

RestClient for Unity 🙌

Supported Unity versions 2017.2 or higher

This **HTTP/REST** Client is based on Promises to avoid the [Callback Hell](#) 💀 and the [Pyramid of doom](#) 🗑️ working with **Coroutines** in **Unity** 🎮, example:

```
var api = "https://jsonplaceholder.typicode.com";
RestClient.GetArray<Post>(api + "/posts", (err, res) => {
    RestClient.GetArray<Todo>(api + "/todos", (errTodos, resTodos) => {
        RestClient.GetArray<User>(api + "/users", (errUsers, resUsers) => {
            //Missing validations to catch errors!
        });
    });
});
```



But working with **Promises** we can improve our code, yay! 🙌

```
RestClient.GetArray<Post>(api + "/posts").Then(response => {
    EditorUtility.DisplayDialog("Success", JsonHelper.ArrayToJson<Post>(response, true), "Ok");
    return RestClient.GetArray<Todo>(api + "/todos");
}).Then(response => {
    EditorUtility.DisplayDialog("Success", JsonHelper.ArrayToJson<Todo>(response, true), "Ok");
    return RestClient.GetArray<User>(api + "/users");
}).Then(response => {
    EditorUtility.DisplayDialog("Success", JsonHelper.ArrayToJson<User>(response, true), "Ok");
}).Catch(err => EditorUtility.DisplayDialog("Error", err.Message, "Ok"));
```

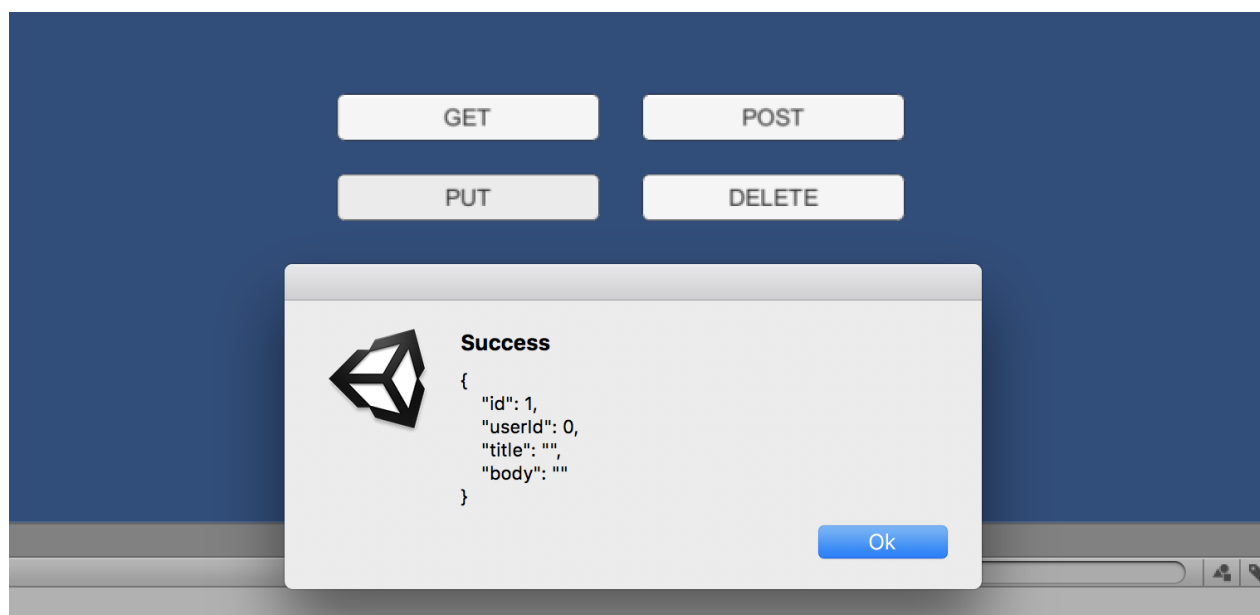
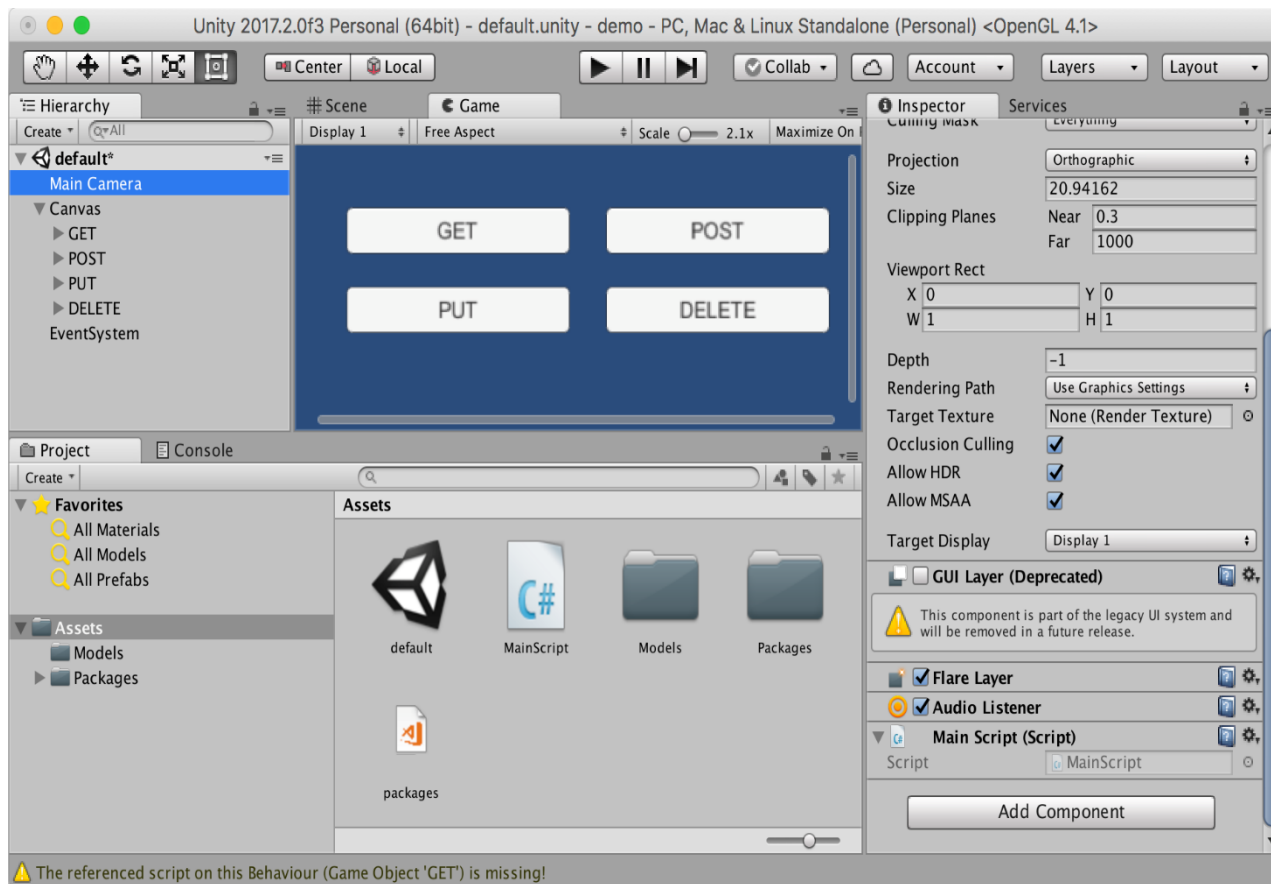
Supported platforms

