PlayerPrefs

COMPLETE EASY-TO-USE SOLUTION

v.1.4

USER GUIDE

Developed by
Unity Plugins EU
www.unityplugins.eu

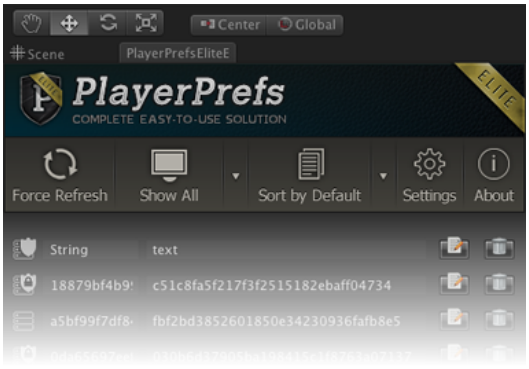
Welcome

PlayerPrefs Elite keep your data is human readable and protect them from modification and cheating. Supported operating systems: iOS, Android, Windows, Mac OS X, Windows Phone 8, BlackBerry, Windows Store (Metro), Unity Web Player, Linux.

PlayerPrefs Elite has an built-in **Secure Keys Manager** that keeps your secure keys in the scene and **Visual PlayerPrefs Editor**: a simple and user friendly tool for editing, saving, deleting and converting player preferences in Play or Edit mode in real time, using Unity Editor.

PlayerPrefs Elite is a lean, fast and elegant player preferences anti-hack protection, editing and visualising tool for **Unity Editor** on **Mac OS X** and **Windows** with an easy learning curve.

PlayerPrefs Elite: Light skin



PlayerPrefs Elite: Dark skin

You'll be glad you tried **PlayerPrefs Elite**, since you'll soon see how easy it is to pick up, and when you've gained more confidence, you'll realise that **PlayerPrefs Elite** is powerful enough to keep serving you. Hopefully you'll find it easy enough to use and learn (even if you're new to **Unity**), simple and uncluttered enough not to be annoying, **quick and efficient** enough to be a **pleasure to use, and powerful** enough to impress you!

PlayerPrefs Elite supports **Unity** & **Unity Pro** (works with **Unity 3, 4** & **Unity 5**).

Thank you for using **PlayerPrefs Elite**!

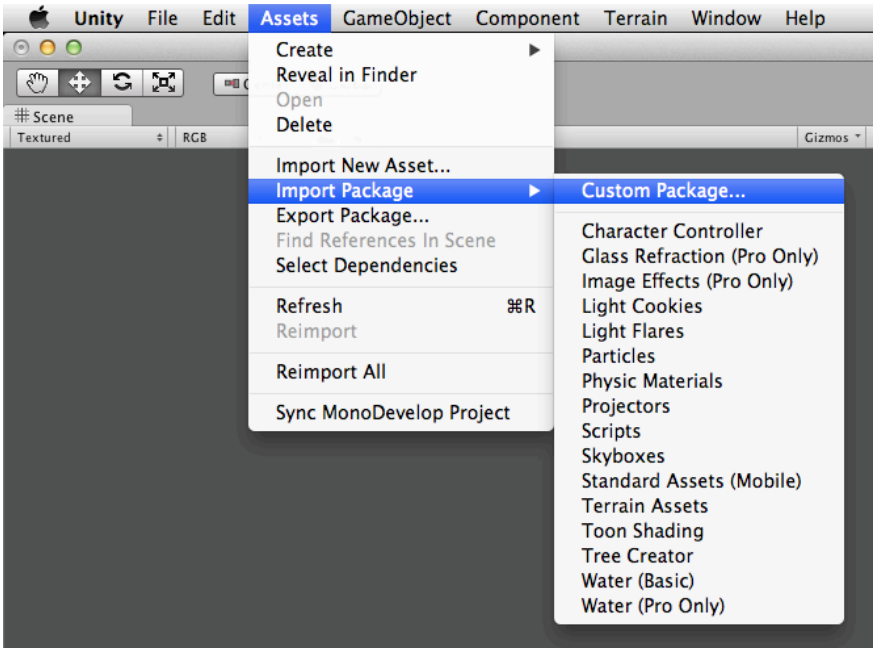
Table of Contents

Table of Contents

Quick start

Import package

To get started, import **PlayerPrefs Elite** package through *Assets->Import Package->Custom Package*

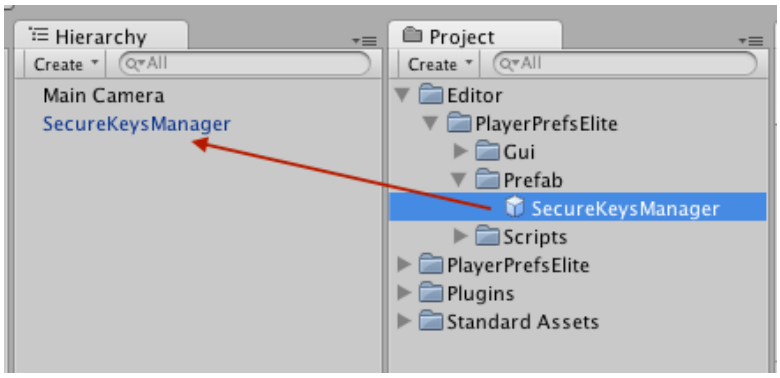


Or import **PlayerPrefsElite** package through *the Unity Asset Store Download Manager*.

