EZ ENTERTAINMENT

EZ DATA MANAGER

Owner's Manual

v 2.0

Thank you for buying our asset and for supporting its further development. This plugin was created to extend the functionality of Unity's native system. Should you need help, find issues or have any suggestions, don't hesitate to send us a message at support@ezentertainment.eu

Please read the quick setup guide before you start using this asset.

Solutions – Ez Data Manager solves the following problems

- Every game or app uses global variables and having them in different classes can become a pain to manage. Ez Data Manager can help you with that by giving you a central location for all of your settings and referenced types such as prefabs/gameobjects/textures/materials/sprites/etc.
- Having a lot of variables of different types, visible in the inspector, is hard to work with. That is why we designed a very well thought out custom inspector that tries to give you (the developer) a very convenient way to add/delete/sort and change variables.
- You can create a lot of variable types, aven arrays and lists. They will be public and you will be able to update them with ease, from any script.
- Ez Data manager can automatically handle the saving and loading of variables to/from Player Preferences. Supported types are int, float, bool, string and also arrays and lists of the same types.
- If sensitive data is being saved/loaded, there is an option to secure it through encryption.
- Save yourself some development time and get started right now!

Quick Setup Guide

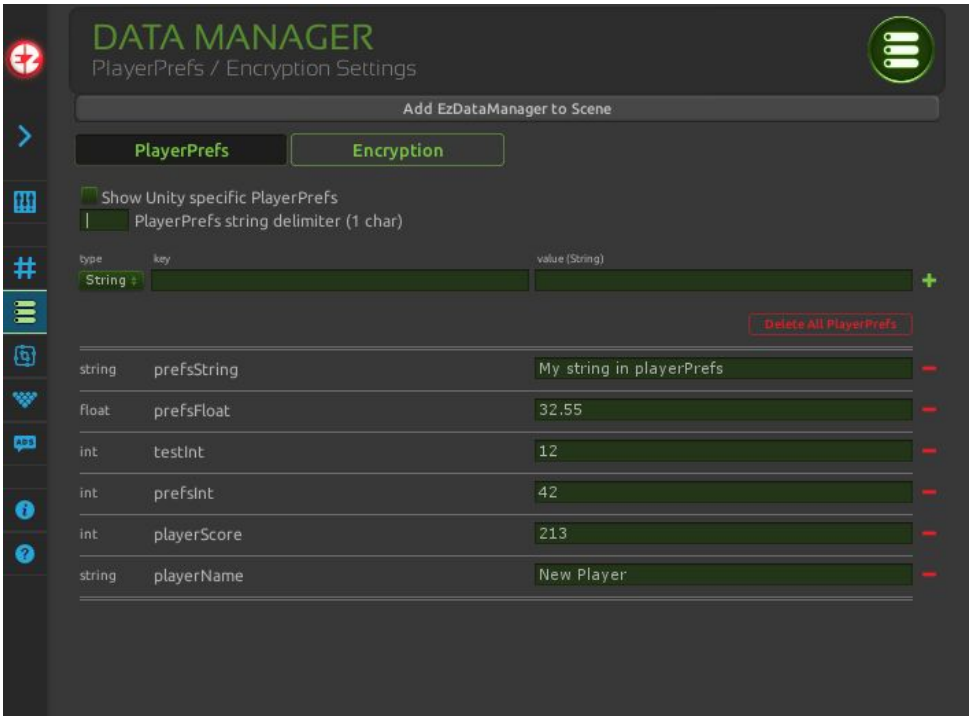
1. Import Ez Data Manager (from @UnityAssetStore)
2. In the top toolbar → Tools → Ez → Control Panel
3. Click on “Data Manager”
4. Click on “Add Ez Data Manager to Scene”
5. Done!

Watch the introduction and tutorial videos on our YouTube channel:
https://www.youtube.com/playlist?list=PLRE6VXhDQg2Ma8wVOoYgCf5mxIVvWyxUd

Ez Data Manager Window

The Ez Data Manager Window can be opened from the Editor toolbar Ez → Control Panel. In the Control Panel window, select “Data Manager”. Using it you can add or remove Ez Data Manager to and from your currently opened scene.
The EzDataManager GameObject is a singleton and should be added only in your main scene (or start scene) as it will persist across scenes.

