Introduction

Retrofit turns your HTTP API into a interface. Calling a request like calling a method.

What's new in Version 2.0

-**[1] Add** `Rx(Reactive Extension)` **support to Retrofit4Unity**, which help us to request asynchronously. For example send request in sub-thread then handle response and update UI in main-thread.
-**[2]use HttpClient as default Request Module**. Of Course we can still use BestHttp with Rx.

Ps:Versiono 1.0 User Manual

Retrofit for Unity 1.0.pdf
196.8 KB

```
public interface IHttpBinInterface
    {
        [Get("/get")]
        IObservable<HttpBinResponse> Get(
            [Query("query1")]string arg1,
            [Query("query2")]string arg2
            );
     }
```

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 this interface.

```
public class HttpBinService:RestAdapter,IHttpBinInterface
    {
        private static HttpBinService _instance;

        public static HttpBinService Instance
        {
            get
            {
                if (_instance == null)
```

```
                {
                    var go = new GameObject("HttpBinService");
                    _instance = go.AddComponent<HttpBinService>();
                }
                return _instance;
            }
        }

        protected override HttpImplement SetHttpImpl()
        {
            var httpImpl = new HttpClientImpl();
            httpImpl.EnableDebug = true;
            return httpImpl;
        }

        protected override void SetRestAPI()
        {
            baseUrl = "http://httpbin.org";
            iRestInterface = typeof (IHttpBinInterface);
        }


        public IObservable<HttpBinResponse> Get(string arg1, string arg2)
        {
            return SendRequest<HttpBinResponse>(arg1,arg2) as IObservable
<HttpBinResponse>;
        }
    }
```

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

```
 var ob = HttpBinService.Instance.Get("abc", "123");
     ob.SubscribeOn(Scheduler.ThreadPool)//send request in sub thread
          .ObserveOn(Scheduler.MainThread)//receive response in main thre
ad
          .Subscribe(data =>
              {
                  // onSuccess
                  Debug.LogFormat("Received on threadId:{0}", Thread.Cur
rentThread.ManagedThreadId);
              },
              error =>
              {
                  Debug.Log("Retrofit Error:" + error);
              });
```

Thanks to the UniRx, we can easy implement asynchronous call to webserver.

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")]
IObservable<<List<User>> GetList(
    [Path("id")] int groupId);
```

Query parameters can also be added.

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

For complex query parameter combinations a Map can be used.