Quick start

Setup Secure Keys Manager

Explore **Editor/ PlayerPrefsElite** / **Prefab** folder.
Select **SecureKeysManager** prefab in **Project** window and drag them to **Hierarchy** window.



Now you need to create a new key that will be used to protect data.

At least one key must be generated.
To generate your first key, press "**Generate new key**" button in Inspector window.

⚠ *Note*

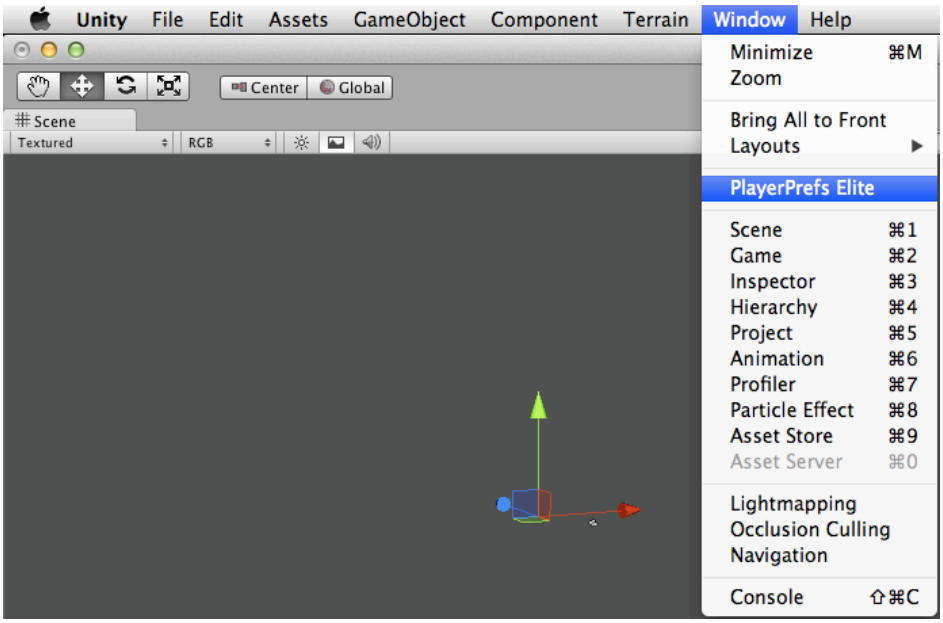
If you use identical names for the player preferences in various scenes of your project, keys in **Secure Keys Manager** should be the same.

Use the **save keys** and **import keys** buttons for quick and easy recovery of the key list in any scene of your project.