PlayerPrefs Tab



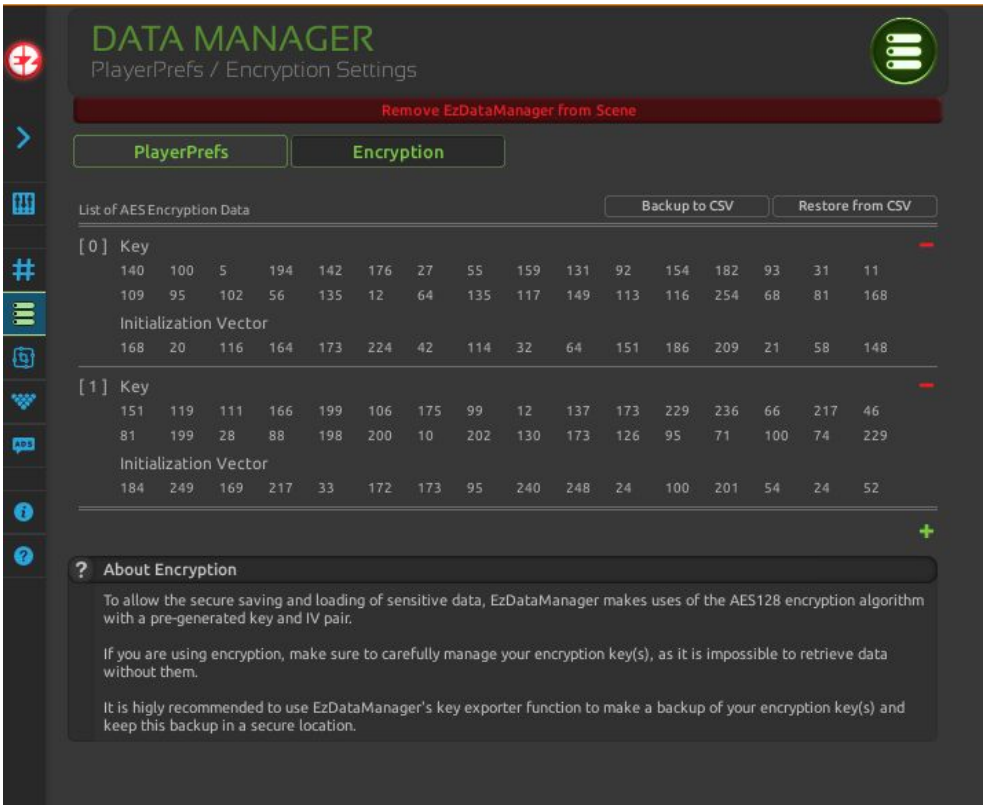
This is the “PlayerPrefs” tab in the Ez Data Manager Window.

Here you can view/edit/add/delete keys and values in Player Preferences.

- Show Unity specific PlayerPrefs - will make keys and values automatically created by Unity visible and editable.

- PlayerPrefs string delimiter - this character will be used as a string delimiter when saving an array or a list in PlayerPrefs. Please make sure that this is a character that does NOT appear in the data you want to save.
- You can add new data to PlayerPrefs by selecting the type (int, float or string), entering a key, a value and by pressing the ‘+’ button or the ‘Enter’ on the keyboard.

Encryption Tab



This is the "Encryption" tab in the Ez Data Manager Window.

Here you can create or delete key/IV pairs for AES encryption, back them up or restore them from backup. You only need to configure this if you intend to secure saved data through encryption.

Only one key/IV pair is generally enough, but you can configure as many as needed.

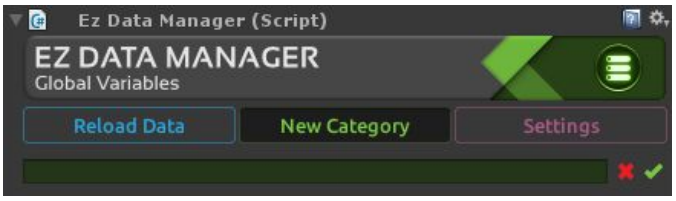
- To create a new key/IV pair, press the green '+' button.
- To delete a key/IV pair click the '-' next to it
- To backup or restore encryption keys/IVs, use the provided buttons.