```
[Get("/group/{id}/users")]
IObservable<<List<User>>GroupList(
    [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")]
IObservable<User> CreateUser(
    [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")]
IObservable<User> UpdateUser(
    [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")]
IObservable<List<Widget>> WidgetList();
```

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

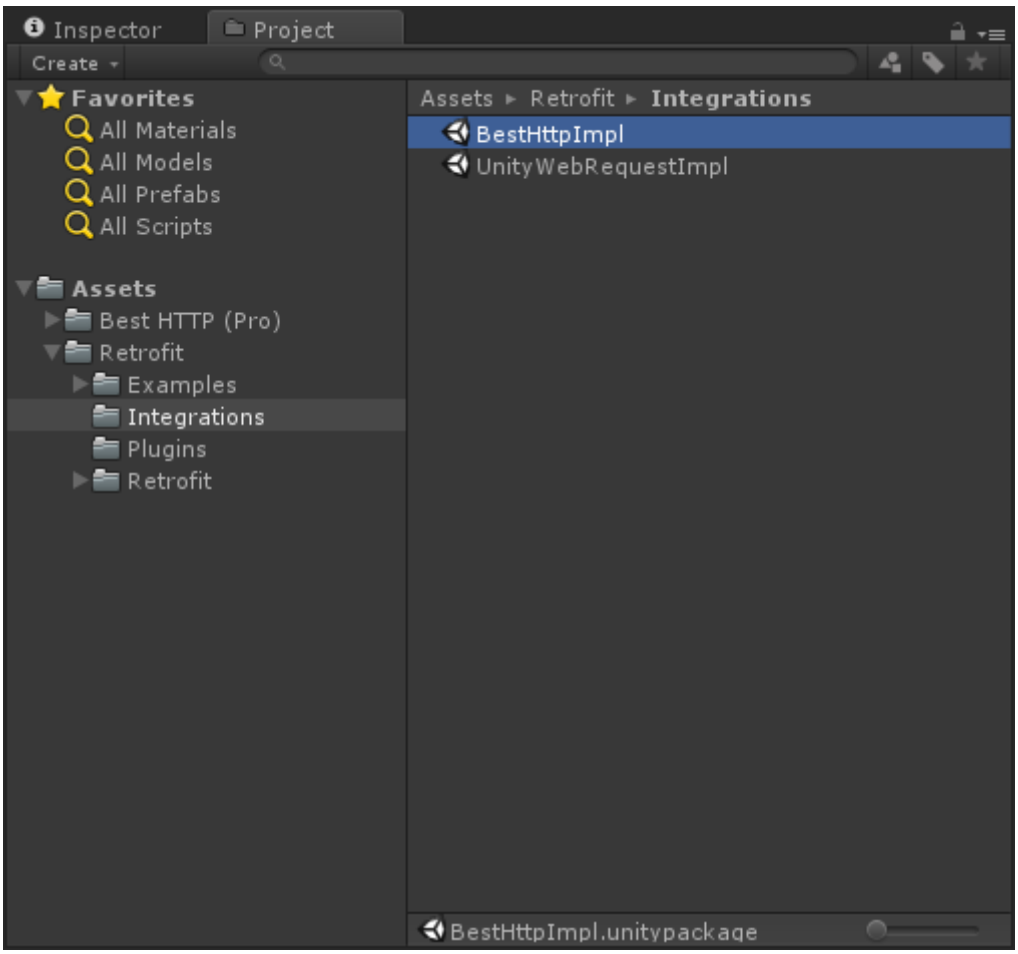
By default, Retrofit uses Newtonsoft.json for its deserialization and serialization.
If you need to communicate with an API that uses a content-format that Retrofit does not support out of the box or you wish to use a different library to implement an existing format, you can easily create your own converter. 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 three integrations to send HTTP request, which are `HttpClientImpl` `UnityWebRequestImpl` and `BestHttpImpl`.
As default, it use HttpClient as request module, because its a quite lite request module.
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.



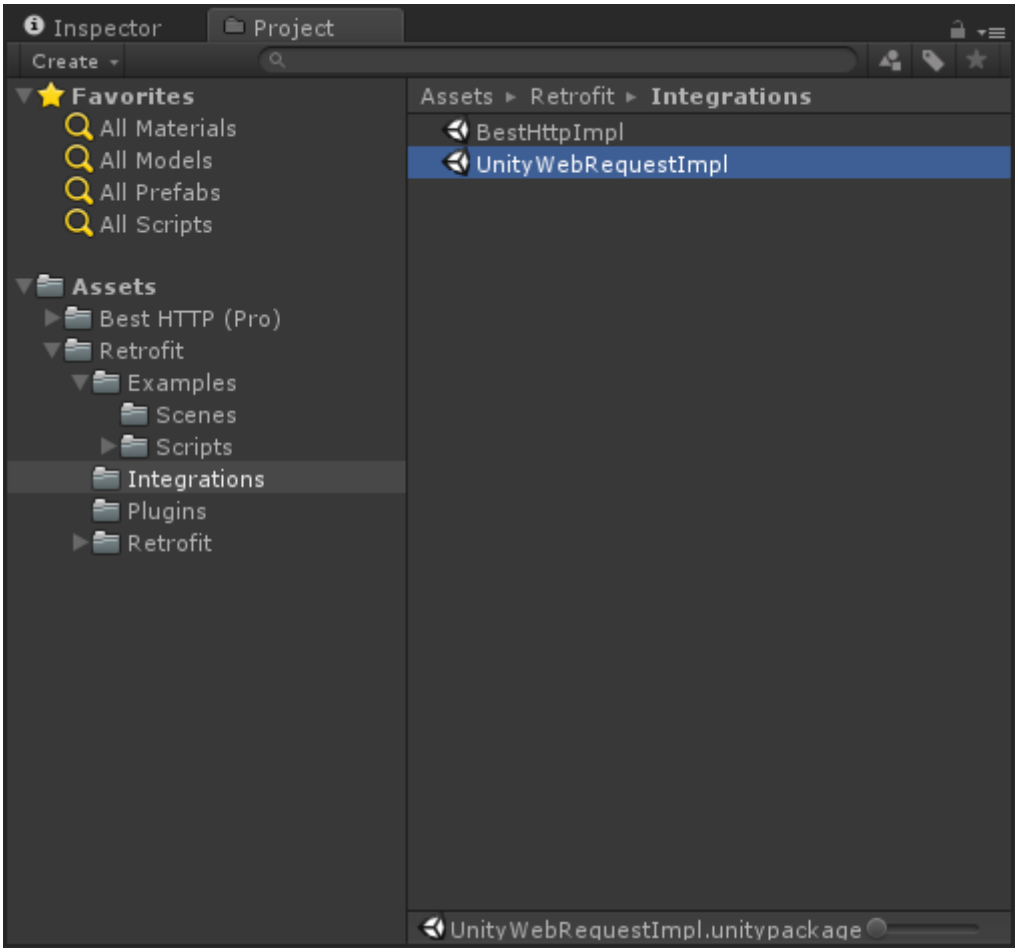
4. Replace the return value in `SetHttpImpl()` from RestAdapter.

```
protected override HttpImplement SetHttpImpl()
        {
            return new BestHttpImpl();
        }
```

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. Replace the return value in `SetHttpImpl()` from RestAdapter.

```
protected override HttpImplement SetHttpImpl()
        {
            return new UnityWebRequestImpl();
        }
```