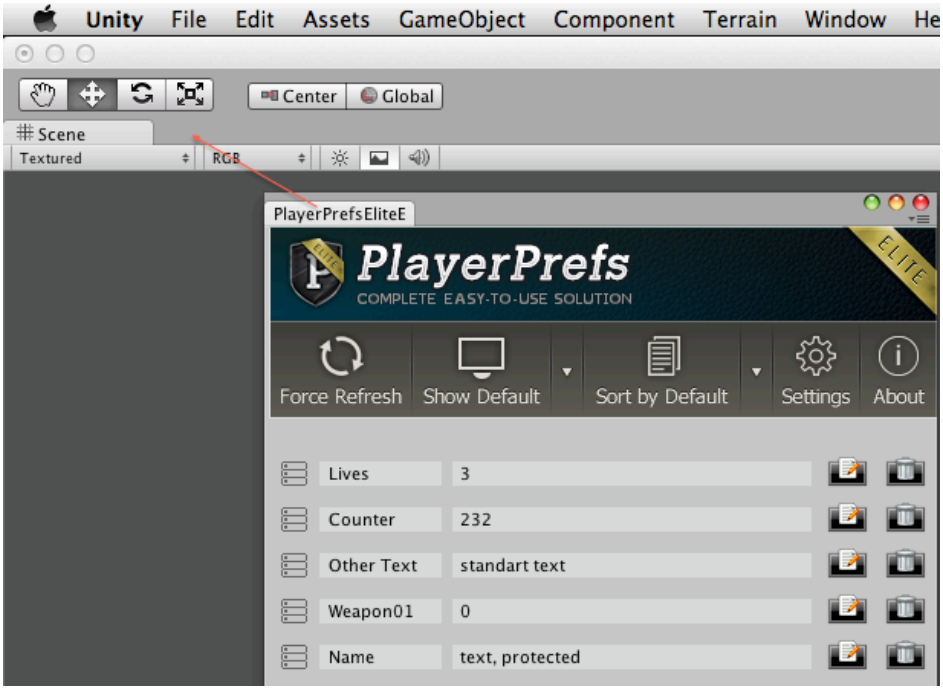
Quick start

Setup Visual PlayerPrefs Editor

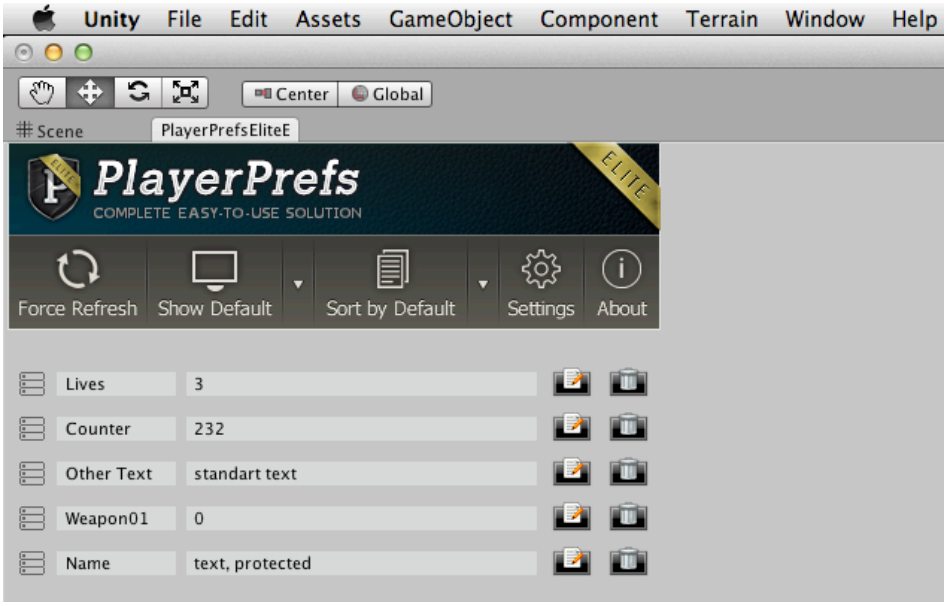
First, open **PlayerPrefs Elite** window through **Window** in title bar.



Now, drag and drop window where you want to keep window in the Unity Editor.



Quick start



PlayerPrefs Editor is ready to use.

Scripts

⚠ *Note*

By default the **Awake** function of different scripts are called in the order the scripts are loaded (which is arbitrary).

if you want call **PlayerPrefsElite** in **Awake** function, set higher execution order priority to **SecureKeysManager** (from Standard Assets), to make sure that secure keys is loaded become call **PlayerPrefsElite**.

http://docs.unity3d.com/Manual/class-ScriptExecution.html

Using **PlayerPrefs Elite** is very easy. You can use the *PlayerPrefsElite* as well as a standard *PlayerPrefs* method. The only difference is you can specify the number of the secret key from the **Secure Keys Manager**, if you want to use different keys.

Scripts

⚠ *Note*

PlayerPrefsElite.SetString (“Player Name”, “Foobar”);

Syntax is similar between **C#** and **JavaScript** (UnityScript).

Set the value as protected

PlayerPrefsElite.SetString

C#

```
public static void SetString (
     string key,
     string value,
     int secureKey
)
```

JavaScript

```
static function SetString (
   key: String,
   value: String,
   secureKey: int
): void
```

Description:

Set the value of the preference identified by *key* as protected.
secureKey is a number of the key in Secure Keys Manager (0 by default).

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetString (“Player Name”, “Foobar”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetString (“Player Name”, “Foobar”, 2);
```

Scripts

PlayerPrefsElite.SetInt

C#

```
public static void SetInt (
      string key,
      int value,
      int secureKey
)
```

JavaScript

```
static function SetInt (
   key: String,
   value: int,
   secureKey: int
): void
```

Description:

Set the value of the preference identified by *key* as protected.
secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetInt ("Player Score", 10);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetInt ("Player Score", 10, 2);
```

PlayerPrefsElite.SetFloat

C#

```
public static void SetFloat (
      string key,
      float value,
      int secureKey
)
```

JavaScript

```
static function SetFloat (
   key: String,
   value: float,
   secureKey: int
): void
```

Description:

Set the value of the preference identified by *key* as protected.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Scripts

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetFloat ("Player Score", 10.0);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetFloat ("Player Score", 10.0, 2);
```

Verify value

PlayerPrefsElite.VerifyString

C#

```
public static bool VerifyString (
     string key,
     int secureKey
)
```

JavaScript

```
static function VerifyString (
   key: String,
   secureKey: int
): boolean
```

Description:

Verify the protected value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyString ("PremiumSword");
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyString (“PremiumSword”, 2);
```

PlayerPrefsElite.VerifyInt

C#

```
public static bool VerifyInt (
     string key,
     int secureKey
)
```

JavaScript

```
static function VerifyInt (
   key: String,
   secureKey: int
): boolean
```

Description:

Verify the protected value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyInt (“Player Score”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyInt (“Player Score”, 2);
```

Scripts

PlayerPrefsElite.VerifyFloat

C#

```
public static bool VerifyFloat (
     string key,
     int secureKey
)
```

JavaScript

```
static function VerifyFloat (
   key: String,
   secureKey: int
): boolean
```

Description:

Verify the protected value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyFloat (“Player Score”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyFloat (“Player Score”, 2);
```

Scripts

Store as encrypted

PlayerPrefsElite.Encrypt

C#

```
public static void Encrypt (
      string key,
      string value,
      int secureKey
)
```

JavaScript

```
static function Encrypt (
   key: String,
   value: String,
   secureKey: int
): void
```

Description:

Store encrypted key and value.
secureKey is a number of the key in Secure Keys Manager (0 by default).