WARNING: If a key/IV pair used for encryption is lost, any data that was encrypted using that pair will be irrevocably lost, as well! It is highly recommended have a backup of your key(s).

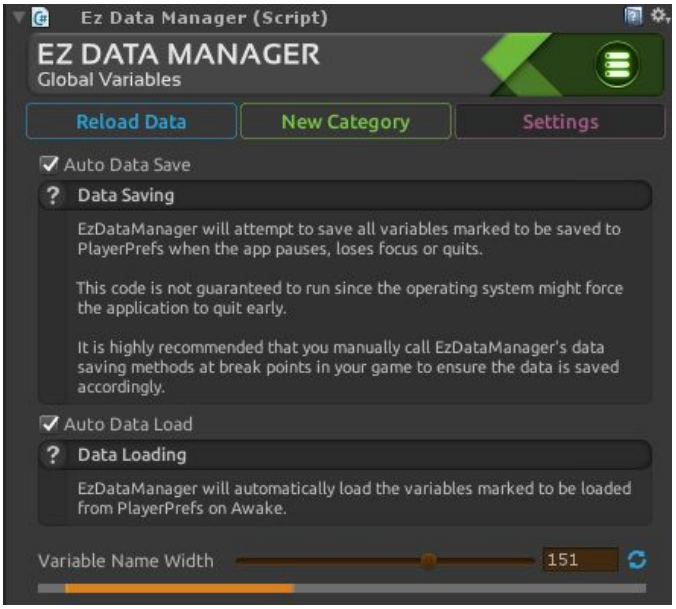
WARNING: If you try to encrypt a null or empty value, an exception will be thrown!

Ez Data Manager Component

When you select the EzDataManager for the first time, you will not have any Categories or variables available and you will have to create them.

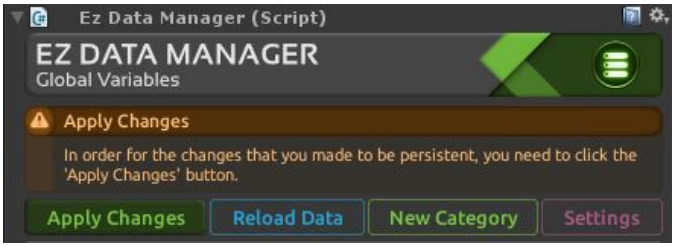


New Category: allows you to create a new category with any name you want. This will help you sort your variables in different categories.

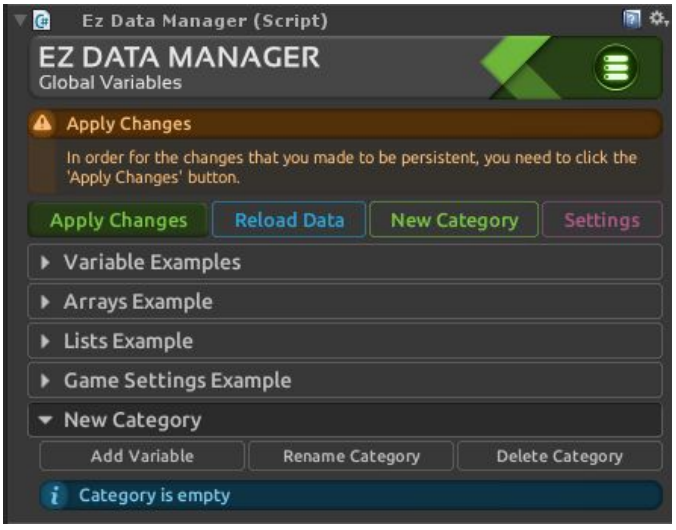


Settings:

- Auto Data Save - EzDataManager will attempt to save all variables marked to be saved to PlayerPrefs when the app pauses, loses focus or quits. This data saving cannot be guaranteed to run since the operating system might force the application to quit early.
- Auto Data Load - EzDataManager will automatically load the variables marked to be loaded from PlayerPrefs on Awake.
- Allows you to adjust the variable name's width with the help of a slider. This option is available to help you read long variable names.



Apply Changes: after any structural change (you add/delete a new category or a new variable, etc.) you will have to click 'apply changes' to make them persistent. This button appears any time you change the database structure.

