Program.cs file:

```
using Microsoft.Identity.Client;
using Newtonsoft.Json.Linq;
using RestSharp;
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

namespace BusinessCentralAPI
{
    class Program
    {
        static string m_PublicToken = string.Empty;
        static string m_ServiceToken = string.Empty;
        static string m_Token = string.Empty;
        static bool m_UsePublicToken = false;
        static string m_PressAnyKey = "Press any key to continue. Esc to cancel.";
        static bool m_Verbose = Properties.Settings.Default.Verbose;

        static void Main(string[] args)
        {
            var packageCode = "INNOVIATEST";
            var packageName = "Innovia Test";

            //0
            DeleteConfigPackage(packageCode);

            // 1
            InsertConfigPackage(packageCode, packageName);

            // 2 Information only.
            // This app will filter the config packages by packageCode
            // for steps 3 through 5 and use the ConfigPackageId found to use in the urls
            GetConfigPackage(packageCode);

            // 3
            var path = Properties.Settings.Default.RapidStartFilePath;
            UploadConfigPackage(packageCode, path);
            // 4
            ImportConfigPackage(packageCode);
            var sleep = !CheckConfigStatus(packageCode);
            TimeSpan interval = new TimeSpan(0, 0, Properties.Settings.Default.SleepInterval);

            int i = 0;
            while (sleep)
            {
                sleep = !CheckConfigStatus(packageCode);
                if (sleep)
                {
                    i += 1;
                    sleep = i < Properties.Settings.Default.MaximumSleeps;
                }
                Thread.Sleep(interval);
            }
            // 5
            ApplyConfigPackage(packageCode);
            CheckConfigStatus(packageCode);

            //Check token use for non-automation APIs.
            ModifyCustomer("Innovia Test");

            Console.Read();
        }
```

```
        private static string GetServiceToken()
        {
            var scopes = new string[] { $"{Properties.Settings.Default.ApiBaseUrl}/.default" };
            SetConfidentialTokenAsync(scopes).GetAwaiter().GetResult();
            return m_ServiceToken;
        }

        private static async Task SetConfidentialTokenAsync(string[] scopes)
        {
            //AuthenticationConfig config = AuthenticationConfig.ReadFromJsonFile("appsettings.json");

            // You can run this sample using ClientSecret or Certificate. The code will differ only when
instantiating the IConfidentialClientApplication

            // Even if this is a console application here, a daemon application is a confidential client
application

            m_ServiceToken = string.Empty;

            IConfidentialClientApplication app;

            app = ConfidentialClientApplicationBuilder.Create(Properties.Settings.Default.ClientId)
                .WithClientSecret(Properties.Settings.Default.ClientSecret)
                .WithAuthority(new Uri(String.Format("https://login.microsoftonline.com/{0}",
Properties.Settings.Default.TenantId)))
                .WithRedirectUri(Properties.Settings.Default.RedirectUrl)
                .Build();

            AuthenticationResult result = null;
            try
            {
                result = await app.AcquireTokenForClient(scopes)
                    .ExecuteAsync();
                Console.ForegroundColor = ConsoleColor.Green;
                Console.WriteLine("Token acquired");
                Console.ResetColor();
            }
            catch (MsalServiceException ex) when (ex.Message.Contains("AADSTS70011"))
            {
                // Invalid scope. The scope has to be of the form "https://resourceurl/.default"
                // Mitigation: change the scope to be as expected
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("Scope provided is not supported");
                Console.ResetColor();
            }

            if (result != null)
            {
                m_ServiceToken = result.AccessToken;
            }
        }

        public static string GetAccessToken()
        {
            if (m_Token != string.Empty)
                return m_Token;
            if (!m_UsePublicToken)
            {
                m_Token = GetServiceToken();
            }
            else
            {
                m_Token = GetAccessTokenPublic();
            }
            return m_Token;
        }

        public static string GetAccessTokenPublic()
        {
```

```
            if (m_PublicToken != string.Empty)
            {
                return m_PublicToken;
            }

            var tenantId = Properties.Settings.Default.TenantId;
            var clientId = Properties.Settings.Default.ClientId; ;
            var authorityUri = String.Format("https://login.microsoftonline.com/{0}/oauth2/v2.0/token",
tenantId);
            var redirectUri = "https://login.microsoftonline.com/common/oauth2/nativeclient";
            var scopes = new List<string> { "https://api.businesscentral.dynamics.com/.default" };

            var publicClient = Microsoft.Identity.Client.PublicClientApplicationBuilder
                        .Create(clientId)
                        .WithAuthority(new Uri(authorityUri))
                        .WithRedirectUri(redirectUri)
                        .Build();

            var accessTokenRequest = publicClient.AcquireTokenInteractive(scopes);
            var accessToken = accessTokenRequest.ExecuteAsync().Result.AccessToken;


            m_PublicToken = accessToken;
            if (!String.IsNullOrEmpty(m_PublicToken))
