Retrofit for Unity

sp958857

Introduction

Retrofit turns your HTTP API into a interface.

```
public interface IRestInterface
    {
        // baseURL/repos/{owner}/{repo}/contributors
        [Get("/repos/{owner}/{repo}/contributors")]
        void GetList(
            Callback<List<Contributor>> cb,
            [Path("owner")] string owner,
            [Path("repo")] string repo);
    }
```

How To Use

Three steps to retrofit your HTTP API Usage:
-**[1] Define HTTP API in Interface**: Define a interface to maintain your HTTP API
-**[2] Extend RestAdapter and Implement your HTTP API defined interface**: Just call `SendRequest(args...)` in the implemented member of the interface.

```
public class RestApiService : RestAdapter,IRestInterface
    {
        void Awake()
        {
            Instance = this;
            base.Awake();
        }
        protected override void SetUp()
        {
            baseUrl = "https://api.github.com";
            httpImpl = new UnityWebRequestImpl();
            iRestInterface = typeof (IRestApi);
        }

        public void GetList(Callback<List<Contributor>> cb, string
owner, string repo)
        {
            SendRequest(cb, owner,repo);
        }
}
```

-**[3] Call it**: Each Call from the created RestApiService can make a HTTP request to the remote webserver.

```
  var cb= new Callback<List<Contributor>>()
            {
                errorCB = delegate (string errorMessage)
                {
                    Debug.Log("Retrofit Error:" + errorMessage);
                },
                successCB = delegate (List<Contributor> response)
                {
                    Debug.Log("Retrofit Success:" + response.toStri
ng());
                }
            };
  RestApiService.Instance.GetList(cb, "square", "retrofit");
```

API Declaration

Attributes on the interface methods and its parameters indicate how a request will be handled.

REQUEST METHOD

Every method must have an HTTP Attribute that provides the request method and relative URL. There are six built-in annotations: GET, POST, PUT, DELETE, PATCH and HEAD. The relative URL of the resource is specified in the Attribute.

```
 [Get("/users")]
```

You can also specify query parameters in the URL.

```
[GET("/users/list?sort=desc")]
```

URL MANIPULATION

A request URL can be updated dynamically using replacement blocks and parameters on the method. A replacement block is an alphanumeric string surrounded by `{` and `}`. A corresponding parameter must be attributed with [Path] using the same string.

```
[Get("/group/{id}/users")]
void GetList(
    Callback<<List<User>> cb,
    [Path("id")] int groupId);
```

Query parameters can also be added.

```
[Get("/group/{id}/users")]
void GetList(
    Callback<<List<User>> cb,
    [Path("id")] int groupId,
    [Query("sort")] string sort);
```

For complex query parameter combinations a Map can be used.

```
[Get("/group/{id}/users")]
void GroupList(
    Callback<<List<User>> cb,
    [Path("id")] int groupId,
    [QueryMap] Dictionary<string, string> options);
```

REQUEST BODY

An object can be specified for use as an HTTP request body with the [Body] attribute.

```
[POST("/users/new")]
void CreateUser(
    Callback<User> cb,
    [Body] User user);
```

The object will also be converted using a converter specified on the subClass of RestAdapter. If no converter is specified, RestAdapter will use `DefalutConverter(Implemented by Newtonsoft.json)` to convert the body.

FORM ENCODED

Methods can also be declared to send form-encoded data.

Form-encoded data is sent when [Field] is present on the parameter. Each key-value pair is attributed with [Field] containing the name and the object providing the value.

```
[POST("/user/edit")]
void UpdateUser(
    Callback<User> cb,
    [Field("first_name")] string first,
    [Field("last_name")] string last);
```

HEADER MANIPULATION

You can set static headers for a method using the [Headers] attribute.

```
[Headers("Cache-Control: max-age=640000")]
[GET("/widget/list")]
void  WidgetList(
    Callback<List<Widget>> cb);
```

```
[Headers({
    "Accept: application/vnd.github.v3.full+json",
    "User-Agent: Retrofit-Sample-App"
})]
[GET("/users/{username}")]
void GetUser(Callback<User> cb,
    [Path("username")] string username);
```

Note that headers do not overwrite each other. All headers with the same name will be included in the request.

