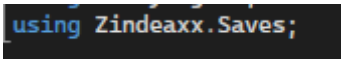
Simple Saving - Documentation

Contents

1. Getting Started:

1. Installation

As soon as you import the package you will be able to use the `SaveGame` class in every single one of your scripts. Since it is a static class you wont have to use any references to use it. Remember to add the correct using to use the `SaveGame` class within your scripts:




```
using Zindeaxx.Saves;
```

2. Saving Data

To save data with the Unity *SaveGame* Plugin, use the *SetObjectValue* function. The *SetObjectValue* function takes two arguments: a string key, and an object value. The key is used to identify the value in the save file, and the value is the data that will be saved. You can save any type of object as long as its data is serializable.

```
SaveGame.SetObjectValue("TestValue", 500);
```

To save a string value, use the `SetObjectValue` function with the string key and value.

```
SaveGame.SetObjectValue("StringVal", "Hi - 202");
```

3. Loading Data

To load data from a save file, use the appropriate Get...Value function. The Get...Value functions return the value of a given key in the save file. If the key is not found, the function returns the default value of the data type.

```
Debug.Log(SaveGame.GetIntValue("TestValue"));
Debug.Log(SaveGame.GetStringValue("StringVal"));
```

4. Checking for Save Files

To check if a save file exists, use the *SaveExists* property.

```
if (SaveGame.SaveExists)
Debug.Log("Save found!");
else
Debug.Log("Save not found!");
```

5. Deleting for Save Files

To delete a save file, use the *DestroySave* function.

```
SaveGame.DestroySave();
```

2. SaveGame Class:

1. Variables

- **SaveFileName: the name of the save file.**
- **SaveFolderLocation: the location of the save data folder.**
- **SaveFileLocation: the location of the main save file.**
- **EncryptionEnabled: a bool variable that determines if encryption is enabled.**
- **SaveFileKey: the encryption key used to encrypt and decrypt save files.**

2. Functions

- **SaveExists**: a property that returns **true** if a save file exists.
- **SaveValues**: a property that returns a **List<SaveGameEntry>** of all the current keys stored in the game's memory.
- **DestroySave**: deletes the save file.
- **GetStringValue(string Key)**: returns a given key's value as a string.
- **GetFloatValue(string Key)**: returns a given key's value as a float.
- **GetIntValue(string Key)**: returns a given key's value as an integer.
- **GetVectorValue(string Key)**: returns a given key's value as a **Vector4**.
- **HasKey(string Key)**: returns **true** if the given key exists in the save file.
- **GetObjectValue(string Key)**: returns the value of a given key in the save file.
- **GetArray<T>(string key)**: returns an array of values of type **T** stored in the save file under the given key.
- **SetObjectValue(string Key, object Value):** This function sets the value of the given key to the given object. If the key already exists, it updates its value. Otherwise, it creates a new key-value pair in the save file. It also saves the updated/created keys to the file.

3. Conclusion:

In conclusion, the SaveGame class provides an easy-to-use interface for saving and retrieving key-value pairs in a save file. The SaveGameExample script demonstrates how to use the SaveGame class to save and retrieve values, and also shows how to enable encryption for the save file. With this package, Unity developers can easily add save functionality to their projects without having to worry about the details of file I/O or encryption.

4. Support/Feedback

- Discord: Zindea#5423

- E-Mail: Zindeaxx@googlemail.com