C# Demo

Scott Hurrey

{% assign sluggedName = page.name | replace: '.md' %}

modified: {{ page.last_modified_at | date: '%b-%d-%y' }}

C# Demo

The rest demo script demonstrates authenticating a REST application, management and use of the authorization token, and creating, updating, discovering, and deleting supported Learn objects

Prerequisites

- You must register a developer account and application in the Developer Portal
- You must register your application in Learn
- You must also configure the script as outlined in the README for the project

Overview

This C# Console Application allows you to:

- Authenticate
- Create, Read, and Update a Data Source
- Create, Read, and Update a Term
- Create, Read, and Update a Course
- Create, Read, and Update a User
- Create, Read, and Update a Membership
- Delete created objects in reverse order of create - membership, user, course, term, datasource.

All generated output is sent to the console.

This is not meant to be a C# tutorial. It will not teach you to write code in C#. It will, however, give a Developer familiar with C# the knowledge necessary to build a Web Services integration.

Assumptions

This help topic assumes the Developer:

- is familiar with C#
- has installed Microsoft Visual Studio
- has obtained a copy of the source code and built it in conjunction with the project README.md file.
- has a REST-enabled Learn instance, like the Developer AMI.

Code Walkthrough

To build an integration with the Learn REST Web Services, regardless of the programming language of choice, can really be summed up in two steps:

1. Use the Application Key and Secret to obtain an OAuth 2.0 access token, as described in the Basic Authentication document.

2. Call the appropriate REST endpoint with the appropriate data to perform the appropriate action.

Authorization and Authentication The REST Services rely on OAuth 2.0 Bearer Tokens for authentication. A request is made to the token endpoint with a Basic Authorization header containing the base64-encoded key:secret string as its key. The token service returns a JSON object containing the Access Token, the Token Type, and the number of seconds until the token expires. The token is set to expire after one hour, and subsequent calls to retrieve the token will return the same token with an updated expiry time until such time that the token has expired. There is no refresh token and currently no revoke token method.

The C# code handles this in bbdn.rest.Authorizer:

```
var authData = string.Format ("{0}:{1}", Constants.KEY, Constants.SECRET);
var authHeaderValue = Convert.ToBase64String (Encoding.UTF8.GetBytes (authData));
client = new HttpClient ();
var endpoint = new Uri(Constants.HOSTNAME + Constants.AUTH_PATH);
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue ("Basic",
↪ authHeaderValue);
var postData = new List<KeyValuePair<string, string>>();
postData.Add(new KeyValuePair<string, string>("grant_type", "client_credentials"));
HttpContent body = new FormUrlEncodedContent(postData);
HttpResponseMessage response;
try {
  response = await client.PostAsync(endpoint, body)
    if (response.IsSuccessStatusCode)
    {
         var content = await response.Content.ReadAsStringAsync();
          token = JsonConvert.DeserializeObject<Token>(content);
```

The JSON response is serialized into the Token object, and you may then retrieve those values from that object.

Calling Services The individual service calls are handled by C# Classes in the `bbdn.rest.services` package, and each individual service class implements the `bbdn.rest.services.IRestService` interface. The interface is used to normalize each service handler to make additional service implementation standardized as new endpoints are added.

IRestService dictates that four methods must be implemented:

- Task CreateObject (TRestModel T);
- Task ReadObject ();
- Task UpdateObject (TRestModel T);
- Task DeleteObject ();

The `Task<TRestModel>` allows the code to run asynchronously, but specify when an operation should be handled synchronously before proceeding with the remaining code. TRestModel is a generic class place holder that allows the code to implement the Interface, but pass it an Object type when it is instantiated in order to take and return an individual service model dependent upon the Rest endpoint being implemented.

The individual service class must be defined in the following way, to ensure the Interface is using the appropriate class type for TRestModel:

```
public class DatasourceService : IRestService<Datasource>, IDisposable
```

Each of these methods creates the JSON body by instantiating the appropriate model from the bbdn.rest.models package when necessary, and then generates the appropriate HTTP Request, ships it to Learn, and serializes the JSON response back into the appropriate model.