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.Encrypt ("Player Name", "Foobar");
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.Encrypt ("Player Name", "Foobar", 2);
```

Compare encrypted values

PlayerPrefsElite.CompareEncrypt

C#

```
public static bool CompareEncrypt(
      string key,
      string value,
      int secureKey
)
```

JavaScript

```
static function CompareEncrypt (
   key: String,
   value: String,
   secureKey: int
): boolean
```

Description:

Compare stored value identified by key and current value.
secureKey is a number of the key in Secure Keys Manager (0 by default).

Scripts

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.CompareEncrypt (“Player Name”, “Foobar”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.CompareEncrypt (“Player Name”, “Foobar”, 2);
```

Returns the value

For return the value use PlayerPrefsElite. GetString, GetInt, GetFloat or *standard PlayerPrefs method* as described in Unity documentation:
http://docs.unity3d.com/ScriptReference/PlayerPrefs.html

Return string value by key “Player Name”

```
PlayerPrefsElite.GetString (“Player Name”);

PlayerPrefs.GetString (“Player Name”);
```

Return int value by key “Player Score”

```
PlayerPrefsElite.GetInt (“Player Score”);

PlayerPrefs.GetInt (“Player Score”);
```

Return float value by key “Player Score”

```
PlayerPrefsElite.GetFloat (“Player Score”);

PlayerPrefs.GetFloat (“Player Score”);
```

Scripts

Save array to PlayerPrefs

⚠ *Note*

There are some things to keep in mind when editing array through **Visual PlayerPrefs Editor**.

Array values stored in PlayerPrefs separated by “|”
Example:
String1|String2|Sring3 etc.,

You can change value and resize array by “|” key runtime.

PlayerPrefs Elite support built-in arrays, Array and ArrayLists.

Built-in arrays (native .NET arrays), statically typed array which allows them to be edited in the inspector. Available in both JS and C#.

Array class is only available in Javascript and JavaScript PlayerPrefs Elite version.

ArrayLists, that are dynamic in size and allow add and remove items. Available in both JS and C#.

Built-in arrays are extremely fast and efficient and the best choice if you need the fastest performance especially for mobile devices.

Scripts

PlayerPrefsElite.SetStringArray

C# built-in array

```
public static void SetStringArray(
     string key,
     string[] value,
     int secureKey
)
```

JavaScript built-in array

```
static function SetStringArray (
   key: String,
   value: String[],
   secureKey: int
): void
```

C# ArrayList

```
public static void SetStringArray(
     string key,
     ArrayList value,
     int secureKey
)
```

JavaScript ArrayList

```
static function SetStringArray (
   key: String,
   value: ArrayList,
   secureKey: int
): void
```

JavaScript Array

```
static function SetStringArray (
key: String,
value: Array,
secureKey: int
): void
```

Description:

Save string array to PlayerPrefs.
secureKey is a number of the key in Secure Keys Manager (0 by default).

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetStringArray ("MyArray", MyArray);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetStringArray ("MyArray", MyArray, 2);
```

Scripts

PlayerPrefsElite.SetIntArray

C# built-in array

```
public static void SetIntArray (
      string key,
      int[] value,
      int secureKey
)
```

JavaScript built-in array

```
static function SetIntArray (
   key: String,
   value: int[],
   secureKey: int
): void
```

C# ArrayList

```
public static void SetIntArray (
      string key,
      ArrayList value,
      int secureKey
)
```

JavaScript ArrayList

```
static function SetIntArray (
   key: String,
   value: ArrayList,
   secureKey: int
): void
```

JavaScript Array

```
static function SetIntArray (
key: String,
value: Array,
secureKey: int
): void
```

Description:

Save int array to PlayerPrefs.
secureKey is a number of the key in Secure Keys Manager (0 by default).

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetIntArray ("MyArray", MyArray);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetIntArray ("MyArray", MyArray, 2);
```

Scripts

PlayerPrefsElite.SetFloatArray

C# built-in array

```
public static void SetFloatArray (
     string key,
     float[] value,
     int secureKey
)
```

JavaScript built-in array

```
static function SetFloatArray (
   key: String,
   value: float[],
   secureKey: int
): void
```

C# ArrayList

```
public static void SetFloatArray (
     string key,
     ArrayList value,
     int secureKey
)
```

JavaScript ArrayList

```
static function SetFloatArray (
   key: String,
   value: ArrayList,
   secureKey: int
): void
```

JavaScript Array

```
static function SetFloatArray (
key: String,
value: Array,
secureKey: int
): void
```

Description:

Save float array to PlayerPrefs.
secureKey is a number of the key in Secure Keys Manager (0 by default).

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetFloatArray ("MyArray", MyArray);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetFloatArray ("MyArray", MyArray, 2);
```

Scripts

Read array from PlayerPrefs

PlayerPrefsElite.GetStringArray

C#

```
public static string[] GetStringArray(
      string key,
)
```

JavaScript

```
static function GetStringArray (
   key: String,
): void
```

Description:

Read string array from PlayerPrefs.

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.GetStringArray ("MyArray");
```

PlayerPrefsElite.GetIntArray

C#

```
public static int[] GetIntArray (
   string key,
)
```

JavaScript

```
static function GetIntArray (
   key: String,
): void
```

Description:

Read int array from PlayerPrefs.

