Guardian

Anti Piracy Protection

version: 1.1

Setting up the class to be used in your game is simple and can be done in a few steps.
First you have to import the package to your project, once the package has been imported
you can use it from all supported Unity Scripts.

If we look at the Tester.cs source that was included in the package,
we can see how it works.

We have to setup the guardian for use in our application with the Init function
Guardian.Init(uint[] BaseKeys);
the Base keys are a collection of 4 uints that can be setup in your script as such

```
public class Tester : MonoBehaviour {

	public uint[] MyBaseKeys = { 8, 16, 32, 64 };

	void Start () {
		// Init Guardian
		Guardian.Init(MyBaseKeys);
	}
}
```

In this instance the BaseKeys are named MyBaseKeys and are given 4 numbers that makes the serial unique to your product/game.
By changing these numbers, you will render the serials generated previously invalid. Which opens new options for version controlling the valid keys from 1.x to 2.x if users have to upgrade.

Onces this is done you can generate a serial number by calling the method(s)
Guardian.Generate(uint seed);
Guardian.Generate(string seed);

as following the example

```
public class Tester : MonoBehaviour {

	public uint[] MyBaseKeys = { 8, 16, 32, 64 };

	void Start () {
		// Init Guardian
		Guardian.Init(MyBaseKeys);
		// Generate and Print key
		Debug.Log(Guardian.Generate("Test"));
	}
}
```

That would generate a serial looking something like this:
7SYC6BKWO3QVJVSRN5ANMJCGQLLC5GHVIEDQ

Now this serial is good but it could become better or at least more readable, if the user is reading it out with ease would should add some dashes to the long serial.

This can be done by adding a space variable in the Guardian Class by using this method
byte Guardian.Spacing;

```
public class Tester : MonoBehaviour {

        public uint[] MyBaseKeys = { 8, 16, 32, 64 };

        void Start () {
                // Init Guardian
                Guardian.Init(MyBaseKeys);
                // Set dashes between every 6 character
                Guardian.Spacing = 6;
                // Generate and Print key
                Debug.Log(Guardian.Generate("Test"));
        }
}
```

So it will look more like this:
7SYC6B-KWO3QV-JVSRN5-ANMJCG-QLLC5G-HVIEDQ

You also have the option to generate multiple serials which can be handy in the final stage of release to generate a list of serials and store them locally.
You can do this by calling the method:

```
IDictionary<uint, string> Generate (uint numberOfKeys, System.Random random);
```

Which can be used as following:

```
var MySerialNumbers = Guardian.Generate(10, new System.Random(10));
foreach(var serial in MySerialNumbers) {
   Debug.Log(serial.Value);
}
```

Which will generate 10 unique serial numbers, that you can write to a local file using the System.IO which you can then later reference for purchase orders and such.

....

Now one serial has been generated.
This serial is now acceptable in your application using the BaseKeys
but we will have to check the serial to see if it's correct or now, some kind of verification method has to be implimented as well.

This can be done with a simple boolean check.

```
public class Tester : MonoBehaviour {

        public byte CheckKey = 0;
        public uint[] MyBaseKeys = { 8, 16, 32, 64 };

        void Start () {
                // Init Guardian
                Guardian.Init(MyBaseKeys);
                // Set dashes between every 6 character
                Guardian.Spacing = 6;

                // Generate and store key in a string
                string MySerial = Guardian.Generate("Test");

                // Check if key is valid
                if(Guardian.ValidateKey(Serial, CheckKey, MyBaseKeys[CheckKey])) {
                    Debug.log("The serial key is valid");
                } else {
                    Debug.log("The serial key is invalid");
                }

        }
}
```

Here i have used the Method
bool Guardian.ValidateKey(string key, int subkeyIndex, uint subkeyBase)

this will return true if the serial number matches the basekey provided.
You have to give it a serial, a subkeyIndex between (0,3) and then number of the base key we are verifing against which can be done as such MyBaseKeys[0].

In the abow example we generate a key from the string test in to a string called MySerial, once the serial key has been generated we check if that key is valid.
By using the Guardian.ValidateKey method.
This will return a true if valid or a false if they serial key is invalid.

A good tip would be to store the key using playerprefs onces it has been validated so when the application starts up again at a later point we just load the serial number from playerprefs and check it in the background instead of having the player re-enter the serial again.

You can even look at the source code of the guardian and modify or extend it to your need if so is needed.
I would take kindly to any ideas for extension and am happy to offer support.
You can reach me at SperoSophia@gmail.com, please set the topic of the mail as "Guardian Support" so i know what product you are seeking assistance of.