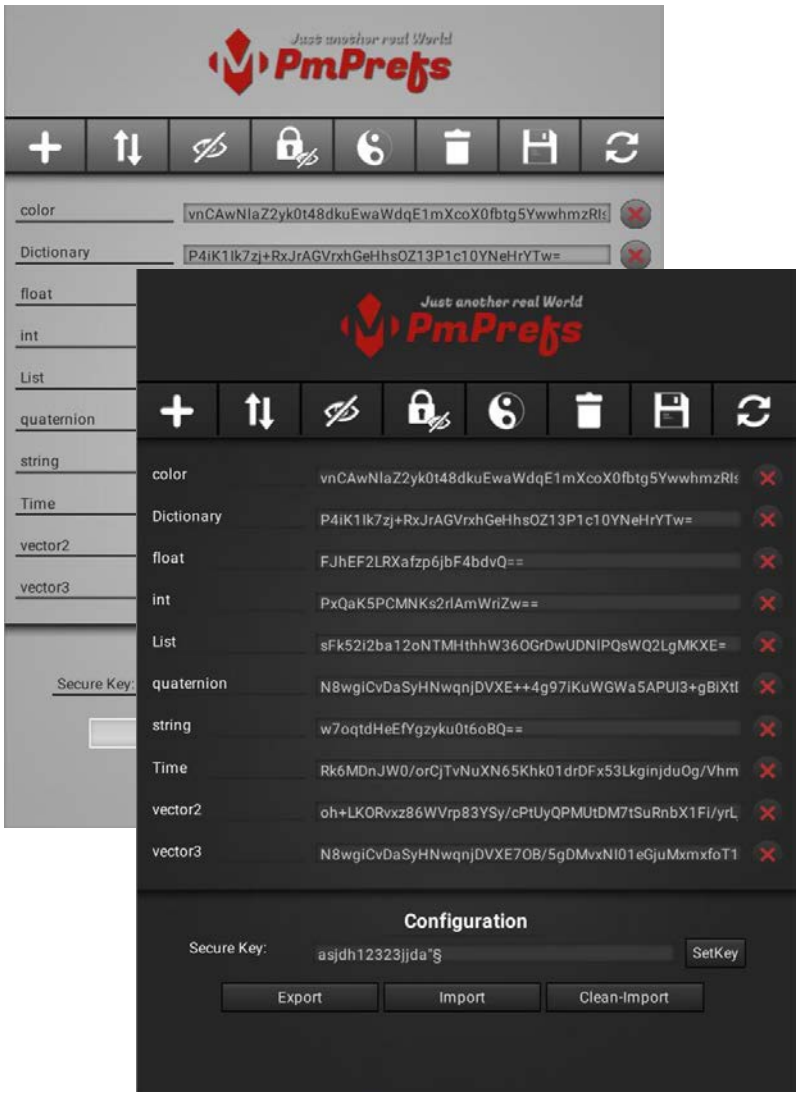
Just another real World

PmPrefs

Documentation

Copyright: 2018©, ProjectMakers
Autor: Daniel März



Content

Foreword

Save and load with PmPrefs as easy as never before.
Anyone who has ever had to save in Unity knows how complex that can be.
There are different methods here, one of which stands out very clearly.

Save with the Unity PlayerPrefs.
It's fast and compatible with every system.

There are a few beauties here.
- only 3 storable types (string, float, int)
- no display or listing of variables readily possible
- Values can only be adjusted via code
- Values are saved in plain text
- no export / import

This is where PmPrefs comes into play.

It has never been easier to save! Don‘t you think?

```
PmPrefs.Save("name", variable);
```

You can save all .NET objects!
So for example: Bools, lists, dictionaries, arrays, etc ...

But if the saving is so easy loading is certainly complex, right?
No, there is only one difference:

```
PmPrefs.Load<type>("name");
```

Seen? I agree!
You have to specify the type when loading, so for example: "int", or "List <string>".

And why is that soooo special?
Because PmPrefs continues to use Unity's PlayerPrefs, supporting all Untiy-supported systems.
That also without special rights on the respective terminal.

Why would you want to encrypt stored variables?
There are several reasons for that. For example, you can store passwords on the device without them being in plain text on the system.

Just imagine, someone writes an online banking app for Windows and would save the password with the PlayerPrefs, then this would be in the Window's registry in plain text. Not very good, right?

Another example would be the game development.
Suppose a game has a world ranking filled with earned gold (or some other point system). The player might come up with the idea of simply changing the value and getting an unfair advantage over the other honest players. This is particularly bitter, with an InGame payment system, and the paying players, rightly, get upset.

Exactly this is the case with the PmPrefs.