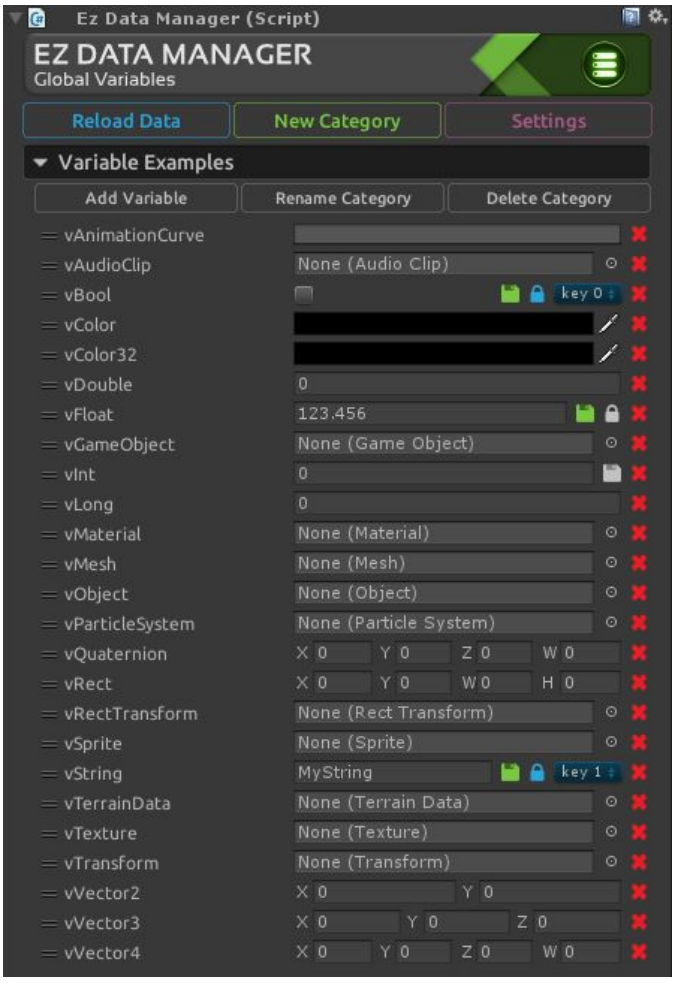


Add Variable: allows you to add a new variable to the target category.

Rename Category: allows you to rename the target category

Delete Category: deletes the target category and all the variables it contains.

Variables



Here you can see how every variable type is drawn in the custom inspector.

Variables can be reordered simply by drag-and-drop.

If the variable's type is supported for PlayerPrefs saving/loading, you can enable this using the disk button.
The following variable types are supported for data saving/loading to/from Player Preferences:

- Int
- Float
- Bool
- String

If saving and loading are enabled for a variable, you can also secure certain sensitive variables using encryption using the lock button.
If saving and loading are enabled for a variable, you can also secure certain sensitive information by enabling encryption through the lock button.

Finally, variables can be deleted by pressing the [X] button.

Arrays

Here you can see how the arrays are drawn in the custom inspector.

Notice that you see the variable name on the left and that you see the type on the show/hide bar (eg.: Color[2]). You also see how many items does an array hold.

Arrays can be reordered simply by drag-and-drop.

If the array's type is supported for PlayerPrefs saving/loading, you can enable this using the disk button.
The following array types are supported for data saving/loading to/from Player Preferences:

- Int[]
- Float[]
- Bool[]
- String[]

If saving and loading are enabled for an array, you can also secure certain sensitive information by enabling encryption through the lock button.
If an array is marked for encryption, you need to select one of the previously configured encryption keys to be used.

Finally, arrays can be deleted by pressing the [X] button.

Lists

Here you can see how the lists are drawn in the custom inspector.

Notice that you see the variable name on the left and that you see the type on the show/hide bar (eg.: List<bool>[1]). You also see how many items does a list hold.

Lists can be reordered simply by drag-and-drop.

