DataSaving System

A powerful, thread-safe save system for Unity with encryption, compression, and multi-slot support

Quick Setup

Get started in 30 seconds:

1. Add SetupWizard

Attach the `SetupWizard` component to any GameObject in your scene.

2. Configure Settings

Set compression (None/GZIP), encryption (None/AES), and enable auto-slot generation if desired.

3. Start Saving

Use the simple API to save and load data immediately!

```
using SaveLoadSystem;

// That's it! Start using the API:
DataManager.Save("playerName", "John");
DataManager.Save("level", 42);
DataManager.Save("position", transform.position);

// Load it back:
string name = DataManager.Load<string>("playerName");
int level = DataManager.Load<int>("level");
Vector3 position = DataManager.Load<Vector3>("position");
```

Features

GZIP Compression

Reduce file sizes with built-in compression

AES Encryption

Secure your data with industry-standard encryption

Multi-Slot Support

Create and manage multiple save slots

⚡ Smart Caching

Fast load times with intelligent memory caching

Async Support

Choose sync or async operations

Unity Types

Built-in support for Vector3, Color, etc.

API Reference

Core Save/Load Operations

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

```
void Save(string key, object value)
```

Saves a value synchronously with the specified key. Thread-safe.

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

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

```
async Task SaveAsync(string key, object value)
```

Saves a value asynchronously. Use for large objects or when you don't want to block the main thread.

```
await DataManager.SaveAsync("largeData", bigObject);

// Fire and forget (not recommended for critical data)
DataManager.SaveAsync("config", config);
```

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

```
T 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");

// Returns 0 if "score" doesn't exist
int defaultScore = DataManager.Load<int>("nonexistent");
```

DataManager.LoadAsync<T>(string key)

```
async Task<T> 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");
```

DataManager.DeleteValue(string key)

```
void DeleteValue(string key)
```

Deletes a saved value and removes it from the save file immediately.

```
// Remove old data
DataManager.DeleteValue("oldSetting");
DataManager.DeleteValue("temporaryData");
```

DataManager.DeleteValueAsync(string key)

```
async Task DeleteValueAsync(string key)
```

Deletes a saved value asynchronously.

```
await DataManager.DeleteValueAsync("largeData");
```

Save Slot Management

DataManager.CreateSaveSlot(string name)

```
void CreateSaveSlot(string name)
```

Creates a new save slot with the specified name.

```
DataManager.CreateSaveSlot("Player1");
DataManager.CreateSaveSlot("NewGame+");
```

DataManager.SetActiveSlot(string name)

```
void SetActiveSlot(string name)
```

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

```
DataManager.SetActiveSlot("Player1");

// Now all saves/loads use the "Player1" slot
DataManager.Save("level", 5);
```

DataManager.DeleteSaveSlot(string name)

```
void DeleteSaveSlot(string name)
```

Deletes the specified save slot and all its data permanently.

```
DataManager.DeleteSaveSlot("OldSave");
```

Warning: This permanently deletes all data in the slot. Cannot be undone.

DataManager.RenameSaveSlot(string oldName, string newName)

```
void RenameSaveSlot(string oldName, string newName)
```

Renames an existing save slot.

```
DataManager.RenameSaveSlot("TempSave", "MainSave");
```

Utility Methods

DataManager.GetSaveSlots()

```
List<SaveSlot> GetSaveSlots()
```

Returns a list of all available save slots.

```
var slots = DataManager.GetSaveSlots();
foreach(var slot in slots)
{
    Debug.Log($"Slot: {slot.Name}, Size: {slot.GetFileSize()} bytes");
}
```

DataManager.DoesSlotExist(string name)

```
bool DoesSlotExist(string name)
```

Checks if a save slot with the given name exists.

```
if(DataManager.DoesSlotExist("Player1"))
{
    DataManager.SetActiveSlot("Player1");
}
else
{
    DataManager.CreateSaveSlot("Player1");
}
```

`DataManager.DeleteAllSaveSlots()`

```
void DeleteAllSaveSlots()
```

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

```
// Nuclear option - deletes everything
DataManager.DeleteAllSaveSlots();
```

⚠ **DANGER:** This deletes ALL save data permanently. Typically only used for "Reset All Data" features.

Supported Unity Types

The system includes built-in converters for Unity types that Newtonsoft.Json doesn't handle natively:

Type	Example Usage
Vector2, Vector3, Vector4	`DataManager.Save("pos", transform.position)`
Vector2Int, Vector3Int	`DataManager.Save("gridPos", gridPosition)`
Quaternion	`DataManager.Save("rotation", transform.rotation)`
Color, Color32	`DataManager.Save("playerColor", Color.red)`
Rect, RectInt	`DataManager.Save("bounds", screenRect)`
Bounds, BoundsInt	`DataManager.Save("area", colliderBounds)`
Matrix4x4	`DataManager.Save("transform", transformMatrix)`
Ray, Ray2D	`DataManager.Save("raycast", cameraRay)`
Plane	`DataManager.Save("ground", groundPlane)`

AnimationCurve	`DataManager.Save("curve", animCurve)`
Gradient	`DataManager.Save("gradient", colorGradient)`

Configuration

Configure the system through the SetupWizard component or at runtime:

Compression Options

```
DataManager.CompressionType = Compression.None;  // No compression
DataManager.CompressionType = Compression.GZIP; // GZIP compression
```

Encryption Options

```
DataManager.EncryptionType = Encryption.None; // No encryption
DataManager.EncryptionType = Encryption.AES;  // AES encryption
```

Note: Configuration changes affect saving immediately but existing save files are loaded using their original settings.