Save System

By Lutor Games

Website • Play Store • LinkedIn • YouTube

Tutorial: ▶ Unity Asset | Save System - How to use it

Table of Contents

Core Concept

This system provides a framework to create and manage persistent data storage for custom data types, allowing developers to save and load data with ease. The framework is designed for Unity-based development, supporting saving data types directly into serialised binary files, so they persist across application sessions.

The main component is the Base_Saveable<T> class, which provides a reusable template for defining data types that can be saved and loaded. The system includes various predefined types like **Float_Saveable** and **IntList_Saveable,** and users can also create custom saveable classes based on their specific data requirements by inheriting from **Base_Saveable<T>.**

Quick Start

1. Declare a saveable field:

```
Float_Saveable health;
```

2. Access the Value (get or set):

```
health.Value = 100f;       // Set value
float currentHealth = health.Value;   // Get value
```

3. Customise the saveable with a unique ID and Default Value

- Each saveable should have a unique **Id** to prevent conflicts.
- Set a **DefaultValue** to initialise the saveable if no saved data exists.

Core Components

1. `Base_Saveable<T>`

- Base class for defining saveable data types. Any type inheriting from **Base_Saveable<T>** will be equipped with saving and loading functionalities.
- Properties:
  - **Id:** A unique identifier used to distinguish saveable instances.
  - **DefaultValue:** The initial value used if no saved data is found.
  - **Value:** The current value, which can be set or retrieved.
  - **OnValueChanged:** An event triggered whenever the **Value** changes.
- Methods:
  - **SetValue(T value):** Sets the **Value** and triggers save.
  - **Clear():** Resets **Value** to **DefaultValue** and clears initialization state.

2. `Saveable<T>`

- Encapsulates serialisation and deserialization logic for data persistence.
- Implements the `ISaveAble` interface, which provides a `SaveId` property as a unique identifier for saved data.
- Methods:
  - `Save():` Asynchronously saves data to a binary file.
  - `InstantSave():` Immediately saves data without async.
  - `Load(out T loadedInfo, Action onLoaded = null):` Loads data from the binary file or uses `DefaultValue` if not available.

Predefined Saveables

Several predefined saveable types come out-of-the-box, including:

- **Int_Saveable:** Saveable for **int** values.
- **IntList_Saveable:** Saveable for **List<int>.**
- **Float_Saveable:** Saveable for **float** values.
- **FloatList_Saveable:** Saveable for **List<float>.**
- **String_Saveable:** Saveable for **string** values.
- **StringList_Saveable:** Saveable for **List<string>.**
- **Ushort_Saveable:** Saveable for **ushort** values.
- **UshortList_Saveable:** Saveable for **List<ushort>.**

Usage Example:

```
Float_Saveable playerHealth = new Float_Saveable { Id = "PlayerHealth",
DefaultValue = 100f };
```

Creating Custom Saveable

You can define custom saveable types to store complex data structures. Here's how:

1. Define the custom data class (e.g., a character with various attributes

```
[Serializable]
public class Character
{
    public string Name;
    public int Age;
}
```

2. Create a saveable class for the custom data type.

- Inherit from **Base_Saveable<T>** where **T** is your custom class or a collection of your class.

```
[Serializable]
public class Characters_Saveable : Base_Saveable<List<Character>> { }
```

3. Use the custom saveable class in your code as you would any predefined saveable

```
Characters_Saveable charactersData = new Characters_Saveable
{
    Id = "GameCharacters",
    DefaultValue = new List<Character> { new Character { Name = "Hero",
    Age = 25 } }
};

// Accessing or modifying values
charactersData.Value.Add(new Character { Name = "Villain", Age = 30 });
```

Saving And Loading

The framework automatically saves data when **`Value`** is set. Data is loaded upon the first access, initialised from **DefaultValue** if no saved data exists.

- **Automatic Saving**: Setting **Value** will trigger **Save().**
- **Manual Saving and Loading**:
    - Call **Save()** for async saving or **InstantSave()** for immediate saving.
    - Call **Load()** if you need to explicitly load the latest data from disk.

Automatic Saving Example:

```
Float_Saveable score = new Float_Saveable { Id = "PlayerScore",
DefaultValue = 0f };
score.Value = 10f; // This automatically saves the score
```

There is no actual need for manual loading or saving :)

Advanced Configuration

Unique Identifier (`Id`)

- Each **Base_Saveable<T>** instance requires a unique Id to identify it in the save system.
- If two saveables share the same Id, they will reference the same saved data. Use unique Ids to avoid data conflicts.

Default Values

- Define **DefaultValue** for initialising the saveable when no saved data exists. This ensures your data is never **null** or **undefined.**

Customizable Events

- OnValueChanged: Use this event to react to value changes in the saveable.
- You may assign custom functions to **OnBeforeSave, OnSave, OnBeforeLoaded,** and **OnLoaded** within Saveable<T> to handle lifecycle events for custom save behaviours.

Example:

```
Float_Saveable volumeSetting = new Float_Saveable { Id =
"VolumeSetting", DefaultValue = 0.5f };

volumeSetting.OnValueChanged += (newValue) => Debug.Log("Volume updated:
" + newValue);
```

Notes

- Ensure each **Saveable** has a valid **Id:** Missing **Id** can cause data conflicts.
- Debugging: The system logs errors for missing or invalid **Ids** to avoid issues.
- Binary System: Data is serialised and saved to binary files by default for efficient storage.

This framework allows you to define, store, and retrieve data in a flexible and consistent manner. By using predefined saveables or creating custom saveable classes, you can efficiently manage the persistence of various data types within Unity applications.

Tutorial: ▶ Unity Asset | Save System - How to use it

Website • Play Store • LinkedIn • YouTube