OAuth2

OAuth 2.0 in the client library

Featured

Updated Oct 23, 2013 by pele...@google.com

An overview on using OAuth2.0 for authentication:

Overview

Use OAuth 2.0 to access to protected data stored on Google APIs.

Google APIs support a variety of flows designed to support different types of client applications. With all of these flows the client application requests an access token that is associated with only your client application and the owner of the protected data being accessed. The access token is also associated with a limited scope that define the kind of data the your client application has access to (for example "Manage your tasks"). An important goal for OAuth 2.0 is to provide secure and convenient access to the protected data, while minimizing the potential impact if an access token is stolen.

Further Reading

1. For general information about OAuth 2.0, see the OAuth 2.0 specification.
2. For information about using OAuth 2.0 to access Google APIs, see Using OAuth 2.0 to Access Google APIs.

Google APIs Console

You first need to register your client application with the Google apis console.

The first time you visit the console you need to

1. Create a new "project" by clicking "Create project...".
2. In the "Services" tab, you need to activate the Google APIs your client application uses (click on the button in the "Status" column) and agree to their terms of service.
3. Click on "API Access". The information you need depends on your application's API:
   1. For unauthenticated access to an API, you only need the "API key" under "Simple API Access". Skip the rest of these instructions.
   2. Otherwise, click on "Create an OAuth 2.0 Client ID..." and follow the instructions. When completed, click "Create client ID".
   3. For **"Installed applications"**, you will need the "Client ID" and "Client secret".
   4. For **"Web Applications"**, you will need the "Client ID" and "Client secret", as well as the "Redirect URIs".
   5. For **"Service Accounts"**, click on "Download private key" and store the p12 file with your web application. You will need the "Email address".

What you do with (or whether you even need) the client ID & secret, redirect URIs, or private key depends a lot on the specific flow you are using. So you need to take a careful look below at the code snippets for the flow you are interested for more specific instructions.

UserCredential and ServiceAccountCredential

UserCredential is a thread-safe helper class for OAuth2 access of protected resources using an access token.

An access token typically has an expiration date of 1 hour, after which you will get an error if you try to use. UserCredential and AuthorizationCodeFlow take care of automatically "refreshing" the token, which simply means getting a new access token. This is done using a long-lived refresh token, which is typically received along with the access token if you use the "access_type=offline" parameter during the authorization code flow.

Most applications will need to persist the credential's access token and refresh token. Otherwise, you will need to present the end user with authorization page in the browser every time your access token expires an hour after you've received it.

To persist the credential's access and refresh tokens, you can provide your own implementation of IDataStore, or you can use one of the following implementations provided by the library:

1. FileDataStore (from GoogleApis.DotNet4): persists the credential in a file.
2. StorageDataStore (from GoogleApis.WP): persists the credential in using the WP's StorageFolder
3. StorageDataStore (from GoogleApis.WinRT): persists the credential in using the Windows 8' StorageFolder

ServiceAccountCredential is similar to UserCredential, but is servers a different purpose. Google OAuth 2 supports server-to-server interactions such as those between a web application and Google Cloud Storage. The requesting application has to prove its own identity to gain access to an API, and an end-user doesn't have to be involved. Read more here.

ServiceAccountCredential stores a private key which is used to sign a request to get a new access token. Both UserCredential and

ServiceAccountCredential implement IConfigurableHttpClientInitializer so they will be able to register themselves as: - Unsuccessful response handler so it will refresh the token on 401 HTTP status code - Interceptor to intercepts the “Authorization” header on every request

Installed Applications

Sample code using the Calendar Service:

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

using Google.Apis.Auth.OAuth2;
using Google.Apis.Books.v1;
using Google.Apis.Books.v1.Data;
using Google.Apis.Services;
using Google.Apis.Util.Store;

namespace Books.ListMyLibrary
{
    /// <summary>
    /// Sample which demonstrates how to use the Books API.
    /// https://code.google.com/apis/books/docs/v1/getting_started.html
    /// </summary>
    internal class Program
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("Books API Sample: List MyLibrary");
            Console.WriteLine("================================");

            try
            {
                new Program().Run().Wait();
            }
            catch (AggregateException ex)
            {
                foreach (var e in ex.InnerExceptions)
                {
                    Console.WriteLine("ERROR: " + e.Message);
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        private async Task Run()
        {
            UserCredential credential;
            using (var stream = new FileStream("client_secrets.json", FileMode.Open, FileAccess.Read))
            {
                credential = await GoogleWebAuthorizationBroker.AuthorizeAsync(
                    GoogleClientSecrets.Load(stream).Secrets,
                    new[] { BooksService.Scope.Books },
                    "user", CancellationToken.None, new FileDataStore("Books.ListMyLibrary"));
            }