A request Header can be updated dynamically using the `[Header]` attribute. A corresponding parameter must be provided to the `[Header]`.

```
[GET("/user")]
void  GetUser(Callback<User> cb,
    [Header("Authorization")] string authorization)
```

Retrofit Configuration

`RestApater` is the class through which your API interfaces are turned into callable objects. By default, Retrofit will give you sane defaults for your platform but it allows for customization.

CONVERTERS

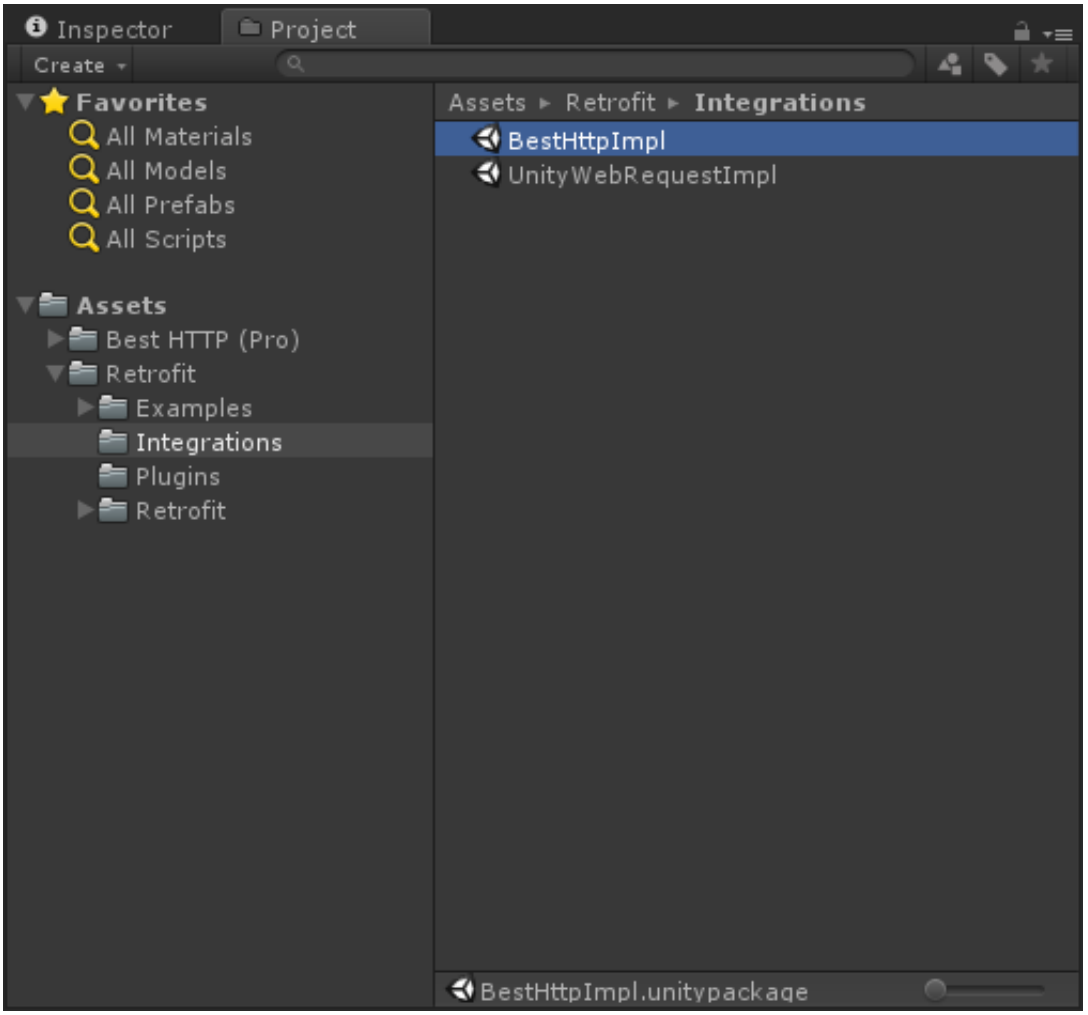
By default, Retrofit uses Newtonsoft.json for its deserialization and serialization.
If you need to communicate with an API that uses a content-format that Retrofit does not support out of the box or you wish to use a different library to implement an existing format, you can easily create your own converter. Create a class that implements the `Converter` interface and pass in an instance when `Setup()` your adapter.