OK, that sounds good! But what about the export / import?
In every project comes the moment ... should he at least ..., in which one tests the functions.
Again, a practical example: In a role-playing game with 20 hours of playing time, the developer is just about to design the final boss and his reverend.
But you do not always want to start from the beginning, but create a memory stand and be able to load it again and again.
That's also useful when working on several things to change the state. So you can for example first load the “Endboss-Export” and then test it, then load the “TutorialEnde-Export” stand and continue working there, or testing.

That's a lot! How do I know how exactly everything works?
Do not worry, it is really very simple and after one or two times you have the shoot out without any documentation!

PmPrefs from ProjectMakers is currently the best way to save and be a must for any Unity project in Unity.

PS - Not only variables want to be saved, also your project should always be saved, but you will find something suitable here.

Explanation

This summary at a glance reflects the key elements of the asset

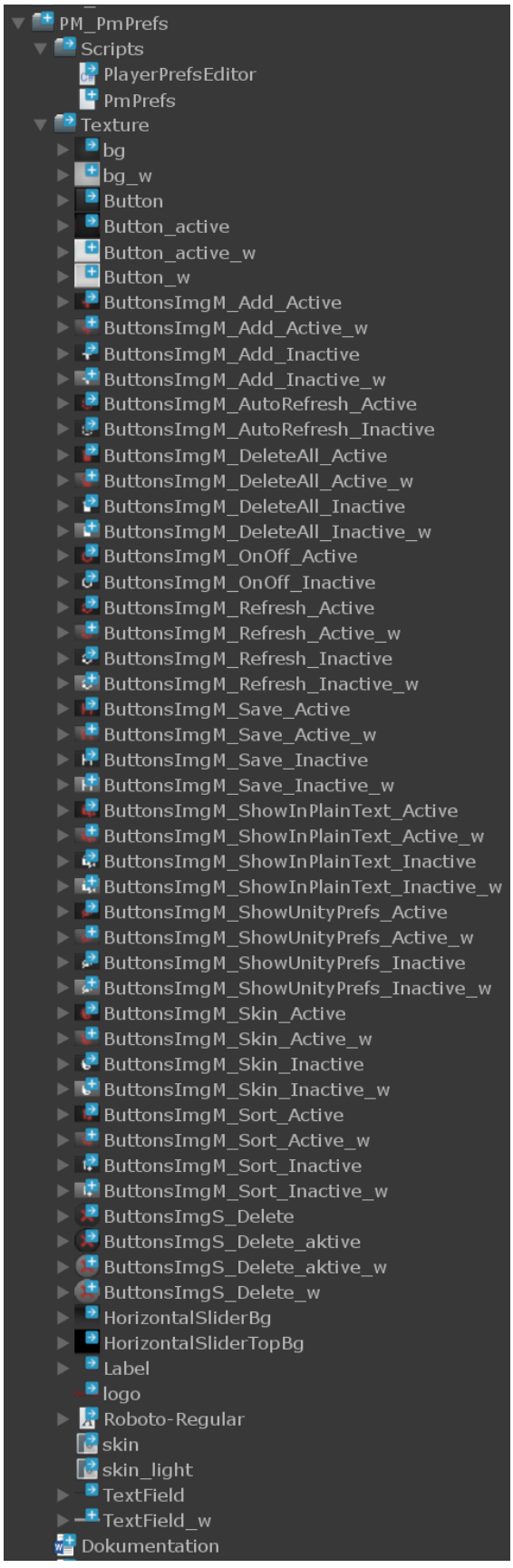
1. Create new value
2. Sort PmPrefs by name.
3. Show or hide Unity specific PlayerPrefs.
4. Show with or without encoding. (Regardless of whether the encryption is active or inactive).
5. Change the skin in dark or light.
6. Delete all PmPrefs? This cannot be undone.
7. Save all PmPrefs.
8. Reload all PmPrefs.
9. List of all PmPrefs (key).
10. List of all PmPrefs (value).
11. Delete this PmPref.
12. SecureKey to encrypt the all PmPrefs.
13. The encryption of all actually saved PmPrefs will be changed! This cannot be undone! (It's better to delete all PmPrefs before change the SecureKey).
14. Exports all currently displayed PmPrefs! Likewise, they are exported encrypted or decrypted, depending on how it is displayed in the PmPrefs Editor.
15. Imports PmPrefs data from a csv file. Important: the Import overwrites all existing PmPrefs and encrypts the new data with the new SecureKey (if a SecureKey is currently specified).
16. Imports PmPrefs data from a csv file. Important: the Clean-Import delete all existing data and also takes over the SecureKey from the import file (if there is one in the imported file, if none exists, the data will not be encrypted)

PmPref Methods

```
/* Save a .NET object as PmPrefs variable.
If is a SecureKey set, the variable was
encrypted saved. */
PmPrefs.Save("name", variable);

/* Load PmPrefs variable */
PmPrefs.Load<type>("name");

/* Delete all PmPref variables */
PmPrefs.DeleteAll();

/* Delete a PmPrefs variable */
PmPrefs.DeleteKey("name");

/* Save all PmPref variables */
PmPrefs.SaveAll();

/* Encrypt a string */
PmPrefs.Encrypt("string");

/* Decrypt a string */
PmPrefs.Decrypt("string");
```

Package Contents

The content of the asset consists of the following files in addition to the documentation.