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.GetIntArray ("MyArray");
```

Scripts

PlayerPrefsElite.GetFloatArray

C#

```
public static float[] GetFloatArray (
    string key,
)
```

JavaScript

```
static function GetFloatArray (
   key: String,
): void
```

Description:

Read float array from PlayerPrefs.

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.GetFloatArray ("MyArray");
```

Verify array

PlayerPrefsElite.VerifyArray

C#

```
public static bool VerifyArray (
     string key,
     int secureKey
)
```

JavaScript

```
static function VerifyArray (
   key: String,
   secureKey: int
): boolean
```

Description:

Verify array value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Scripts

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyArray (“MyArray”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyArray (“MyArray”, 2);
```

Save Vector to PlayerPrefs

PlayerPrefsElite.SetVector2

C#

```
public static void SetVector2 (
     string key,
     Vector2 value,
     int secureKey
)
```

JavaScript

```
static function SetVector2 (
   key: String,
   value: Vector2,
   secureKey: int
): void
```

Description:

Set the value of the preference identified by *key* as protected.
secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetVector2 (“myVector2”, myVector2);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetVector2 (“myVector2”, myVector2, 2);
```

Scripts

PlayerPrefsElite.SetVector3

C#

```
public static void SetVector3 (
     string key,
     Vector3 value,
     int secureKey
)
```

JavaScript

```
static function SetVector3 (
   key: String,
   value: Vector3,
   secureKey: int
): void
```

Description:

Set the value of the preference identified by *key* as protected.
secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetVector3 (“myVector3”, myVector3);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetVector3 (“myVector3”, myVector3, 2);
```

PlayerPrefsElite.SetVector4

C#

```
public static void SetVector4 (
     string key,
     Vector4 value,
     int secureKey
)
```

JavaScript

```
static function SetVector4 (
   key: String,
   value: Vector4,
   secureKey: int
): void
```

Description:

Set the value of the preference identified by *key* as protected.
secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetVector4 (“myVector4”, myVector4);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetVector4 (“myVector4”, myVector4, 2);
```

Read Vector from PlayerPrefs

PlayerPrefsElite.GetVector2

C#

```
public static Vector2 GetVector2 (
      string key,
)
```

JavaScript

```
static function GetVector2 (
   key: String,
): void
```

Description:

Read Vector2 from PlayerPrefs.

Example:

```
PlayerPrefsElite.GetVector2 (“MyArray”);
```

Scripts

PlayerPrefsElite.GetVector3

C#

```
public static Vector3 GetVector3 (
      string key,
)
```

JavaScript

```
static function GetVector3 (
   key: String,
): void
```

Description:

Read Vector3 from PlayerPrefs.

Example:

```
PlayerPrefsElite.GetVector3 (“MyArray”);
```

PlayerPrefsElite.GetVector4

C#

```
public static Vector4 GetVector4 (
      string key,
)
```

JavaScript

```
static function GetVector4 (
   key: String,
): void
```

Description:

Read Vector4 from PlayerPrefs.

Example:

```
PlayerPrefsElite.GetVector4 (“MyArray”);
```

Scripts

Verify Vector

PlayerPrefsElite.VerifyVector2

C#

```
public static bool VerifyVector2 (
     string key,
     int secureKey
)
```

JavaScript

```
static function VerifyVector2 (
   key: String,
   secureKey: int
): boolean
```

Description:

Verify the protected value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyVector2 ("MyArray");
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyVector2 ("MyArray", 2);
```

Scripts

PlayerPrefsElite.VerifyVector3

C#

```
public static bool VerifyVector3 (
     string key,
     int secureKey
)
```

JavaScript

```
static function VerifyVector3 (
   key: String,
   secureKey: int
): boolean
```

Description:

Verify the protected value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyVector3 (“MyArray”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyVector3 (“MyArray”, 2);
```

Scripts

PlayerPrefsElite.VerifyVector4

C#

```
public static bool VerifyVector4 (
     string key,
     int secureKey
)
```

JavaScript

```
static function VerifyVector4 (
   key: String,
   secureKey: int
): boolean
```

Description:

Verify the protected value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyVector4 (“MyArray”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyVector4 (“MyArray”, 2);
```

Scripts

Save boolean to PlayerPrefs

⚠ *Note*

PlayerPrefsElite.SetBoolean use random numbers generation once is function called and keep your data safe and secure.

PlayerPrefsElite.SetBoolean

C#

```
public static void SetBoolean (
      string key,
      bool value,
      int secureKey
)
```

JavaScript

```
static function SetBoolean (
   key: String,
   value: boolean,
   secureKey: int
): void
```

Description:

Set the value of the preference identified by *key* as protected.
secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.SetBoolean (“Item1”, true);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.SetBoolean (“Item1”, true, 2);
```

Scripts

Read boolean from PlayerPrefs

PlayerPrefsElite.GetBoolean

C#

```
public static bool GetBoolean(
      string key,
)
```

JavaScript

```
static function GetBoolean (
   key: String,
): void
```

Description:

Read boolean from PlayerPrefs (return true or false).

Example:

```
PlayerPrefsElite.GetBoolean (“Item1”);
```

Scripts

Verify boolean

PlayerPrefsElite.VerifyBoolean

C#

```
public static bool VerifyBoolean (
      string key,
      int secureKey
)
```

JavaScript

```
static function VerifyBoolean(
   key: String,
   secureKey: int
): boolean
```

Description:

Verify the protected value.

