SIMPLE API - DOCS

WHAT IS SIMPLEAPI?

SimpleAPI is a set of static methods that are made to help you with your HTTP requests and to streamline in an elegant way your asynchronous operations, don't know what that means? Then SimpleAPI is just for you!

HOW DOES IT WORK?

SimpleAPI leverages the Threading Tasks system to make use of async-await operations, in short, whenever you need to perform a CRUD* operation on a web server, you can do so in an elegant way, without writing code that's too different to what you are used to, let's see an example!

**CRUD stands for:*

- *Create*
- *Read*
- *Update*
- *Delete*

EXAMPLE

Say that you want to get an array of strings from your own local data provider and print them using Debug.Log(). You would probably go about it like this:

```
void PrintData()
{
   List<string> strings = SomeManagerClass.variousStrings;
   strings.ForEach(t => Debug.Log(t.text));
}
```

Instead, if you want to fetch the same data from an external online service, you would do it like this with SimpleApi:

```
Using SimpleAPI;

async void PrintData()
{
   List<string> strings = await API.GetArray<string>("https://someStringsProvider.com/strings");
   strings.ForEach(t => Debug.Log(t.text));
}
```

Note that since we are using SimpleAPI, our method has to be marked with the keyword async and inside of it, we must await for the resolution of the API call. This is very convenient as we can be sure that the line below the API call will not be executed until we have finished fetching the online data!

PREREQUISITES

SimpleApi is composed of a namespace that you have to use at the beginning of your class:

```
using SimpleAPI
```

Since the API maps JSON data to classes, you will have to build classes that accommodate for each of the required data clusters you will be dealing with. MAKE SURE YOUR CLASSES ARE MARKED AS SERIALIZABLE!

An example for this is provided in the MockApiExample scene.

METHODS

THE METHODS INSIDE THE API CLASS CAN BE CALLED AS FOLLOWS:

```
await API.Get<T>(URL);
```

fetches a single entry from an API located at the provided URL and maps it to the class provided inside the generic T.

```
await API.GetArray<T>(URL);
```

fetches a List of entries from an API located at the provided URL and maps it to a list of objects mapped to the class provided inside the generic T.

```
await API.Put<T>(URL, DATA);
```

modifies a single entry inside an API located at the provided URL and returns an object mapped to the class provided inside the generic T.

```
await API.Post<T>(URL, DATA);
```

creates a new entry inside an API located at the provided URL and returns an object mapped to the class provided inside the generic T.

```
await API.GetTexture(URL);
```

fetches a single Texture2D from an API located at the provided URL.

```
await API.GetAudio(URL);
```

fetches a single AudioClip from an API located at the provided URL.

```
await API.GetAssetBundle(URL);
```

fetches a single AssetBundle from an API located at the provided URL.

ADDITIONALLY, THERE ARE METHODS INSIDE THE API CLASS THAT ACCEPT MULTIPLE TYPES:

```
await API.Put<T, R>(URL, DATA);
```

modifies a single entry of type T inside an API located at the provided URL and returns an object mapped to the class provided inside the generic R. This can be useful if your API has custom routes that accept a request with an object type and return a different response object.

```
await API.Post<T, R>(URL, DATA);
```

creates a new entry of type T inside an API located at the provided URL and returns an object mapped to the class provided inside the generic R. This can be useful if your API has custom routes that accept a request with an object type and return a different response object.