Scripts

PlayerPrefsEditor.cs
PmPrefs.dll

Texture

bg.png
bg_w.png
Button.png
ButtonsImgM_Add_Active.png
ButtonsImgM_Add_Active_w.png
ButtonsImgM_Add_Inactive.png
ButtonsImgM_Add_Inactive_w.png
ButtonsImgM_AutoRefresh_Active.png
ButtonsImgM_AutoRefresh_Inactive.png
ButtonsImgM_DeleteAll_Active.png
ButtonsImgM_DeleteAll_Active_w.png
ButtonsImgM_DeleteAll_Inactive.png
ButtonsImgM_DeleteAll_Inactive_w.png
ButtonsImgM_OnOff_Active.png
ButtonsImgM_OnOff_Inactive.png
ButtonsImgM_Refresh_Active.png
ButtonsImgM_Refresh_Active_w.png
ButtonsImgM_Refresh_Inactive.png
ButtonsImgM_Refresh_Inactive_w.png
ButtonsImgM_Save_Active.png
ButtonsImgM_Save_Active_w.png
ButtonsImgM_Save_Inactive.png
ButtonsImgM_Save_Inactive_w.png
ButtonsImgM_ShowInPlainText_Active.png
ButtonsImgM_ShowInPlainText_Active_w.png
ButtonsImgM_ShowInPlainText_Inactive.png
ButtonsImgM_ShowInPlainText_Inactive_w.png
ButtonsImgM_ShowUnityPrefs_Active.png
ButtonsImgM_ShowUnityPrefs_Active_w.png
ButtonsImgM_ShowUnityPrefs_Inactive.png
ButtonsImgM_ShowUnityPrefs_Inactive_w.png
ButtonsImgM_Skin_Active.png
ButtonsImgM_Skin_Active_w.png
ButtonsImgM_Skin_Inactive.png
ButtonsImgM_Skin_Inactive_w.png
ButtonsImgM_Sort_Active.png
ButtonsImgM_Sort_Active_w.png
ButtonsImgM_Sort_Inactive.png
ButtonsImgM_Sort_Inactive_w.png
ButtonsImgS_Delete.png
ButtonsImgS_Delete_aktive.png
ButtonsImgS_Delete_aktive_w.png
ButtonsImgS_Delete_w.png
Button_active.png
Button_active_w.png
Button_w.png
HorizontalSliderBg.png
HorizontalSliderTopBg.png
Label.png
logo.png
Roboto-Regular.ttf
skin.guiskin
skin_light.guiskin
TextField.png
TextField_w.png

Glossary

PlayerPrefs

Stores and accesses player preferences between game sessions.
In Editor and Standalone.

On macOS PlayerPrefs are stored in
`~/Library/Preferences`
folder, in a file named
`unity.[company name].[product name].plist,`
where company and product names are the names set up in Project Settings.
The same .plist file is used for both Projects run in the Editor and standalone players.

On Windows, PlayerPrefs are stored in the registry under
`HKCU\Software\[company name]\[product name]`
key, where company and product names are the names set up in Project Settings.

On Linux, PlayerPrefs can be found in ~/.config/unity3d/[CompanyName]/[ProductName] again using the company and product names specified in the Project Settings.

On Windows Store Apps, Player Prefs can be found in
`%userprofile%\AppData\Local\Packages\ProductPackageId]>\LocalState\playerprefs.dat`

On Windows Phone 8, Player Prefs can be found in application's local folder, See Also:
`Directory.localFolder`

On Android data is stored (persisted) on the device. The data is saved in SharedPreferences. C#/JavaScript, Android Java and Native code can all access the PlayerPrefs data. The PlayerPrefs data is physically stored in
`/data/data/pkg-name/shared_prefs/pkg-name.xml.`

On WebGL, PlayerPrefs are stored using the browser's IndexedDB API.

On iOS, PlayerPrefs are stored in
`/Library/Preferences/[bundle identifier].plist.`

Unity

Unity Technologies SF is a video game development company, which is best known for the development of Unity, a licensed game engine.
It was founded in Denmark in 2004 as Over the Edge I/S, and became Unity Technologies ApS in 2006.
It moved its headquarters to San Francisco and became "Unity Technologies SF“ in 2009.

References

- ProjectMakers
  - AGB (German)
  - Impressum (German)
  - Datenschutz (German)
  - Contact
  - Discord
- Unity Technologies
- PlayerPrefs Documentation