Intergrate With Different Http Module

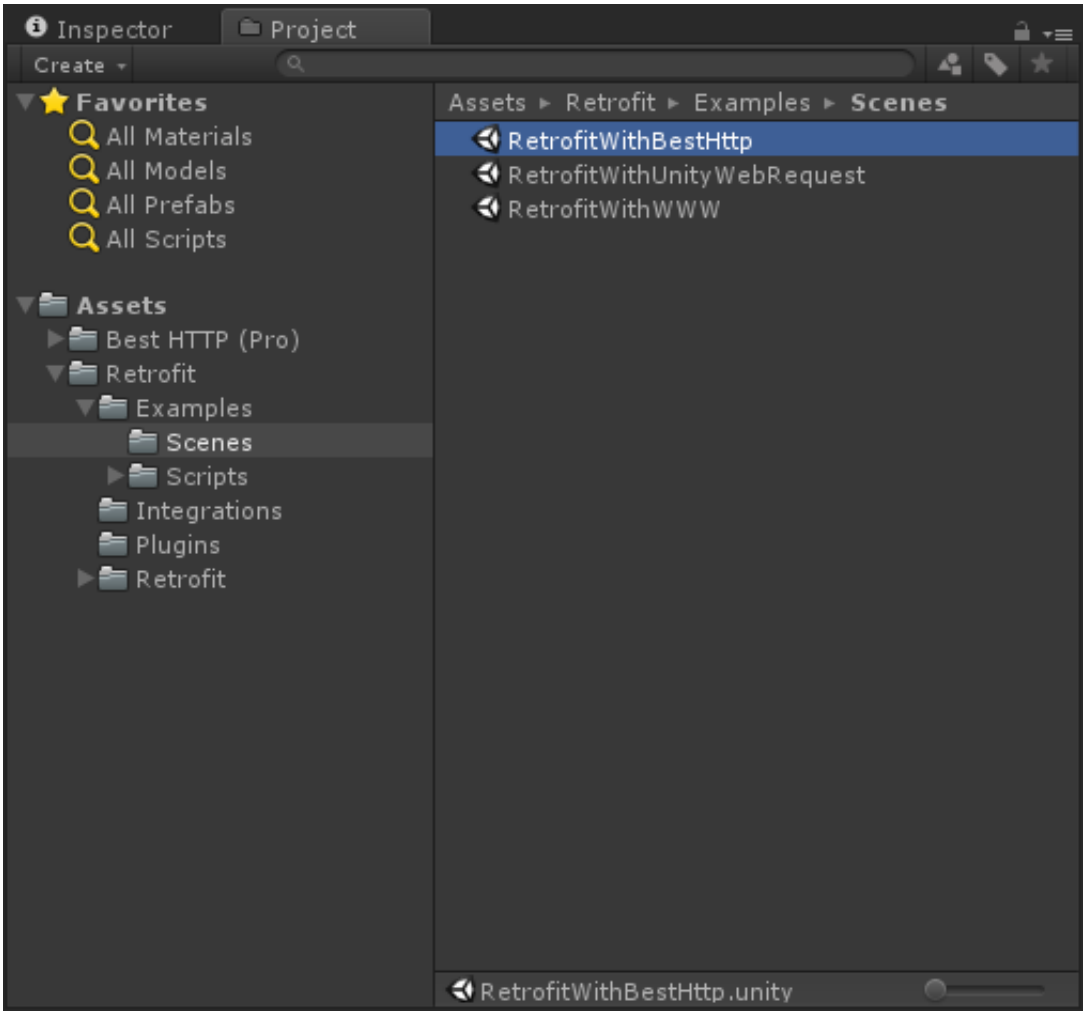
Retrofit for Unity is just a reformatted code style to request a HTTP API. It doesn't send a real HTTP Request in its module, which means you can integrated `Retrofit for Unity` with different HTTP Plugins or Unity Official `UnityWebRequest` system.
By default, it provides two integrations to send HTTP request, which are `UnityWebRequestImpl` and `BestHttpImpl`.
Personally I recommand to use BestHttpImpl, because it is a out of box solution for requesting RESTful API.
If you need to integrate Retrofit with other HTTP Plugins, just create a class that implements the `HttpImplement` interface and pass in an instance when `Setup()` your adapter.