            // Create the service.
            var service = new BooksService(new BaseClientService.Initializer()
                {
                    HttpClientInitializer = credential,
                    ApplicationName = "Books API Sample",
                });

            var bookselve = await service.Mylibrary.Bookshelves.List().ExecuteAsync();
...
        }
    }
}
```

1. We create a new UserCredential instance by calling to GoogleWebAuthorizationBroker.AuthorizeAsync method. This static method gets the following:

1. The client secret (or a stream to the client secret)
2. The required scopes
3. The user identifier
4. The cancellation token for cancelling an operation
5. And an optional data store. If the data store is not specifies, the default FileDataStore with a default “Google.Apis.Auth” folder (the folder is created in Environment.SpecialFolder.ApplicationData.

2. We set the output user credential as a HttpClientInitializer on the Books service (using the initializer). As explained above user credential implements HTTP client initializer.

3. Notice that in this sample we choose to load the client secrets from a file but you can do the following as well:

```
        credential = await GoogleWebAuthorizationBroker.AuthorizeAsync(
                  new ClientSecrets
                  {
                      ClientId = "PUT_CLIENT_ID_HERE",
                      ClientSecret = "PUT_CLIENT_SECRETS_HERE"
                  },
                  new[] { BooksService.Scope.Books },
                  "user", CancellationToken.None, new FileDataStore("Books.ListMyLibrary"));
```

Take a look in our Books sample.

Web Applications (using ASP.NET MVC)

After adding the right Google.Apis NuGet package for working with Drive, YouTube or whatever service you want to work with, you should **add the Google.Apis.Auth.MVC package**.

The following code demonstrate a ASP.NET MVC application that query a Google API service.

1. Add your own implementation of FlowMetadata

```
using System;
using System.Web.Mvc;

using Google.Apis.Auth.OAuth2;
using Google.Apis.Auth.OAuth2.Flows;
using Google.Apis.Auth.OAuth2.Mvc;
using Google.Apis.Drive.v2;
using Google.Apis.Util.Store;

namespace Test.MVC4
{
    public class AppFlowMetadata : FlowMetadata
    {
        private static readonly IAuthorizationCodeFlow flow =
            new GoogleAuthorizationCodeFlow(new GoogleAuthorizationCodeFlow.Initializer
                {
                    ClientSecrets = new ClientSecrets
                    {
                        ClientId = "PUT_CLIENT_ID_HERE",
                        ClientSecret = "PUT_CLIENT_SECRET_HERE"
                    },
                    Scopes = new[] { DriveService.Scope.Drive },
                    DataStore = new FileDataStore("Drive.Api.Auth.Store")
                });

        public override string GetUserId(Controller controller)
        {
            // In this sample we use the session to store the user identifiers.
            // That's not the best practice, because you should manage your users accounts.
            // We consider providing an "OpenId Connect" library for helping you doing that.
            var user = controller.Session["user"];
            if (user == null)
            {
                user = Guid.NewGuid();
                controller.Session["user"] = user;
            }
            return user.ToString();

        }

        public override IAuthorizationCodeFlow Flow
        {
            get { return flow; }
        }
    }
}
```

FlowMetaData is an abstract class that contains your own logic of retrieving the user identifier and the IAuthorizationCodeFlow you are using. Here we create a new GoogleAuthorizationCodeFlow and specify the scopes, client secrets and the data store we use. Consider add your own implementation of DataStore which uses EntityFramework for example. For .NET 4 application we provide only a file data store.

2. Now you should implement your own controller which uses the drive service for example:

```
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Web.Mvc;

using Google.Apis.Auth.OAuth2.Mvc;
using Google.Apis.Drive.v2;
using Google.Apis.Services;

using Test.MVC4;

namespace Test.MVC4.Controllers
{
    public class HomeController : Controller
```

```
    {
        public async Task<ActionResult> IndexAsync(CancellationToken cancellationToken)
        {
            var result = await new AuthorizationCodeMvcApp(this, new AppFlowMetadata()).
                AuthorizeAsync(cancellationToken);

            if (result.Credential != null)
            {
                var service = new DriveService(new BaseClientService.Initializer
                    {
                        HttpClientInitializer = result.Credential,
                        ApplicationName = "ASP.NET MVC Sample"
                    });

                // YOUR CODE SHOULD BE HERE..
                // SAMPLE CODE:
                var list = await service.Files.List().ExecuteAsync();
                ViewBag.Message = "FILES COUNT IS: " + list.Items.Count();
                return View();
            }
            else
            {
                return new RedirectResult(result.RedirectUri);
            }
        }
    }
}
```

3. You should implement your callback controller. The implementation will look something like that:

```
namespace Test.MVC4.Controllers
{
    public class AuthCallbackController : Google.Apis.Auth.OAuth2.Mvc.Controllers.AuthCallbackController
    {
        protected override Google.Apis.Auth.OAuth2.Mvc.FlowMetadata FlowData
        {
            get { return new AppFlowMetadata(); }
        }
    }
}
```

Service Account (available only on .NET framework 4)

Google APIs also support Service Accounts. Unlike the credential in which a client application requests access to an end-user's data, Service Accounts provide access to the client application's own data. Your client application signs the request for an access token using a private key downloaded from the Google APIs console. After creating a new client ID, you should choose a “Service Account” application type and then you can download the private key.

Sample code using Plus service. A working sample is available here

```
using System;
using System.Security.Cryptography.X509Certificates;

using Google.Apis.Auth.OAuth2;
using Google.Apis.Plus.v1;
using Google.Apis.Plus.v1.Data;
using Google.Apis.Services;

namespace Google.Apis.Samples.PlusServiceAccount
{
    /// <summary>
    /// This sample demonstrates the simplest use case for a Service Account service.
    /// The certificate needs to be downloaded from the APIs Console
    /// <see cref="https://code.google.com/apis/console/#:access"/>:
    ///   "Create another client ID..." -> "Service Account" -> Download the certificate as "key.p12" and replace the
    ///   placeholder.
    /// The schema provided here can be applied to every request requiring authentication.
    /// <see cref="https://developers.google.com/accounts/docs/OAuth2#serviceaccount"/> for more information.
    /// </summary>
    public class Program
    {
        // A known public activity.
        private static String ACTIVITY_ID = "z12gtjhq3qn2xxl2o224exwiqruvtda0i";

        public static void Main(string[] args)
        {
            Console.WriteLine("Plus API - Service Account");
            Console.WriteLine("===========================");

            String serviceAccountEmail = "SERVICE_ACCOUNT_EMAIL_HERE";

            var certificate = new X509Certificate2(@"key.p12", "notasecret", X509KeyStorageFlags.Exportable);

            ServiceAccountCredential credential = new ServiceAccountCredential(
               new ServiceAccountCredential.Initializer(serviceAccountEmail)
               {
                   Scopes = new[] { PlusService.Scope.PlusMe }
```

```
            }.FromCertificate(certificate));

            // Create the service.
            var service = new PlusService(new BaseClientService.Initializer()
            {
                HttpClientInitializer = credential,
                ApplicationName = "Plus API Sample",
            });

            Activity activity = service.Activities.Get(ACTIVITY_ID).Execute();
            Console.WriteLine("  Activity: " + activity.Object.Content);
            Console.WriteLine("  Video: " + activity.Object.Attachments[0].Url);

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }
    }
}
```

1. In this sample code we create ServiceAccountCredential. We set its required scopes and load the private key from the certificate using the FromCertificate method.

2. As all other samples code, we set the credential as HttpClientInitializer.

Windows Phone

The following sample code is very similar to the regular to .NET 4 application OAuth flow. The following sample creates a new DriveService using the GoogleWebAuthorizationBroker for WP. By default StorageDataStore will be used to store the access token and the refresh token.

```
using System;
using System.Windows;
using System.IO;
using System.Threading;
using System.Text;
using Microsoft.Phone.Controls;

using Google.Apis.Drive.v2;
using Google.Apis.Services;
using Google.Apis.Auth.OAuth2;

namespace Drive.WP.Sample
{
    public partial class MainPage : PhoneApplicationPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private async void Button_Click(object sender, RoutedEventArgs e)
        {
            var credential = await GoogleWebAuthorizationBroker.AuthorizeAsync(
                new FileStream("client_secrets.json", FileMode.Open, FileAccess.Read),
                new[] { DriveService.Scope.Drive },
                "user",
                CancellationToken.None);

            var initializer = new BaseClientService.Initializer()
            {
                HttpClientInitializer = credential,
                ApplicationName = "WP Drive Sample Application",
            };

            var service = new DriveService(initializer);
            var list = await service.Files.List().ExecuteAsync();
            StringBuilder allFiles = new StringBuilder();
            foreach (var file in list.Items)
            {
                allFiles.Append(file.Title + Environment.NewLine);
            }

            text.Text = allFiles.ToString();
        }
    }
}
```

Windows 8 Applications

You should already be familiar with the following code. It creates a new CalendarService using the GoogleWebAuthorizationBroker for Windows 8 application. By default StorageDataStore will be used to store the access token and the refresh token.

```
            var credential = await GoogleWebAuthorizationBroker.AuthorizeAsync(
                new Uri("ms-appx:///Assets/client_secrets.json"),
                new[] { Uri.EscapeUriString(CalendarService.Scope.Calendar) },
                "user", CancellationToken.None);
            var calendarService = new CalendarService(new BaseClientService.Initializer
```

```
            {
                HttpClientInitializer = credential,
                ApplicationName = "WinRT sample"
            });
        var calendarListResource = await calendarService.CalendarList.List().ExecuteAsync();
```