The [UnityWebRequest](#) system supports most Unity platforms:

- All versions of the Editor and Standalone players
- WebGL
- Mobile platforms: iOS, Android
- Universal Windows Platform ([RSG.Promise_standard.dll](#) is required)
- PS4 and PSVita
- XboxOne
- HoloLens
- Nintendo Switch

Demo

Do you want to see this beautiful package in action? Download the demo [here](#)



Installation

Unity package

Download and install the **.unitypackage** file of the latest release published [here](#).

Nuget package

Other option is downloading this package from **NuGet** with **Visual Studio** or using the **nuget-cli**, a [NuGet.config](#) file is required at the root of your **Unity Project**, for example:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <config>
    <add key="repositoryPath" value="./Assets/Packages" />
  </config>
</configuration>
```

The package to search for is [Proyecto26.RestClient](#).

Getting Started

The default methods (**GET, POST, PUT, DELETE, HEAD**) are:

```
RestClient.Get("https://jsonplaceholder.typicode.com/posts/1").Then(res => {
    EditorUtility.DisplayDialog("Response", res.Text, "Ok");
});
RestClient.Post("https://jsonplaceholder.typicode.com/posts", newPost).Then(res => {
    EditorUtility.DisplayDialog("Status", res.StatusCode.ToString(), "Ok");
});
RestClient.Put("https://jsonplaceholder.typicode.com/posts/1", updatedPost).Then(res => {
    EditorUtility.DisplayDialog("Status", res.StatusCode.ToString(), "Ok");
});
RestClient.Delete("https://jsonplaceholder.typicode.com/posts/1").Then(res => {
    EditorUtility.DisplayDialog("Status", res.StatusCode.ToString(), "Ok");
});
RestClient.Head("https://jsonplaceholder.typicode.com/posts").Then(res => {
    EditorUtility.DisplayDialog("Status", res.StatusCode.ToString(), "Ok");
});
```

And we have a generic method to create any type of request:

```

RestClient.Request(new RequestHelper {
    Uri = "https://jsonplaceholder.typicode.com/post",
    Method = "POST",
    Timeout = 10,
    Headers = new Dictionary<string, string> {
        { "Authorization", "Bearer JWT_token..." }
    },
    Body = newPost, //Serialize object using JsonUtility by default
    BodyString = SerializeObject(newPost), // Use it instead of 'Body' to serialize objects to
JSON string using other tools
    BodyRaw = CompressToRawData(newPost), // Use it instead of 'Body' to send raw data directly
    SimpleForm = new Dictionary<string, string> {}, //Content-Type: application/x-www-form-
urlencoded
    FormSections = new List<IMultipartFormSection>() {}, //Content-Type: multipart/form-data
    DownloadHandler = new DownloadHandlerFile(destPah), //Download large files
    ContentType = "application/json", //JSON is used by default
    Retries = 3, //Number of retries
    RetrySecondsDelay = 2, //Seconds of delay to make a retry
    ChunkedTransfer = true,
    IgnoreHttpException = true //Prevent to catch http exceptions
}).Then(response => {
    EditorUtility.ShowDialog("Status", response.StatusCode.ToString(), "Ok");
});

```

With all the methods we have the possibility to indicate the type of response, in the following example we're going to create a class and the **HTTP** requests to load **JSON** data easily:

```

[Serializable]
public class User
{
    public int id;
    public string name;
    public string username;
    public string email;
    public string phone;
    public string website;
}

```

- **GET JSON**

```

var usersRoute = "https://jsonplaceholder.typicode.com/users";
RestClient.Get<User>(usersRoute + "/1").Then(firstUser => {
    EditorUtility.ShowDialog("JSON", JsonUtility.ToJson(firstUser, true), "Ok");
});

```

- **GET Array (JsonHelper is an extension to manage arrays)**

```

RestClient.GetArray<User>(usersRoute).Then(users => {
    EditorUtility.ShowDialog("Array", JsonHelper.ArrayToJsonString<User>(users, true), "Ok");
});

```

Also we can create different classes for custom responses:

```
[Serializable]
public class CustomResponse
{
    public int id;
}
```

- **POST**

```
RestClient.Post<CustomResponse>(usersRoute, newUser).Then(customResponse => {
    EditorUtility.ShowDialog("JSON", JsonUtility.ToJson(customResponse, true), "Ok");
});
```

- **PUT**

```
RestClient.Put<CustomResponse>(usersRoute + "/1", updatedUser).Then(customResponse => {
    EditorUtility.ShowDialog("JSON", JsonUtility.ToJson(customResponse, true), "Ok");
});
```

Custom HTTP Headers and Options

HTTP Headers, such as Authorization, can be set in the **DefaultRequestHeaders** object for all requests

```
RestClient.DefaultRequestHeaders["Authorization"] = "Bearer ...";
```

Also we can add specific options and override default headers for a request

```
var currentRequest = new RequestHelper {
    Uri = "https://jsonplaceholder.typicode.com/photos",
    Headers = new Dictionary<string, string> {
        { "Authorization", "Other token..." }
    }
};
RestClient.GetArray<Photo>(currentRequest).Then(response => {
    EditorUtility.ShowDialog("Header", currentRequest.GetHeader("Authorization"), "Ok");
});
```

```
currentRequest.UploadProgress; //To know the progress by uploading data to the server
currentRequest.DownloadProgress; //To know the progress by downloading data from the server
currentRequest.Abort(); //Abort the request manually
```

And later we can clean the default headers for all requests

```
RestClient.CleanDefaultHeaders();
```

Full Example

- Unity

```
[Serializable]
public class ServerResponse {
    public string id;
    public string date; //DateTime is not supported by JsonUtility
}
[Serializable]
public class User {
    public string firstName;
    public string lastName;
}
RestClient.Post<ServerResponse>("www.api.com/endpoint", new User {
    firstName = "Juan David",
    lastName = "Nicholls Cardona"
}).Then(response => {
    EditorUtility.DisplayDialog("ID: ", response.id, "Ok");
    EditorUtility.DisplayDialog("Date: ", response.date, "Ok");
});
```

- NodeJS as Backend (Using [Express](#))

```
router.post('/', function(req, res) {
    console.log(req.body.firstName)
    res.json({
        id: 123,
        date: new Date()
    })
});
```

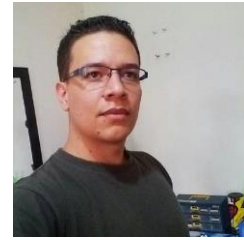
Collaborators



[Juan Nicholls](#)



[Diego Ossa](#)




[Nasdull](#)

Credits

- Promises library for C#: [Real Serious Games/C-Sharp-Promise](#)

Supporting

I believe in Unicorns  Support [me](#), if you do too.

Happy coding

Made with 