Return *true* if value is not modified.
Return *false* if value modified by malicious software or user.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.VerifyBoolean ("Item1");
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.VerifyBoolean ("Item1", 2);
```

Scripts

Encryption and decryption

PlayerPrefsElite.EncryptString

C#

```
public static string EncryptString
(
     string key,
     int secureKey
)
```

Description:

Encrypt string.

Return encrypted value.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.EncryptString (“Item1”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.EncryptString (“Item1”, 2);
```

Scripts

Encryption and decryption

PlayerPrefsElite.DecryptString

C#

```
public static string DecryptString
(
     string key,
     int secureKey
)
```

Description:

Decrypt string.

Return decrypted value.

secureKey is a number of the key in Secure Keys Manager. (0 by default)

Example:

secureKey not set, 0 by default

```
PlayerPrefsElite.DecryptString (“Item1”);
```

secureKey is 2 (key 2 in Secure Keys Manager)

```
PlayerPrefsElite.DecryptString (“Item1”, 2);
```

Deep Explore

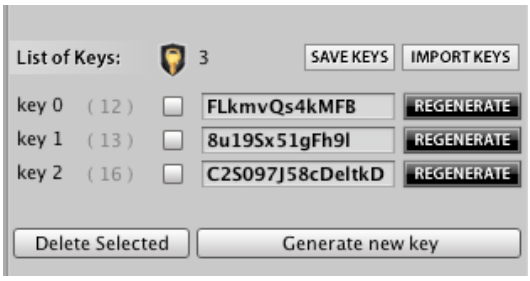
Secure Keys Manager

Secure Keys Manager keeps your secure keys in the scene.

Settings:

Show Alerts:
Show system warnings.

Key Length Min/Max value:
Range of key length when generating a new key or regeneration existing.



List of Keys:
The list of keys, key length (is in parentheses). The first key is always indexed as “0” (zero).

Save and Import Keys buttons:
For security reasons, the keys are not stored locally.
All keys stores and accesses through Unity Editor preferences based on the company name and product name. If you change the name of the company or project in PlayerSettings, you will need to resave the keys to be able to import into the scene again.

Regenerate button:
Regenerate the key.

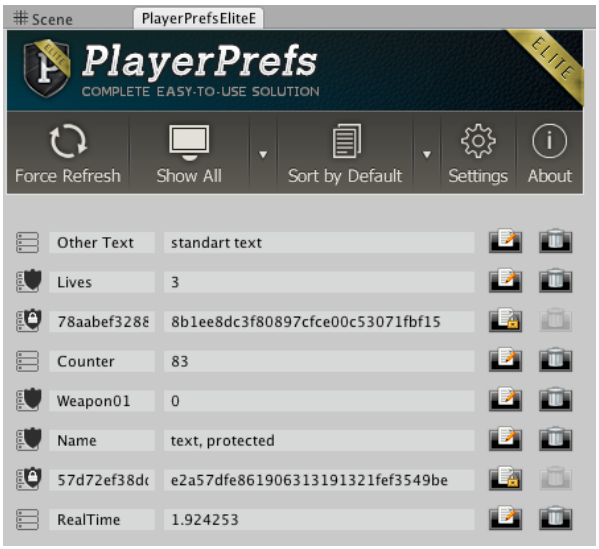
Delete Selected button:
Delete selected keys.

Generate new key button:
Generate a new key at the end of the list. Quantity of the keys in the list is unlimited.

Deep Explore

Visual PlayerPrefs Editor

Visual PlayerPrefs Editor is a simple and user friendly tool for editing, saving, deleting and converting player preferences in Play or Edit mode in real time, using Unity Editor. You can view or test your data for locked / unlocked content such as: premium items, virtual money, points, lives, etc. without changes in the script.



Note to Mac developers:
Due to specific storage method used for player preferences in **Mac OS X**, when you start the stage for the first time and write new names in player preferences, you need to wait 1-5s to update new records. If you create a new name in the player preferences inside gameplay, try to use **Force Refresh**.

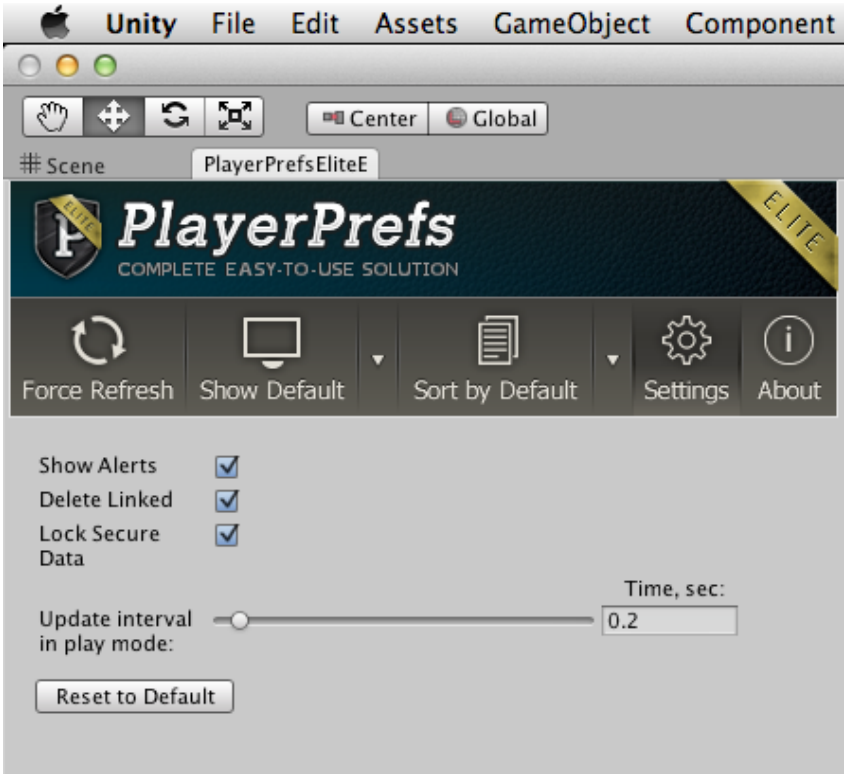
Force Refresh:
Try force refresh if new entries or values is not shown.

