Use with caution.

Configuration

`DataManager.CompressionType`

Set to `Compression.None` or `Compression.GZIP`

`DataManager.EncryptionType`

Set to `Encryption.None` or `Encryption.AES`