INTEGRATE WITH BESTHTTP

1. Import BestHttp plugins.
2. Import Retrofit for Unity.
3. Import `BestHttpImpl.unitypackage` under the fold names Integrations.



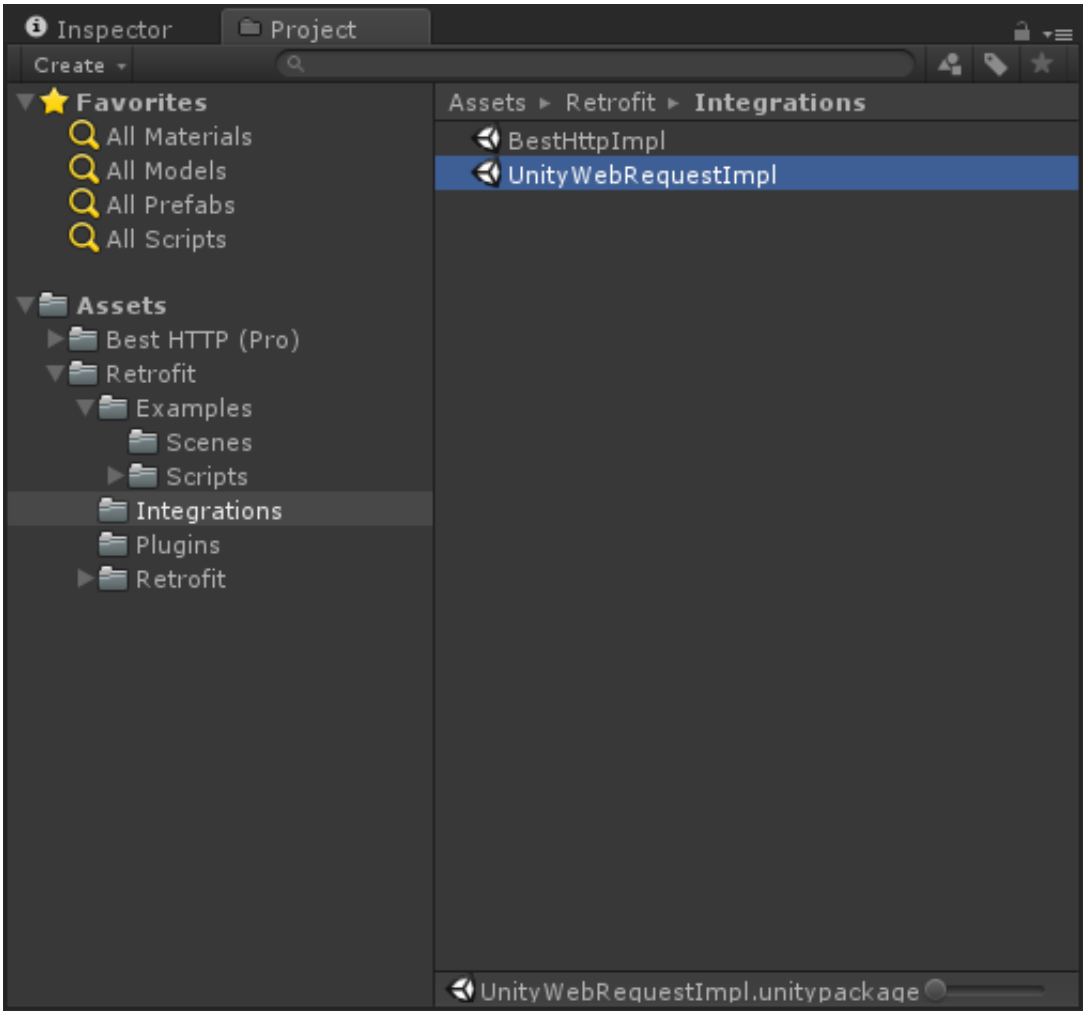
4. Play RetrofitWithBestHttp.scene to send request by BestHttp in Retrofit style.



