Tiny Save API

Hello and welcome.

Let me start off by saying thank you for your download, and I hope you find the asset fulfils your coding needs.

This is my go-to code to persist data locally. I've packaged it up, added even MORE comments (I'm a fan of commenting even "obvious" code), added a Prefab Manager and a demonstration scene, all to make life as easy as possible for you.

The API contains one save method, two load methods, as well as a public encrypt and a public decrypt method.

The API saves to a local file in the persistent data path, unless the application is a WebGL running application. In the case of WebGL, PlayerPrefs is used as the storage because it uses the browser's IndexedDB API[1].

The format of the serialised data can be either binary, plain JSON, or encrypted JSON to make the JSON unreadable to humans.

[1] See the Unity Documentation on PlayerPrefs for more information.

Serialisation Types

Binary Serialisation

Binary Serialisation allows for more data types to be serialised[2]. Using the Binary Serialiser option (default), allows the developer to serialise just the data object, instead of a whole class if desired, e.g. A single Dictionary<Tk, Tv> can be serialised by itself.

Because the data is serialised into a binary, object specific format, it's by default much harder to gain access to sensitive data.

Json Serialisation

Internally, JSON serialisation uses the Unity serializer; therefore, the object you pass in must be supported by the serializer: it must be a MonoBehaviour, ScriptableObject, or plain class/struct with the Serializable attribute applied[3]. Because of this, care must be taken to make sure that all data types are those that the Unity Serialiser can understand

Json Serialisation is best used when data is being transferred from remote sources (such as a remote server), or working with data from other sources, such as Google Sheets. It's human readable by default, which must be taken into account if the data is sensitive. If data security is a concern, the Json text can be encrypted first, using the internal encryption methods. In those cases, selecting the JsonEncryted SerializationType will perform the encryption and subsequent decryption for you.

[2] See the Microsoft Documentation for more information on Binary Serialization.
[3] Unity Serialisation information can be found here.

Installation

The Tiny Save API requires you to set your “Api Compatibility Level” to either “.NET 4.x” or “.NET Standard 2.0” (these might be named slightly different in older Unity versions, for example 2018.4 calls it “.NET 4.x Equivalent)”). You can find this under the Edit menu > Project Settings > Player section. This pertains mostly to older Unity versions, as the newer Unity versions don’t allow the older API options anymore.

Note: The static class contains a static TinySave.SecurityKey property which is the password you can use to encrypt JSON data to an unreadable form. Before using the API, the TinySave.SecurityKey value should be changed. This can be done either directly in the TinySave class, or by using the TinySaveManager. Please be aware that any JSON that has been encrypted and saved with a particular SecurityKey, will need that same string SecurityKey to unencrypt the data. It’s best then to set the security key early in development then leave it untouched for the remainder of the project. This only applies if you’re using JsonEncrypted as your TinySave. SerializationType. Also, because of the Triple DES encryption used, the key length should be either 16 or 24 characters in length. If using the TinySaveManager, the Inspector checks for string validity, and either pads strings that are too short, or truncates them if they’re too long.

Usage

The easiest way to save and load data is with the following two lines:

```
var saveResult = await TinySave.SaveAsync(name, data, SerializationType.Binary);

var loadResult = await TinySave.LoadAsync<T>(name, SerializationType.Binary);
```

In the above code, T represents a data structure, such as a class. For example:

```
[Serializable]
public class PlayerData
{
  public string PlayerName;
  public int LivesLeft;
  public int Score;
}
```

Demonstration

A test of Tiny Save API abilities can be viewed by opening the *TinySaveTestSceneSimple* scene. When the game runs, the Game view shows two buttons and a text area. The simplest way to run the test is to press the buttons in sequence and see the results in the right hand UI panel.

For more advanced features, check the *TinySaveTestSceneAdvanced* sene. The scene contains a *TinySaveManager* Prefab that has the *TinySaveManagerComponent* attached. The *TinySaveManager* helps the developer set the default serialisation type and security key.

TinySave.SaveAsync

```
Task<Response> SaveAsync ( string name, object obj, SerializationType
serializationOverride = SerializationType.None )
```

This method saves the data supplied in the object parameter. `name` is the name used as either the filename, or the key for the PlayerPrefs. `obj` is the object to be serialised. `serializationOverride` is optional and is used if the developer wishes to override the default `TinySave.SerializationType`.

The method returns a `Response` enum value indicating the method success, or failure, as well as the newly created object of type T.

The general way to call this method is:

```
var saveResult = await TinySave.SaveAsync ( name, dataObject );
```

TinySave.LoadAsync

```
Task<(Response, T)> LoadAsync<T> ( string name, SerializationType
serializationOverride = SerializationType.None )
```

This method will load a class of struct from either a file or, in the case of WebGL, the PlayerPrefs. `name` is the name used as either the filename, or the key for the PlayerPrefs. `serializationOverride` is optional and is used if the developer wishes to override the default `TinySave.SerializationType`.

The method returns a Tuple indicating the method success, of failure, as well as the newly created object of type T.

The general way to call this method is:

```
var loadResult = await TinySave.LoadAsync<Class> ( name );
```

```
Task<Response> LoadAsync<T> ( string name, T item, SerializationType
serializationOverride = SerializationType.None )
```

This method will load the Json data and overwrite a pre-existing **MonoBehaviour** or **ScriptableObject**. The data is read from either a file, or in the case of WebGL, the PlayerPrefs. `name` is the name used as either the filename, or the key for the PlayerPrefs. `serializationOverride` is optional and is used if the developer wishes to override the default `TinySave.SerializationType`, but must still be either `Json` or `JsonEncrypted`.

The method returns a `Response` enum value indicating the method success, or failure. The object of type T passed in has its data overwritten with that of the Json content.

The general way to call this method is:

```
var loadResult = await TinySave.LoadAsync<TinySaveTester> ( name, myComponent,
SerializationType.Json );
```

TinySave.Encrypt

```
byte [ ] Encrypt ( string dataString, string securityKeyOverride = null )
```

This is a publicly exposed method to take a string and return an encrypted byte array. It's used internally by the API but can also be used by the developer to encrypt any strings for security.

TinySave.Decrypt

```
string Decrypt ( byte [ ] dataBytesArray, string securityKeyOverride = null )
```

This is a publicly exposed method to take an encrypted byte array and returns an unencrypted string. It's used internally by the API but can also be used by the developer to decrypt any secure strings (as byte arrays).

Versions

1.0 [04-12-2019]
Initial Publication