            {
                Console.ForegroundColor = ConsoleColor.Green;
                Console.WriteLine("Token acquired");
                Console.ResetColor();
            }
            return accessToken;
        }

        static public string GetFileEtag(string jsonString)
        {
            var parsed = JObject.Parse(jsonString);

            if (!parsed.ContainsKey("file"))
                return string.Empty;

            var file = parsed.SelectToken("file").Value<JArray>();

            return GetEtagFromJsonObject(((JObject)file[0]).ToString());
        }

        static public string GetEtagFromJsonObject(string jsonString)
        {
            var newJson = jsonString.Replace("@odata.etag", "etag");
            var parsed = JObject.Parse(newJson);

            if (!parsed.ContainsKey("etag"))
                return string.Empty;

            var etag = parsed.SelectToken("etag").Value<string>();
            if (string.IsNullOrEmpty(etag))
                return string.Empty;

            return etag;
        }

        static void GetConfigPackage(string code)
        {
            try
            {

                string apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages?$filter=code eq
'{code}'";
                var apiCaller = new ProtectedApiCallHelper();
```

```
                ProtectedApiCallHelper.ApiCallDisplayOnly(apiUrl, Method.GET, "application/json", null,
null, GetAccessToken(), true);
            }
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }

            Console.WriteLine();
            Console.WriteLine(m_PressAnyKey);
            Console.WriteLine();
            var keyInfo = Console.ReadKey(true);
            var key = keyInfo.Key;
            if (key.ToString().ToLower() == "escape")
                Environment.Exit(0);
        }

        static string GetConfigPackageId(string code)
        {
            try
            {
                string apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages?$filter=code eq
'{code}'";
                var result = ProtectedApiCallHelper.ApiCallReturnJson(apiUrl, Method.GET,
"application/json", null, null, GetAccessToken(), m_Verbose);

                if (!result.ContainsKey("value"))
                    return string.Empty;

                var value = result.SelectToken("value").Value<JArray>();
                if (value.Count == 0)
                    return string.Empty;

                var configJson = (JObject)value[0];

                if (!configJson.ContainsKey("id"))
                    return string.Empty;

                var id = configJson.SelectToken("id").Value<string>();
                if (string.IsNullOrEmpty(id))
                    return string.Empty;

                return id;
            }
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }
            return string.Empty;
        }
       static string GetConfigImportStatus(JObject json)
        {
            try
            {
                if (!json.ContainsKey("importStatus"))
                    return string.Empty;

                var status = json.SelectToken("importStatus").Value<string>();
                if (string.IsNullOrEmpty(status))
                    return string.Empty;

                return status;
            }
```

```
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }
            return string.Empty;
        }

        static string GetConfigApplyStatus(JObject json)
        {
            try
            {
                if (!json.ContainsKey("applyStatus"))
                    return string.Empty;

                var status = json.SelectToken("applyStatus").Value<string>();
                if (string.IsNullOrEmpty(status))
                    return string.Empty;

                return status;
            }
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }
            return string.Empty;
        }
        static void InsertConfigPackage(string code, string packageName)
        {
            try
            {
                string apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages";
                JObject insertJson = new JObject();
                insertJson.Add("code", code);
                ProtectedApiCallHelper.ApiCallDisplayOnly(apiUrl, Method.POST, "application/json",
insertJson, null, GetAccessToken(), m_Verbose);
            }
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }

            Console.WriteLine();
            Console.WriteLine(m_PressAnyKey);
            Console.WriteLine();
            var keyInfo = Console.ReadKey(true);
            var key = keyInfo.Key;
            if (key.ToString().ToLower() == "escape")
                Environment.Exit(0);

        }

       static void UploadConfigPackage(string code, string filePath)
        {
            try
            {
                if (CheckConfigStatus(code))
                {
                    var configPackageId = GetConfigPackageId(code);
                    var etag = "*";//GetFileEtag(ProtectedApiCallHelper.ApiCallReturnJson(etagApiUrl,
Method.GET, "application/json", null, null, GetAccessToken(), m_Verbose).ToString());
```

```
                    //alternate etag logic
                    //var etagApiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages({configPackageI
d})?$expand=file";
                    //var etag = GetFileEtag(ProtectedApiCallHelper.ApiCallReturnJson(etagApiUrl,
Method.GET, "application/json", null, null, GetAccessToken(), m_Verbose).ToString());

                    //FileStream fsSource = new FileStream(filePath, FileMode.Open, FileAccess.Read);
                    // Read the source file into a byte array.