Show Default/All:
Show Default does not show system names and protected data.
Show All shows all without exception.

Sort by:
Sort by Default sorts as well as stored in player preferences.
Sort by A-Z sort list alphabetically.
Sort by Z-A sort list reversed.

Deep Explore

PlayerPrefs Editor Settings:



Show alerts - Show PlayerPrefs Elite system warnings.
Delete Linked deletes related records if the data is protected.
Lock Secure prohibits to deleting and editing protected data.
Update interval in play mode. A high value - increases performance, low - increases the number of updates per second.
Delete All remove all keys and values stored in player preferences.

Visual PlayerPrefs Editor: understanding buttons & icons

 - standard key and value

 - protected value

 - secured data

 - edit value

 - edit features is locked

 - delete key

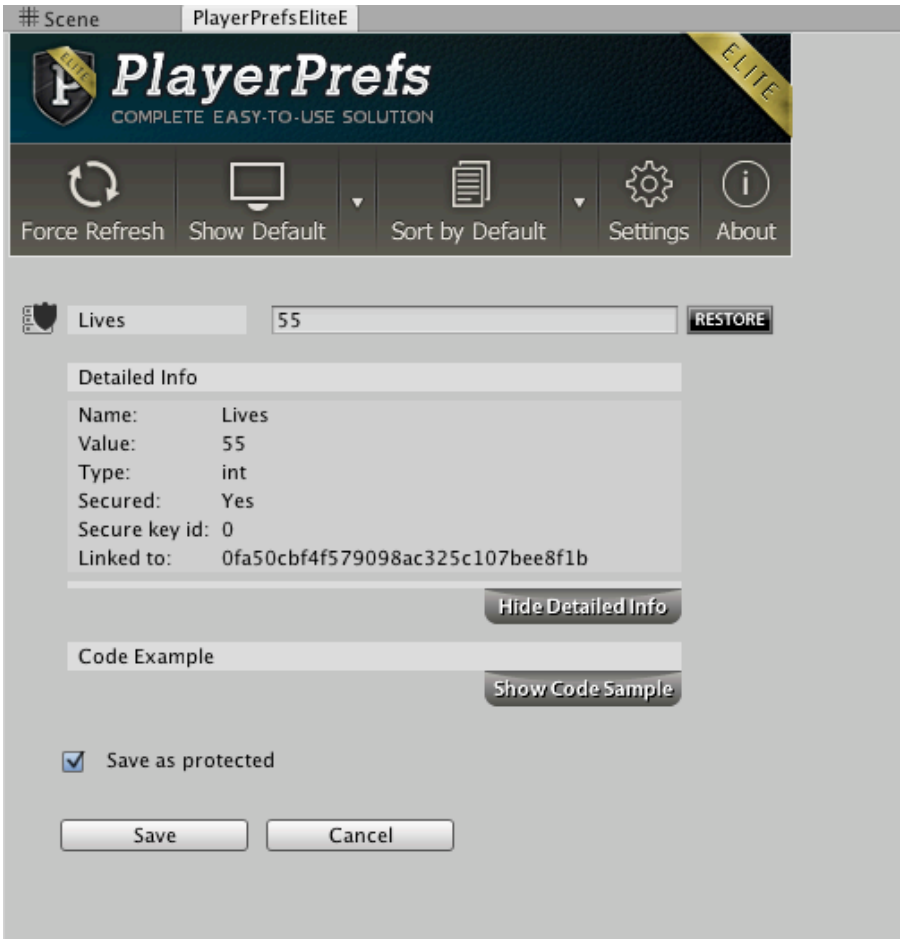
 - edit features is not available

 - delete key is not available

Deep Explore

Visual PlayerPrefs Editor: edit value

To edit existing value, push “Edit value” button ( ).



Restore button - return actual value to the key.

Detailed Info **-** info, which may be useful in developing process.

Name - key name

Value - actual value information

Type - type of value

Secured - shows, if value for the key is protected

Secure key id - key number in Secure Key Manager; used for protect value

Linked to - show linked key name

Deep Explore

Visual PlayerPrefs Editor: code example

To view code example, press “**Show Code Sample**” button in “**Edit value**” mode.
Code is generated based on **key name** and **type**.

“**Copy**” button will copy code example to clipboard.

If this value is protected, checkbox “**Save as protected**” is on.
Once you set checkbox “**Save as protected**” turned off and save it as standard value, verification test will fail (very useful for testing).

Script Examples

Additional Script Hints

Is a good idea to combine and save different values on the same level.

Use different values

One of examples - always keep different values to unlock content.