End points are generally defined as `/learn/api/public/v1/<object type>/<objectId>`. Object ID can be either the pk1, like `_1_1`, or as the batchuid. This value should be prepended by externalId:, like `externalId:test101`.

For example, to retrieve a course by the pk1 `_1_1`, you would call **GET /learn/api/public/v1/courses/_1_1**. To retrieve by the batchuid test101, you would call **GET /learn/api/public/v1/courses/externalId:test101.**

Create is sent to Learn as a HTTP POST message with a JSON body that defines the object. The endpoint should omit the objectId, as this will be generated on creation.

Read is sent to Learn as a HTTP GET message with an empty body. The endpoint should include the objectId being retrieved.

Update is sent to Learn as a HTTP PATCH message with a JSON body that defines the object. The endpoint should include the objectId being updated.

Delete is sent to Learn as a HTTP DELETE message with empty body. The endpoint should include the objectId being deleted.

Datasources

Datasources are handled in `bbdn.rest.services.DatasourceService`. As illustrated above, this Class implements the IRestService interface and exposes four methods. It also includes methods required to implement the IDisposible interface which is required to use the async/await functionality..

Create

```
public async Task<Datasource> CreateObject (Datasource dataSource)
{
       Datasource datasource = new Datasource();
       var uri = new Uri( Constants.HOSTNAME + Constants.DATASOURCE_PATH);
       try {
            var json = JsonConvert.SerializeObject (dataSource);
            var body = new StringContent (json, Encoding.UTF8, "application/json");
            HttpResponseMessage response = await client.PostAsync (uri, body);
            if (response.IsSuccessStatusCode) {
                 var content = await response.Content.ReadAsStringAsync ();
                 datasource = JsonConvert.DeserializeObject <Datasource> (content);
                 Debug.WriteLine (@" Datasource successfully created.");
            }
       } catch (Exception ex) {
            Debug.WriteLine (@" ERROR {0}", ex.Message);
       }
       return datasource;
}
```

Read

```
public async Task<Datasource> ReadObject ()
{
  Datasource datasource = new Datasource();
  var uri = new Uri(Constants.HOSTNAME + Constants.DATASOURCE_PATH + "/externalId:" +
↪ Constants.DATASOURCE_ID);
  try {
       var response = await client.GetAsync (uri);
       if (response.IsSuccessStatusCode) {
            var content = await response.Content.ReadAsStringAsync ();
            datasource = JsonConvert.DeserializeObject<Datasource>(content);
           }
  } catch (Exception ex) {
  Debug.WriteLine (@" ERROR {0}", ex.Message);
  }
  return datasource;
}
```

Update

```
public async Task<Datasource> UpdateObject (Datasource updateDataSource)
{
        Datasource datasource = new Datasource();
        try {
             var json = JsonConvert.SerializeObject (updateDataSource);
             var body = new StringContent (json, Encoding.UTF8, "application/json");
             HttpResponseMessage response =  await HttpClientExtensions.PatchAsync (client,
↪ Constants.HOSTNAME + Constants.DATASOURCE_PATH + "/externalId:" + Constants.DATASOURCE_ID,
↪ body);
             if (response.IsSuccessStatusCode) {
                  Debug.WriteLine (@" Datasource successfully updated.");
                  var content = await response.Content.ReadAsStringAsync();
                  datasource = JsonConvert.DeserializeObject<Datasource>(content);
             }
        } catch (Exception ex) {
             Debug.WriteLine (@" ERROR {0}", ex.Message);
        }
        return (datasource);
}
```

Delete

```
public async Task<Datasource> DeleteObject ()
{
        Datasource datasource = new Datasource();
        var uri = new Uri(Constants.HOSTNAME + Constants.DATASOURCE_PATH + "/externalId:" +
↪ Constants.DATASOURCE_ID);
        try {
             var response = await client.DeleteAsync (uri);
             if (response.IsSuccessStatusCode) {
                  Debug.WriteLine (@" Datasource successfully deleted.");
                  var content = await response.Content.ReadAsStringAsync();
                  datasource = JsonConvert.DeserializeObject<Datasource>(content);
             }
        } catch (Exception ex) {
             Debug.WriteLine (@" ERROR {0}", ex.Message);
        }
        return (datasource);
}
```

Terms

Terms are handled in `bbdn.rest.services.TermService`. As illustrated above, this Class implements the IRestService interface and exposes four methods. It also includes methods required to implement the IDisposible interface which is required to use the async/await functionality..