                    byte[] bytes = File.ReadAllBytes(filePath);

                    var apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages({configPackageI
d})/file('{code}')/content";

                    ProtectedApiCallHelper.ApiCallDisplayOnly(apiUrl, Method.PATCH, "application/octet-
stream", bytes, etag, GetAccessToken(), true);
                }
                else
                {
                    Console.WriteLine("Previous Import or Apply still processing.");
                }
            }
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }
            Console.WriteLine();
            Console.WriteLine(m_PressAnyKey);
            Console.WriteLine();
            var keyInfo = Console.ReadKey(true);
            var key = keyInfo.Key;
            if (key.ToString().ToLower() == "escape")
                Environment.Exit(0);
        }

        private static bool CheckConfigStatus(string code)
        {
            var configPackageId = GetConfigPackageId(code);
            string apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages({configPackageI
d})";
            var result = ProtectedApiCallHelper.ApiCallReturnJson(apiUrl, Method.GET, "application/json",
null, null, GetAccessToken(), m_Verbose);

            var importStatus = GetConfigImportStatus(result);
            if (m_Verbose)
            {
                Console.ForegroundColor = ConsoleColor.DarkYellow;
                Console.WriteLine($"Import Status: {importStatus}");
                Console.ResetColor();
            }
            if (importStatus != "Completed" & importStatus != "No")
            {
                return false;
            }

            var applyStatus = GetConfigApplyStatus(result);
            if (m_Verbose)
            {
                Console.ForegroundColor = ConsoleColor.DarkYellow;
                Console.WriteLine($"Apply Status: {applyStatus}");
                Console.ResetColor();
            }
```

```
            if (applyStatus != "Completed" & applyStatus != "No")
            {
                return false;
            }
            return true;
        }
        private static void ImportConfigPackage(string code)
        {
            try
            {
                if (CheckConfigStatus(code))
                {
                        var configPackageId = GetConfigPackageId(code);
                        var apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages({configPackageI
d})/Microsoft.NAV.import";
                        ProtectedApiCallHelper.ApiCallDisplayOnly(apiUrl, Method.POST, null, null, null,
GetAccessToken(), true);
                }
                else
                {
                    Console.WriteLine("Previous Import or Apply still processing.");
                }
            }
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }

            Console.WriteLine();
            Console.WriteLine(m_PressAnyKey);
            Console.WriteLine();
            var keyInfo = Console.ReadKey(true);
            var key = keyInfo.Key;
            if (key.ToString().ToLower() == "escape")
                Environment.Exit(0);
        }
        private static void ApplyConfigPackage(string code)
        {
            try
            {
                if (CheckConfigStatus(code))
                {
                    var configPackageId = GetConfigPackageId(code);
                    var apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages({configPackageI
d})/Microsoft.NAV.apply";
                    ProtectedApiCallHelper.ApiCallDisplayOnly(apiUrl, Method.POST, null, null, null,
GetAccessToken(), true);
                }
                else
                {
                    Console.WriteLine("Previous Import or Apply still processing.");
                }
            }
            catch (Exception ex)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(ex.Message);
                Console.ResetColor();
            }

            Console.WriteLine();
            Console.WriteLine(m_PressAnyKey);
            Console.WriteLine();
```

```
            var keyInfo = Console.ReadKey(true);
            var key = keyInfo.Key;
            if (key.ToString().ToLower() == "escape")
                Environment.Exit(0);
        }

       private static void DeleteConfigPackage(string code)
        {
            var configPackageId = GetConfigPackageId(code);
            if (!String.IsNullOrEmpty(configPackageId))
            {
                var apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/microsof
t/automation/v2.0/companies({Properties.Settings.Default.CompanyId})/configurationPackages({configPackageI
d})";