If the list's type is supported for PlayerPrefs saving/loading, you can enable this using the disk button.
The following List types are supported for data saving/loading to/from Player Preferences:

- List<int>
- List<float>
- List<bool>
- List<string>

If saving and loading are enabled for a list, you can also secure certain sensitive information by enabling encryption through the lock button.
If a list is marked for encryption, you need to select one of the previously configured encryption keys to be used.

Finally, lists can be deleted by pressing the [X] button.

Ez Data Manager Code

Usings

To use the variables, remember to add:

```
using Ez.DataManager;
```

Accessing Variables, Arrays and Lists

To access any global variable

```
EzDataManager.Instance.variableName;
```

To access any global array

```
EzDataManager.Instance.arrayName;
```

To access any global list

```
EzDataManager.Instance.listName;
```

Saving and Loading data to/from Player Preferences

The following types of variables under EzDataManager can be configured to be automatically saved/loaded to/from PlayerPrefs, as follows:

- int - saved as int
- int[] and List<int> - saved as string
- float - saved as float
- float[] and List<float> - saved as string
- bool, bool[] and List<bool> - saved as string
- string, string[] and List<string> - saved as string

Note: All variables marked for encryption will be saved as strings.

WARNING: If a variable/array/list marked for encryption is null or empty, an exception will be thrown when attempting to encrypt it and data saving will fail! Always make sure there is a stored value in order for encryption to work!

Save All to PlayerPrefs - EzDataManager Helper Method

To save all variables, arrays and lists with data saving/loading enabled to PlayerPrefs

```
EzDataManager.SaveAllToPlayerPrefs();
```

Load All from PlayerPrefs - EzDataManager Helper Method

To load from PlayerPrefs all variables, arrays and lists with data saving/loading enabled

```
EzDataManager.LoadAllFromPlayerPrefs();
```

Save Category to PlayerPrefs - EzDataManager Helper Method

To save all variables, arrays and lists with data saving/loading enabled under a specified category to PlayerPrefs

```
EzDataManager.SaveCategoryToPlayerPrefs(string categoryName);
```

Load All from PlayerPrefs - EzDataManager Helper Method

To load from PlayerPrefs all variables, arrays and lists with data saving/loading enabled under a specified category

```
EzDataManager.LoadCategoryFromPlayerPrefs(string categoryName);
```

Manually Saving and Loading Individual Variables to/from Playerprefs

We recommend using the helper methods described above as a simple, reliable and fast method to save or load EzDataManager's variables. However, if needed, it is possible to work with individual variables, even with variables not kept by EzDataManager.

WARNING: If you try to encrypt a null or empty value, an exception will be thrown!

Usings

In order to save and load individual variables, use the following 'using' in your script(s):

```
using Ez.DataManager.Prefs;
```

Manually Saving Unencrypted Variables to PlayerPrefs

To make saving data easier, EzDataManager extends supported types with a Save() method that allows a quick saving of variables to PlayerPrefs under a specified key. The extension method works the same way for all supported types:

```
variable.Save(string prefsKey); // Saves this variable to PlayerPrefs under the specified key
```

Usage Examples:

```
int myInt = 5;
myInt.Save("exampleInt");

int[] myIntArray = new int[] { 2, 3, 4 };
myIntArray.Save("exampleIntArray");

List<int> myIntList = new List<int>() { 101, 102, 103 };
myIntList.Save("exampleIntList");

float myFloat = 1.2f;
myFloat.Save("exampleFloat");

float[] myFloatArray = new float[5];
myFloat.Save("exampleFloatArray");

List<float> myFloatList = new List<float>();
myFloatList.Save("exampleFloatList");

bool myBool = false;
myBool.Save("exampleBool");

bool[] myBoolArray = new bool[] { false, true, false, true };
myBoolArray.Save("exampleBoolArray");

List<bool> myBoolList = new List<bool>() { false, false, true };
myBoolList.Save("exampleBoolList");

string myString = "test";
myString.Save("exampleString");

string[] myStringArray = new string[] { "", "Unity3D", "Scene", "!!!" };
myStringArray.Save("exampleStringArray");

List<string> myStringList = new List<string>(myStringArray);
myStringList.Save("exampleStringList");
```

Manually Saving Encrypted Variables to PlayerPrefs

To be able to encrypt variables prior to saving them, you must first configure at least one encryption key/IV pair in EzDataManager's custom window (top menu → Tools → Ez → Control Panel). If multiple key/IV pairs are configured, you can select which one to be used when encrypting.