Create

```
public async Task<Term> CreateObject(Term newTerm)
{
        Term term = new Term();
        var uri = new Uri(Constants.HOSTNAME + Constants.TERM_PATH);
        try
        {
            var json = JsonConvert.SerializeObject(newTerm);
            var body = new StringContent(json, Encoding.UTF8, "application/json");
            HttpResponseMessage response = await client.PostAsync(uri, body);
            if (response.IsSuccessStatusCode)
```

```
                    {
                        var content = await response.Content.ReadAsStringAsync();
                        term = JsonConvert.DeserializeObject<Term>(content);
                        Debug.WriteLine(@" Term successfully created.");
                    }
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(@" ERROR {0}", ex.Message);
                }
                return term;
            }
```

Read

```
    public async Task<Term> ReadObject()
    {
                Term term = new Term();
                var uri = new Uri(Constants.HOSTNAME + Constants.TERM_PATH + "externalId:" +
↪   Constants.TERM_ID);
                try
                {
                    var response = await client.GetAsync(uri);
                    if (response.IsSuccessStatusCode)
                    {
                        var content = await response.Content.ReadAsStringAsync();
                        term = JsonConvert.DeserializeObject<Term>(content);
                    }
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(@" ERROR {0}", ex.Message);
                }
                return term;
    }
```

Update

```
            public async Task<Term> UpdateObject(Term updateTerm)
            {
                Term term = new Term();
                try
                {
                    var json = JsonConvert.SerializeObject(updateTerm);
                    var body = new StringContent(json, Encoding.UTF8, "application/json");
                    HttpResponseMessage response = await
↪   HttpClientExtensions.PatchAsync(client, Constants.HOSTNAME + Constants.TERM_PATH +
↪   "externalId:" + Constants.TERM_ID, body);
                    if (response.IsSuccessStatusCode)
                    {
                        Debug.WriteLine(@" Term successfully updated.");
                        var content = await response.Content.ReadAsStringAsync();
                        term = JsonConvert.DeserializeObject<Term>(content);
                    }
                }
                catch (Exception ex)
                {
```

```
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return (term);
        }
```

Delete

```
        public async Task<Term> DeleteObject()
        {
            Term term = new Term();
            var uri = new Uri(Constants.HOSTNAME + Constants.TERM_PATH + "externalId:" +
↪  Constants.TERM_ID);
            try
            {
                var response = await client.DeleteAsync(uri);
                if (response.IsSuccessStatusCode)
                {
                    Debug.WriteLine(@" Term successfully deleted.");
                    var content = await response.Content.ReadAsStringAsync();
                    term = JsonConvert.DeserializeObject<Term>(content);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return (term);
        }
```

Courses

Course are handled in `bbdn.rest.services.CourseService`. As illustrated above, this Class implements the IRestService interface and exposes four methods. It also includes methods required to implement the IDisposible interface which is required to use the async/await functionality..

Create

```
        public async Task<Course> CreateObject(Course newCourse)
        {
            Course course = new Course();
            var uri = new Uri(Constants.HOSTNAME + Constants.COURSE_PATH);
            try
            {
                var json = JsonConvert.SerializeObject(newCourse);
                var body = new StringContent(json, Encoding.UTF8, "application/json");
                HttpResponseMessage response = await client.PostAsync(uri, body);
                if (response.IsSuccessStatusCode)
                {
                    var content = await response.Content.ReadAsStringAsync();
                    course = JsonConvert.DeserializeObject<Course>(content);
                    Debug.WriteLine(@" Course successfully created.");
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
```

```
            return course;
        }
```