                try
                {
                    ProtectedApiCallHelper.ApiCallDisplayOnly(apiUrl, Method.DELETE, null, null, null,
GetAccessToken(), m_Verbose);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                }
            }
        }

        private static string GetCustomerId(string name)
        {
            var apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/v2.0/com
panies({Properties.Settings.Default.CompanyId})/customers?$filter=displayName eq '{name}'";
            var result = ProtectedApiCallHelper.ApiCallReturnJson(apiUrl, Method.GET, "application/json",
null, null, GetAccessToken(), m_Verbose);

            if (!result.ContainsKey("value"))
                return string.Empty;

            var value = result.SelectToken("value").Value<JArray>();
            if (value.Count == 0)
                return string.Empty;

            var configJson = (JObject)value[0];

            if (!configJson.ContainsKey("id"))
                return string.Empty;

            var id = configJson.SelectToken("id").Value<string>();
            if (string.IsNullOrEmpty(id))
                return string.Empty;

            return id;
        }

        private static void ModifyCustomer(string name)
        {
            var customerId = GetCustomerId(name);
            var apiUrl =
$"{Properties.Settings.Default.ApiBaseUrl}/v2.0/{Properties.Settings.Default.EnvironmentName}/api/v2.0/com
panies({Properties.Settings.Default.CompanyId})/customers({customerId})";
            var etag = "*";

            JObject patchJson = new JObject();
            patchJson.Add("addressLine1", "New Address 1234");
            try
            {
                ProtectedApiCallHelper.ApiCallDisplayOnly(apiUrl, Method.PATCH, "application/json",
patchJson, etag, GetAccessToken(), true);
            }
```

```
            catch (Exception e)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(e.Message);
                Console.ResetColor();
            }

            Console.WriteLine();
            Console.WriteLine(m_PressAnyKey);
            Console.WriteLine();
            var keyInfo = Console.ReadKey(true);
            var key = keyInfo.Key;
            if (key.ToString().ToLower() == "escape")
                Environment.Exit(0);

        }
    }
}
```

ProtectedAPICallHelper.cs:

```
/*
 The MIT License (MIT)

Copyright (c) 2015 Microsoft Corporation

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using RestSharp;
using System;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace BusinessCentralAPI
{
    /// <summary>
    /// Helper class to call a protected API and process its result
    /// </summary>
    ///
    public class ProtectedApiCallHelper
    {

        public string ResponseContent { get; set; }
        public JObject ResponseJson { get; set; }
        /// <summary>
```

```
        /// Constructor
        /// </summary>
        /// <param name="httpClient">HttpClient used to call the protected API</param>
        public ProtectedApiCallHelper()
        {
        }

        public async Task CallWebApiAndProcessResultASync(string webApiUrl, Method method, string
bodyContentType, Object body,string etag, string accessToken, bool writeJsonToConsole,Action<JObject>
processResult)
        {
            if (!string.IsNullOrEmpty(accessToken))
            {
                var client = new RestClient(webApiUrl);
                client.Timeout = -1;

                var request = new RestRequest(method);
                request.AddHeader("Authorization", "Bearer " + accessToken);
                if (method == Method.PATCH & !string.IsNullOrEmpty(etag))
                {
                    request.AddHeader("If-Match", etag);
                }
                if (!string.IsNullOrEmpty(bodyContentType))
                {
                    request.AddHeader("Content-Type", bodyContentType);
                    if (body != null)
                    {
                        request.AddParameter(bodyContentType, body, ParameterType.RequestBody);
                    }
                }

                System.Threading.CancellationToken cancellationToken =
default(System.Threading.CancellationToken);
                IRestResponse response = await client.ExecuteAsync(request, method, cancellationToken);

                ResponseContent = response.Content;
                ResponseJson = JsonConvert.DeserializeObject(ResponseContent) as JObject;

                if (response.IsSuccessful)
                {
                    if (writeJsonToConsole)
                    {
                        Console.WriteLine();
