JSON File Based PlayerPrefs V1.1.2

This Unity library is designed to replace the standard Unity PlayerPrefs.

The main difference being that it saves all records in JSON format to a file in the Application.persistentDataPath.

This makes it much easier to store your save games data in places like the Steam Cloud services, iCloud, Google Play or your own backend etc..

The Library also adds some functionality such as extra record types and more refined/flexible data access.

Keys are type specific, meaning that if you save a string under the key "key1" it will not be overridden when you save an int with the same key "key1";

Tested on Windows, OSX, Linux, iOS and Android.

Installation

Grab the latest unity package from the releases tab of this repo and import it into your project.

Save File Encryption

I have added a very simple scrambler to the saved json so that players cannot easily cheat by changing the values in the saved game file, this can be enabled or disabled with the bool ScrambleSaveData at the top of FileBasedPrefs.cs. Please note, this is not super secure encription, just a small string scrambler method that i think will stop 99.99% of users from editing the data. Adding your own better encryption would not be super difficult if you had the time.

At the top of the file FileBasedPrefs.cs is a string named "EncryptionCodeword". Please change that string to your own randomised value.

Once you choose a value and release a version of your software with it, you must stick with it or it will break all old save files that were created with the old value.

Save File Name

At the top of the script FileBasedPrefs.cs you can specify what name you would like your save file to have in the string SaveFileName.

Usage

Get and Set

`FileBasedPrefs.SetString(string key, string value);`

`FileBasedPrefs.GetString(string key, (optional)string defaultValue);`

`FileBasedPrefs.SetInt(string key, int value);`

`FileBasedPrefs.GetInt(string key, (optional)int defaultValue);`

`FileBasedPrefs.SetFloat(string key, float value);`

`FileBasedPrefs.GetFloat(string key, (optional)float defaultValue);`

`FileBasedPrefs.SetBool(string key, bool value);`

`FileBasedPrefs.GetBool(string key, (optional)bool defaultValue);`

Util Methods

`FileBasedPrefs.HasKey(String key);` will return true if there is any data type saved under the requested key

`FileBasedPrefs.HasKeyForString(String key);` will return true if there is a string saved under the requested key

`FileBasedPrefs.HasKeyForInt(String key);` will return true if there is a int saved under the requested key

`FileBasedPrefs.HasKeyForFloat(String key);` will return true if there is a float saved

under the requested key

FileBasedPrefs.HasKeyForBool(String key); will return true if there is a bool saved under the requested key

FileBasedPrefs.DeleteKey(String key); This will delete ALL data records saved under the key regardless of type

FileBasedPrefs.DeleteString(String key); This will delete any string recorded under the key

FileBasedPrefs.DeleteInt(String key); This will delete any int recorded under the key

FileBasedPrefs.DeleteFloat(String key); This will delete any float recorded under the key

FileBasedPrefs.DeleteBool(String key); This will delete any bool recorded under the key

FileBasedPrefs.DeleteAll(); // Deletes all records and replaces the save file with a blank one

Save File Helper Methods

FileBasedPrefs.ManualySave(); // see the Advanced Usage section for important details regarding optimisation.

FileBasedPrefs.GetSaveFileAsJson(); // returns the saved prefs as a json object in string format.

FileBasedPrefs.GetSaveFilePath(); // returns the full path to your save file.

FileBasedPrefs.OverwriteLocalSaveFile(string data); // overwrites the save file with whatever data you like. Warning, this will break the FileBasedPrefs methods if the data you save is not in the SaveFile json format.

Advanced Usage / Speed optimisation / Manual File writing

Unfortunately, encrypting a text file is pretty slow, no matter how you do it, so if you are using one of the Set methods during active gameplay, then you might notice some slowdown in fps.

To counteract this i have included some advanced features that give you more control of when and how the library saves/encrypts data.

At the top of FileBasedPrefs.cs is a bool named "AutoSaveData";

If set to true (the default setting), then every time a Set method is called, it will write and encrypt the save data immediately.

If set to false, then it will only save and encrypt the data to file when you call FileBasedPrefs.ManualySave();

This means that the save data is stored in memory until you specifically tell it to write the file. This is much faster and causes no performance issues.

Included in the plugin package is a script named "FileBasedPrefsQuitListener".

If you attach that script to a gameobject in your splash or main menu scene, then it will call ManualySave() whenever the game quits, pauses or goes in or out of focus.

This means that the data is written and encrypted at a time when a slight hit in fps will not be noticeable to the user.