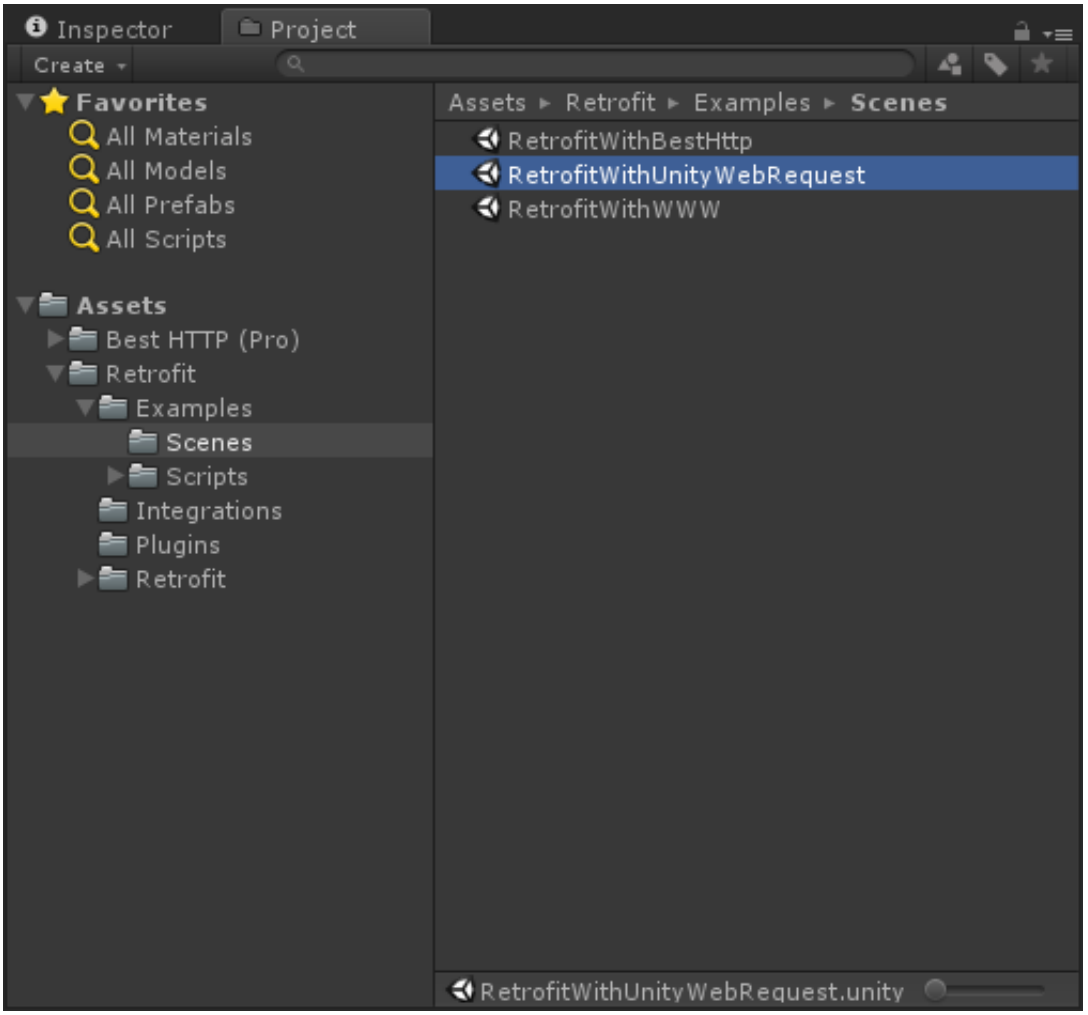
INREGRATE WITH UNITYWEBREQUEST

> To integrate with UnityWebRequest, it requires unity version 5.4 or higher.

1. Import Retrofit for Unity.
2. Import `UnityWebRequestImpl.unitypackage` under the fold names Integrations.



3. Play RetrofitWithUnityWebRequest.scene to send request by UnityWebRequest in Retrofit style.


Inspector
Project
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Best HTTP (Pro)
Retrofit
Examples
Scenes
Scripts
Integrations
Plugins
Retrofit
Assets ▸ Retrofit ▸ Examples ▸ Scenes
RetrofitWithBestHttp
RetrofitWithUnityWebRequest
RetrofitWithWWW
RetrofitWithUnityWebRequest.unity