# Ablage der PlayerPrefs

Speichert die aktuelle Spielkonfiguration

Speicherort in Windows: Registry Key HKEY\_CURRENT\_USER\Software\DefaultCompany\SaveLoad

# Handling des UnitySerializers

Alle zu serialisierenden Gameobjekte müssen das Script *StoreInformation.cs* angehängt haben.  
Das Gameobjekt erhält so zum einen eine Unique ID und zum anderen werden damit.

Alle Kinder eines Objektes müssen dieses Script auch angehängt haben.

## Zu beachten

* Zur Runtime hinzugefügte Elemente werden nur gesichert wenn bei der Runtime die StoreInformation.cs – Component hinzugefügt wird.  
  -> Cube erscheint dann rosa?

1. How UnitySerializer Works

This page is NOT complete.

UnitySerializer is based on my SilverlightSerializer – modified to work with Unity components.

It would have been possible to build this serializer around BinaryFormatter – however, due to the need for injection into the serialization process to handle certain Unity components I chose to base it off my existing work which has been included in many different packages and downloaded directly more than 4,000 times.

There are many ways to extend the serialization process to handle other component types, this page will be the source of documentation for those.

**Where is the data stored?**

The data is stored in different places depending on the technique you use.  If you use the complete Save/Load features of LevelSerializer then the data is stored in PlayerPrefs in a string key called \_Save\_Game\_Data\_.  Checkpoints are stored based on the current player’s name and are held in keys in PlayerPrefs called [PlayerName]\_RESUME\_

You can store data anywhere you like if you use the base LevelSerializer.SerializeLevel and .LoadSavedLevel which return and take a string respectively.

**What does Unity Serializer save in my own scripts?**

By default Unity Serializer saves all of the public properties and fields of your script.

Neue Components die während der Laufzeit hinzugefügt werden, werden nicht gesichert. -> Müsste man dann wohl noch von Hand nachtragen. Grund hier wohl das Inventar, dass man am Anfang über den SaveGameManager machen muss.

Unity Serializer: Gute Ansätze, aber im 2013 nicht mehr weiterentwickelt.  
Mit dem neuen Unity nicht vollends kompatibel. Ab Unity 4.3 GUI Kompatibilität verloren.

Problem im Moment, dass ganzer Level neu geladen wird auf default Zustand.  
Problem möglicherweise, dass kein RootObjekt angegeben wird?

-> API lesen, was das genau macht

UnitySerializer warns you that you need to have a SaveGameManager in each scene.

**Funktionen**

//Positionen werde nicht gesichert - wohl wirklick nur ein "Level-Speichern", damit man weiss auf welchem Level man war

LevelSerializer.SerializeLevelToFile("Test");

LevelSerializer.LoadSavedLevelFromFile("Test");

ebenfalls gleiches Verhalten mit

LevelSerializer.SerializeLevel()

**Anyway:**

**Save & Load Individual Objects**

**Damit könnten wir einzelne serializieren rsp. vorher alle Gameobjekte in Liste tun, diese Serialisieren und dann zurückholen.  
Funktioniert im Moment mi Spielfigur nicht.**

LevelSerializer and JSONLevelSerializer now contain commands to save any object tree.  Use:

 LevelSerializer.SaveObjectTree(someGameObject)

If the specified game object has a UniqueIdentifier or other serializer script then the object and all of its children that have scripts will be save.

If it does not then it will be saved using the new EmptyObjectIdentifier and so will any of its children that aren’t currently being saved.

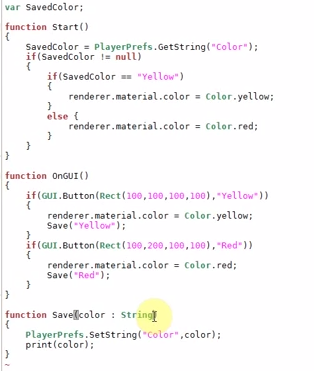
If you want to flag all objects in a tree (all children and other descendants) for serialization using the new empty object format (for example the root object in SaveObjectTree is saved, but some of its children aren’t and need to be) you can use EmptyObjectIdentifier.FlagAll(someGameObject) as mentioned in the previous section.

To load back in an object without delaying the current game or reloading the scene you can use:

LevelSerializer.LoadObjectTree(data)

**Daten speichern mit PlayerPrefs:**

On Windows standalone players, PlayerPrefs are stored in the registry under HKCUSoftware[company name][product name] key, where company and product names are the names set up in Project Settings.



Speichern des Spiels: Serialisierung der Objekte

* Serialisierung nach XML, JSON oder in ein binary File:  
  Es stehen einem die UnityKomponenten im Weg
  + ***XML:***InvalidOperationException: To be XML serializable, types which inherit from IEnumerable must have an implementation of Add(System.Object) at all levels of their inheritance hierarchy. UnityEngine.Transform does not implement Add(System.Object).
  + C# Serialiazation (binary):  
    SerializationException: Type UnityEngine.GameObject is not marked as Serializable
* Unity Community empfiehlt:

UnitySerializer http://whydoidoit.com/unityserializer/getting-started-guide/

**Probleme beim Speichern overall:**Zustände von Methoden / StateMachines / etc. wie speichern?

**Alternative1:**Selber ein Writer schreiben, der sich die entsprechenden GameObjekte holt,  
deren Variablen ausliest und für das entsprechende Objekt speichert.

Beim Laden sucht man das GameObjekt und setzt die Variablen wieder.  
„First Spawn a new GameObject and find the proper element in the XML for that GameObject”.

**Alternative2:  
Save Game Unity Serializer**http://u3d.as/content/ixion-software/save-game-json-binary/33x

**Official Tutorials by Unity**:

Ab 27:00 über Serialisierung. Er handelt aber nur Klassen die keine GameObject Membervariablen haben.

http://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/persistence-data-saving-loading

http://docs.unity3d.com/ScriptReference/SerializeField.html

“GameObject is not a base type and cannot be serialized. You cannot save GameObject outside of Unity.”

**The Serialization Problem**  
There are some problems with serialization in Unity. They are not necessarily the problems you might expect either:

1. A GameObject does not have a Stable ID
2. If you’ve used Coroutines, how can you save and restore their state
3. The logistics of serializing a GameObject and some or all of its components

Okay that last item is probably not a great surprise but the first two are possibly unique to Unity.

Unity can serialize the following types:

* All basic data types (like int, string, float, and bool).
* Some built-in types (including Vector2, Vector3, Vector4, Quaternion,Matrix4x4, Color, Rect, and LayerMask).
* All classes inheriting from UnityEngine.Object (including GameObject,Component, MonoBehavior, Texture2D, and AnimationClip).
* Enums.
* Arrays and lists of a serializable type.

**Some Serialization Rules**

* Avoid structs
* Classes you want to be serializable need to be marked with [Serializable]
* Public fields are serialized (so long as they reference a [Serializable] class)
* Private fields are serialized under some circumstances (editor).
* Mark private fields as [SerializeField] if you wish them to be serialized.
* [NonSerialized] exists for fields that you do not want to serialize.

SerializationException: Type UnityEngine.GameObject is not marked as Serializable.

That means UnityEngine.GameObject is not marked with the [Serializable attribute](http://msdn.microsoft.com/en-us/library/system.serializableattribute(v=vs.110).aspx), which is a requirement (just because Unity says a type is serializable doesn't mean it's serializable with *all*serialization engines; in this case, it probably means that the type is serializable with Unity's engine, not the one that is part of the .NET BCL).