Read

```
        public async Task<Course> ReadObject()
        {
            Course course = new Course();
            var uri = new Uri(Constants.HOSTNAME + Constants.COURSE_PATH + "/externalId:" +
↪   Constants.COURSE_ID);
            try
            {
                var response = await client.GetAsync(uri);
                if (response.IsSuccessStatusCode)
                {
                    var content = await response.Content.ReadAsStringAsync();
                    course = JsonConvert.DeserializeObject<Course>(content);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return course;
        }
```

Update

```
        public async Task<Course> UpdateObject(Course updateCourse)
        {
            Course course = new Course();
            try
            {
                var json = JsonConvert.SerializeObject(updateCourse);
                var body = new StringContent(json, Encoding.UTF8, "application/json");
                HttpResponseMessage response = await
↪   HttpClientExtensions.PatchAsync(client, Constants.HOSTNAME + Constants.COURSE_PATH +
↪   "/externalId:" + Constants.COURSE_ID, body);
                if (response.IsSuccessStatusCode)
                {
                    Debug.WriteLine(@" Course successfully updated.");
                    var content = await response.Content.ReadAsStringAsync();
                    course = JsonConvert.DeserializeObject<Course>(content);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return (course);
        }
```

Delete

```
        public async Task<Course> DeleteObject()
        {
            Course course = new Course();
```

```
            var uri = new Uri(Constants.HOSTNAME + Constants.COURSE_PATH + "/externalId:" +
↪ Constants.COURSE_ID);
            try
            {
                var response = await client.DeleteAsync(uri);
                if (response.IsSuccessStatusCode)
                {
                    Debug.WriteLine(@" Course successfully deleted.");
                    var content = await response.Content.ReadAsStringAsync();
                    course = JsonConvert.DeserializeObject<Course>(content);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return (course);
        }
```

Users

Users are handled in `bbdn.rest.services.UserService`. As illustrated above, this Class implements the IRestService interface and exposes four methods. It also includes methods required to implement the IDisposible interface which is required to use the async/await functionality..

Create

```
        public async Task<User> CreateObject(User newUser)
        {
            User user = new User();
            var uri = new Uri(Constants.HOSTNAME + Constants.USER_PATH);
            try
            {
                var json = JsonConvert.SerializeObject(newUser);
                var body = new StringContent(json, Encoding.UTF8, "application/json");
                HttpResponseMessage response = await client.PostAsync(uri, body);
                if (response.IsSuccessStatusCode)
                {
                    var content = await response.Content.ReadAsStringAsync();
                    user = JsonConvert.DeserializeObject<User>(content);
                    Debug.WriteLine(@" User successfully created.");
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return user;
        }
```

Read

```
        public async Task<User> ReadObject()
        {
            User user = new User();
            var uri = new Uri(Constants.HOSTNAME + Constants.USER_PATH + "externalId:" +
↪ Constants.USER_ID);
            try
```

```
            {
                var response = await client.GetAsync(uri);
                if (response.IsSuccessStatusCode)
                {
                    var content = await response.Content.ReadAsStringAsync();
                    user = JsonConvert.DeserializeObject<User>(content);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return user;
        }
```

Update

```
        public async Task<User> UpdateObject(User updateUser)
        {
            User user = new User();
            try
            {
                var json = JsonConvert.SerializeObject(updateUser);
                var body = new StringContent(json, Encoding.UTF8, "application/json");
                HttpResponseMessage response = await
↪  HttpClientExtensions.PatchAsync(client, Constants.HOSTNAME + Constants.USER_PATH +
↪  "externalId:" + Constants.USER_ID, body);
                if (response.IsSuccessStatusCode)
                {
                    Debug.WriteLine(@" User successfully updated.");
                    if (response.IsSuccessStatusCode)
                    {
                        var content = await response.Content.ReadAsStringAsync();
                        user = JsonConvert.DeserializeObject<User>(content);
                    }
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(@" ERROR {0}", ex.Message);
            }
            return user;
        }
```