                        Console.WriteLine($"{method.ToString()}: {webApiUrl}");
                        if(bodyContentType != null & body != null & writeJsonToConsole)
                        {
                            if (bodyContentType.ToLower() == "application/json")
                            {
                                if (body.GetType() == typeof(JObject))
                                {
                                    var json = (JObject)body;
                                    Display(json);
                                    Console.ResetColor();
                                }
                            }
                        }
                        Console.ForegroundColor = ConsoleColor.Green;
                        Console.WriteLine("Successful");
                        if (!string.IsNullOrEmpty(ResponseContent))
                        {
                            Console.WriteLine();
                            processResult(ResponseJson);
                            Console.WriteLine();
                        }
                    }
                }
                else
                {
                    Console.WriteLine();
```

```
                    Console.WriteLine($"{method.ToString()}: {webApiUrl}");
                    if (bodyContentType != null)
                    {
                        if (bodyContentType.ToLower() == "application/json" & writeJsonToConsole)
                        {
                            Display((JObject)body);
                            Console.ResetColor();
                        }
                    }
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine($"Failed to call the Web Api: {response.StatusCode}");
                    // Note that if you got reponse.Code == 403 and reponse.content.code ==
"Authorization_RequestDenied"
                    // this is because the tenant admin as not granted consent for the application to call
the Web API
                    Console.WriteLine($"Content: {ResponseContent}");
                    Console.WriteLine();

                }
                Console.ResetColor();
            }
        }

        public static async Task ApiCallAsync(string ApiUrl, Method method, string bodyContentType, Object
body, string etag, string accessToken,bool writeJsonToConsole ,ProtectedApiCallHelper apiCaller)
        {
            if (!string.IsNullOrEmpty(accessToken))
            {
                await apiCaller.CallWebApiAndProcessResultASync(ApiUrl, method, bodyContentType, body,
etag, accessToken, writeJsonToConsole,Display);
            }
        }

        public static void ApiCallDisplayOnly(string ApiUrl, Method method, string bodyContentType, Object
body, string etag, string accessToken, bool writeJsonToConsole)
        {
            if (!string.IsNullOrEmpty(accessToken))
            {
                var apiCaller = new ProtectedApiCallHelper();
                ProtectedApiCallHelper.ApiCallAsync(ApiUrl, method, bodyContentType, body, etag,
accessToken, writeJsonToConsole, apiCaller).GetAwaiter().GetResult();
            }
        }

        public static JObject ApiCallReturnJson(string ApiUrl, Method method, string bodyContentType,
Object body, string etag, string accessToken, bool writeJsonToConsole)
        {
            if (!string.IsNullOrEmpty(accessToken))
            {
                var apiCaller = new ProtectedApiCallHelper();
                ProtectedApiCallHelper.ApiCallAsync(ApiUrl, method, bodyContentType, body, etag,
accessToken, writeJsonToConsole, apiCaller).GetAwaiter().GetResult();
                return apiCaller.ResponseJson;
            }
            return new JObject();
        }

        public static string ApiCallReturnString(string ApiUrl, Method method, string bodyContentType,
Object body, string etag, string accessToken, bool writeJsonToConsole)
        {
            if (!string.IsNullOrEmpty(accessToken))
            {
                var apiCaller = new ProtectedApiCallHelper();
                ProtectedApiCallHelper.ApiCallAsync(ApiUrl, method, bodyContentType, body, etag,
accessToken, writeJsonToConsole, apiCaller).GetAwaiter().GetResult();
                return apiCaller.ResponseContent;
            }
            return string.Empty;
        }
```

```
        public static void Display(JObject result)
        {
            Console.ForegroundColor = ConsoleColor.Blue;
            if (result == null)
                return;
            foreach (JProperty child in result.Properties().Where(p => !p.Name.StartsWith("@")))
            {
                Console.WriteLine($"{child.Name} = {child.Value}");
            }
        }
    }
}
```