To encrypt a variable an extension .Encrypt() method is available for supported types.

```
variable.Encrypt([int keyIndex = 0]); // Encrypts this variable using the key/IV pair at index
                                      // and returns the encrypted value as a byte[]
```

Calling .Encrypt().Save() saves the encrypted value of variable to PlayerPrefs. All encrypted values are saved as string values in PlayerPrefs.
Usage examples:

```
int myInt = 5;
myInt.Encrypt().Save("exampleInt");                     // Encrypted with the default key - key 0

int[] myIntArray = new int[] { 2, 3, 4 };
myIntArray.Encrypt(2).Save("exampleIntArray");          // Encrypted with the third key - key 2

List<int> myIntList = new List<int>() { 101, 102, 103 };
myIntList.Encrypt(0).Save("exampleIntList");            // Encrypted with the first key - key 0

float myFloat = 1.2f;
myFloat.Encrypt(1).Save("exampleFloat");                // Encrypted with the second key - key 1

float[] myFloatArray = new float[5];
myFloat.Encrypt().Save("exampleFloatArray");            // Encrypted with the default key - key 0

List<float> myFloatList = new List<float>();
myFloatList.Encrypt().Save("exampleFloatList");         // Encrypted with the default key - key 0

bool myBool = false;
myBool.Encrypt().Save("exampleBool");                   // Encrypted with the default key - key 0

bool[] myBoolArray = new bool[] { false, true, false, true };
myBoolArray.Encrypt().Save("exampleBoolArray");         // Encrypted with the default key - key 0

List<bool> myBoolList = new List<bool>() { false, false, true };
myBoolList.Encrypt().Save("exampleBoolList");           // Encrypted with the default key - key 0

string myString = "test";
myString.Encrypt().Save("exampleString");               // Encrypted with the default key - key 0

string[] myStringArray = new string[] { "", "Unity3D", "Scene", "!!!" };
myStringArray.Encrypt().Save("exampleStringArray");     // Encrypted with the default key - key 0

List<string> myStringList = new List<string>(myStringArray);
myStringList.Encrypt().Save("exampleStringList");       // Encrypted with the default key - key 0
```

Important note: make sure to use a valid key index, otherwise an exception is thrown. Key count starts at 0.

Manually Loading Unencrypted Variables from PlayerPrefs

To easily load unencrypted variable data from PlayerPrefs for supported data types, an extension .Load() method can be used:

```
variable.Load(string prefsKey, [bool encrypted = false], [int keyIndex =0]);

// Attempts to load and return the value stored in PlayerPrefs under the specified key.
// If nothing is found, returns the variable's value.
```

Usage examples:

```
int myInt = 5.Load("myExampleInt");
 // myInt = 5 if no value is found for "myExampleInt" in PlayerPrefs

int[] myIntArray = new int[] { 2, 3, 4 };
   myIntArray = myIntArray.Load("exampleIntArray");
// myIntArray = myIntArray if no value is found for "exampleIntArray" in PlayerPrefs

List<int> myIntList = new List<int>() { 101, 102, 103 };
   myIntList = myIntList.Load("exampleIntList");
// myIntList = myIntList if no value is found for "exampleIntList" in PlayerPrefs


float myFloat = 1.2f.Load("exampleFloat");
   // myFloat = 1.2f if no value is found for "exampleFloat" in PlayerPrefs


float[] myFloatArray = new float[5].Load("exampleFloatArray");
     // myFloatArray = a new float[5] if no value is found for "exampleFloatArray" in PlayerPrefs


List<float> myFloatList = new List<float>();
   myFloatList = myFloatList.Load("exampleFloatList");
// myFloatList = myFloatList if no value is found for "exampleFloatList" in PlayerPrefs

bool myBool = true.Load("exampleBool");
  // myBool = true if no value is found for "exampleBool" in PlayerPrefs

bool[] myBoolArray = new bool[] { false, true, false, true };
   myBoolArray = myBoolArray.Load("exampleBoolArray");
// myBoolArray = myBoolArray if no value is found for "exampleBoolArray" in PlayerPrefs

List<bool> myBoolList = new List<bool>() { false, false, true };
   myBoolList = myBoolList.Load("exampleBoolList");
// myBoolList = myBoolList if no value is found for "exampleBoolList" in PlayerPrefs

string myString = "test";
   myString = myString.Load("exampleString");
// myString = mystring if no value is found for "exampleString" in PlayerPrefs

string[] myStringArray = new string[] { "", "Unity3D", "Scene", "!!!" };
   myStringArray = myStringArray.Load("exampleStringArray");
// myStringArray = myStringArray if no value is found for "exampleStringArray" in PlayerPrefs

List<string> myStringList = new List<string>(myStringArray).Load("exampleStringList");
          // myStringList = new List(myStringArray) if no value is found for "exampleStringList" in
PlayerPrefs
```