Delete

```
        public async Task<User> DeleteObject()
        {
            User user = new User();
            var uri = new Uri(Constants.HOSTNAME + Constants.USER_PATH + "externalId:" +
↪  Constants.USER_ID);
            try
            {
                var response = await client.DeleteAsync(uri);
                if (response.IsSuccessStatusCode)
                {
                    Debug.WriteLine(@" User successfully deleted.");
```

```
                var content = await response.Content.ReadAsStringAsync();
                user = JsonConvert.DeserializeObject<User>(content);
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine(@" ERROR {0}", ex.Message);
        }
        return (user);
    }
```

Memberships

Memberships are handled in `bbdn.rest.services.MemberService`. As illustrated above, this Class implements the IRestService interface and exposes four methods. It also includes methods required to implement the IDisposible interface which is required to use the async/await functionality.. In addition, the endpoint for memberships is a bit different, in that it is a sub-call to courses, so the endpoint would look like `/learn/api/public/v1/courses/<courseId>/users/<userId>`.

Create

```
public async Task<Membership> CreateObject(Membership newMembership)
{
    Membership membership = new Membership();
    var uri = new Uri(Constants.HOSTNAME + Constants.COURSE_PATH + "/externalId:" +
    Constants.COURSE_ID + "users/externalId:" + Constants.USER_ID);
    try
    {
        var json = JsonConvert.SerializeObject(membership);
        var body = new StringContent(json, Encoding.UTF8, "application/json");
        HttpResponseMessage response = await client.PostAsync(uri, body);
        if (response.IsSuccessStatusCode)
        {
            var content = await response.Content.ReadAsStringAsync();
            membership = JsonConvert.DeserializeObject<Membership>(content);
            Debug.WriteLine(@" Membership successfully created.");
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@" ERROR {0}", ex.Message);
    }
    return membership;
}
```

Read

```
public async Task<Membership> ReadObject()
{
    Membership membership = new Membership();
    var uri = new Uri(Constants.HOSTNAME + Constants.COURSE_PATH + "/externalId:" +
    Constants.COURSE_ID + "users/externalId:" + Constants.USER_ID);
    try
    {
        var response = await client.GetAsync(uri);
        if (response.IsSuccessStatusCode)
        {
            var content = await response.Content.ReadAsStringAsync();
```

```
            membership = JsonConvert.DeserializeObject<Membership>(content);
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@" ERROR {0}", ex.Message);
    }
    return membership;
}
```

Update

```
public async Task<Membership> UpdateObject(Membership updateMembership)
{
    Membership membership = new Membership();
    try
    {
        var json = JsonConvert.SerializeObject(updateMembership);
        var body = new StringContent(json, Encoding.UTF8, "application/json");
        HttpResponseMessage response = await HttpClientExtensions.PatchAsync
        (client, Constants.HOSTNAME + Constants.COURSE_PATH + "/externalId:"
        + Constants.COURSE_ID + "users/externalId:" + Constants.USER_ID, body);
        if (response.IsSuccessStatusCode)
        {
            Debug.WriteLine(@" Membership successfully updated.");
            var content = await response.Content.ReadAsStringAsync();
            membership = JsonConvert.DeserializeObject<Membership>(content);
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@" ERROR {0}", ex.Message);
    }
    return (membership);
}
```

Delete

```
public async Task<Membership> DeleteObject()
{
    Membership membership = new Membership();
    var uri = new Uri(Constants.HOSTNAME + Constants.COURSE_PATH + "/externalId:" +
    Constants.COURSE_ID + "users/externalId:" + Constants.USER_ID);
    try
    {
        var response = await client.DeleteAsync(uri);
        if (response.IsSuccessStatusCode)
        {
            Debug.WriteLine(@" Membership successfully deleted.");
            var content = await response.Content.ReadAsStringAsync();
            membership = JsonConvert.DeserializeObject<Membership>(content);
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@" ERROR {0}", ex.Message);
    }
```

```
            return (membership);
        }
```

Conclusion

All of the code snippets included in this document at included in a sample REST Demo C# application available on GitHub. There is a README.html included that talks more specifically about building and running the code. Feel free to review the code and run it against a test or development Learn instance to see how it works.