JavaScript

```
var Weapon01: int = 0; // this is your premium weapon, 0 is locked
```

C#

```
int Weapon01 = 0; // this is your premium weapon, 0 is locked
```

*When the purchase is made, you change Weapon01 to a different value a and save in player preferences. Do not set "1" :-) , use a **random** value!*

C# & JavaScript

```
if (PurchaseSuccessful) {
   // purchase successful
   // set random number from 100 to 10000
   Weapon01 = Random.Range(100, 10000);

   // save it to player preferences.
   PlayerPrefsElite.SetInt ("Weapon01", Weapon01);
}
```

First you load the value from the player preferences:

```
Weapon01 = PlayerPrefs.GetInt("Weapon01");
```

Script Examples

Inside the game you check, Weapon01 unlocked or not, and check it for hacking / cheating and etc.:

C# & JavaScript

```
if (Weapon01 != 0) {
   // weapon01 is unlocked
   // Verify the protected value
   If (!PlayerPrefsElite.VerifyInt("Weapon01")){
      // return false, Weapon01 is modified
      Weapon01 = 0; // return, break etc., from this point
   }
}
```

or like this:

C# & JavaScript

```
if (Weapon01 != 0 && PlayerPrefsElite.VerifyInt("Weapon01")) {
   // well done
   // Weapon01 is unlocked and VerifyInt return true
}
```

Combine values

For example, you have a lot of variables inside the game and you can tie them to unlock objects, virtual money and so forth.

Create a chain of variables and save them using **PlayerPrefsElite**.

Store encrypted variables (scores + lives):

C# & JavaScript

```
PlayerPrefsElite.Encrypt ("Combined", scores.ToString()+lives.ToString());
```

Compare and verify this values on the fly:

C# & JavaScript

```
if(!PlayerPrefsElite.CompareEncrypt("Combined", scores.ToString()+lives.ToString())){
   // return false, modified value detected
}
```

This method compare stored values (score+lives) by key "Combined" and current value (score+lives).

You can try different variants, selecting those that are better suited to your project.

Feel free modify or create custom methods based on existing, or ask new.

Special Note

All keys must be the same in your project, if you use same PlayerPrefs names in different scenes.

When you're ready to release your project, please save your secure keys. If you lose them and do update with new keys, users who has made an update, will not pass verification.

Export Compliance

If you submit your app to Apple App Store, Google Play, Windows Store etc., you must answer questions about Export Compliance (cryptography).

PlayerPrefs Elite use md hash, its message digest algorithms and this restriction should not apply. Select "No".

Encryption and decryption using custom algorithm and this restriction should not apply. Select "No".

Upgrade

From version 1.2 PlayerPrefs Elite use C# by default.
If you have an earlier version, remove **PlayerPrefsElite.js** and **SecureKeysManager.js** in **/Standard Assets/PlayerPrefsElite/** folder.

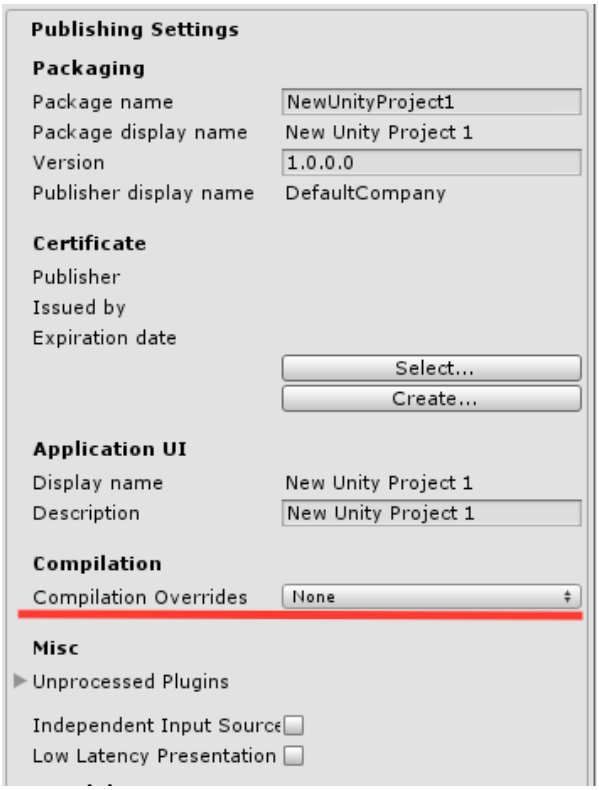
JavaScript version available in /PlayerPrefsElite/JavaScriptVersion/ folder.

If you prefer using the JavaScript version, follow this instruction:

1. Remove PlayerPrefsElite.cs and SecureKeysManager.cs files in **/Standard Assets/PlayerPrefsElite/** folder.
2. Import JavaScriptVersion.unitypackage asset.

Windows Store

To access C# classes from JS or Boo, you need to set **Compilation Overrides** to None in PlayerSettings, Publishing Settings.



If your project is C# only, you can set **Compilation Overrides** to None or Use Net Core.

Support / Contact info

Unity Plugins EU
custom plugin development

Web site:
www.UnityPlugins.eu

Customer support:
support@unityplugins.eu

Please note, that we require 2-4 business days for processing email support requests.
For email support we'll require you to provide **invoice number** of your purchase.
Also, don't forget to add the info about your **Unity engine** and **OS version** to your message (for example: Unity Pro 5.1.1f4, Windows 8.1).

Support language: english, russian.

Designed by:
ExpertDesigner.eu