Manually Loading Encrypted Variables from PlayerPrefs

To easily load encrypted variable data from PlayerPrefs for supported data types, the .Load() extension method can be used with additional parameters:

```
variable.Load(string prefsKey, true, [int keyIndex = 0]);

// Attempts to load and return the value stored in PlayerPrefs under the specified key.
// If nothing is found, returns the variable's value.
// To decrypt the stored value, the "encrypted" parameter must be passed as true.
// keyIndex specified the key/IV pair to be used for decryption. By default the first pair is used (0)
```

Important note: make sure to use a valid key index, otherwise an exception is thrown. Key count starts at 0. Decryption must be performed with the same key/IV pair that was used for encryption, otherwise the encrypted data is NOT retrievable.

Usage examples:

```
int myInt = 5.Load("myExampleInt", true);
 // myInt = 5 if no value is found for "myExampleInt" in PlayerPrefs OR decryption fails with key 0
```

```
int[] myIntArray = new int[] { 2, 3, 4 };
   myIntArray = myIntArray.Load("exampleIntArray", true, 2);
// myIntArray = myIntArray if no value is found for "exampleIntArray" in PlayerPrefs OR decryption
fails with key 2

List<int> myIntList = new List<int>() { 101, 102, 103 };
   myIntList = myIntList.Load("exampleIntList", true);
// myIntList = myIntList if no value is found for "exampleIntList" in PlayerPrefs OR decryption fails
with key 0


float myFloat = 1.2f.Load("exampleFloat", true, 1);
   // myFloat = 1.2f if no value is found for "exampleFloat" in PlayerPrefs OR decryption fails with
key 1

float[] myFloatArray = new float[5].Load("exampleFloatArray", true);
     // myFloatArray = a new float[5] if no value is found for "exampleFloatArray" in PlayerPrefs OR
decryption fails with key 0

List<float> myFloatList = new List<float>();
   myFloatList = myFloatList.Load("exampleFloatList", true);
// myFloatList = myFloatList if no value is found for "exampleFloatList" in PlayerPrefs OR decryption
fails with key 0

bool myBool = true.Load("exampleBool", true);
  // myBool = true if no value is found for "exampleBool" in PlayerPrefs OR decryption fails with key 0

bool[] myBoolArray = new bool[] { false, true, false, true };
   myBoolArray = myBoolArray.Load("exampleBoolArray", true);
// myBoolArray = myBoolArray if no value is found for "exampleBoolArray" in PlayerPrefs OR decryption
fails with key 0

List<bool> myBoolList = new List<bool>() { false, false, true };
   myBoolList = myBoolList.Load("exampleBoolList", true);
// myBoolList = myBoolList if no value is found for "exampleBoolList" in PlayerPrefs OR decryption
fails with key 0

string myString = "test";
   myString = myString.Load("exampleString", true);
// myString = mystring if no value is found for "exampleString" in PlayerPrefs OR decryption fails with
key 0

string[] myStringArray = new string[] { "", "Unity3D", "Scene", "!!!" };
   myStringArray = myStringArray.Load("exampleStringArray", true);
// myStringArray = myStringArray if no value is found for "exampleStringArray" in PlayerPrefs OR
decryption fails with key 0

List<string> myStringList = new List<string>(myStringArray).Load("exampleStringList", true);
          // myStringList = new List(myStringArray) if no value is found for "exampleStringList" in
PlayerPrefs OR decryption fails with key 0
```