Subject: WAK

From: [redacted]

To:

Date: Sunday, January 17, 2016 8:35 PM

>>>>> Which classes/attributes are used for what?

Here is the README. Hopefully this explains a little more.

WEB API KIT (WAK)
by Haptix Games Inc

The Web API Kit gives you the freedom to interconnect your application with the world wide web primarly using the HTTP protocol. Lots of companies publish their API and make it available to HTTP clients. With WAK you can take advantage of those public endpoints and create your own client using Unity's WWW class or other HTTP enabling assets. Then distribute your client with your application and have the world's data at your fingertips.

Many times beginners struggle with the concept of storing and retrieving custom data. Often they turn to solutions that store the application information on the device using flat files or SQLlite. The video tutorials that accompany this asset demonstrate how to build your own API server using eXist-db and XML/JSON technologies.

Once developers become familiar with web technologies and Unity's WWW class, they begin experimenting with retrieving and posting text data. This results in something like this:

```
WWW httpOperation = new WWW("http://some.domain.com/textFile.txt");
yield return httpOperation;
string retrievedString = httpOperation.text;
```

That looks simple enough, but we are missing a couple features:
* error checking
* ability to timeout the operation
* ability to cancel the operation on user request
* simplified HTTP request construction, including HTTP headers, HTML Form fields, etc
* outbound/inbound data conversion and mapping

Don't worry, WAK comes to the rescue. The world of WAK revolves around HTTP operations that you define. An emphasis is placed on using .NET attributes on classes and fields. If you are not familiar with attributes then this might be a good time to look them up as well as attribute-oriented programming.

Let's try to rewrite the above flat file request in WAK. Create a class and inherit from 'hg.ApiWebKit.core.http.HttpOperation'. Let's add a few attributes to it from the 'hg.ApiWebKit.core.attributes' namespace. We are retrieving a file which means it is a HTTP GET operation. And we need to define the URL, so we will use the 'HttpPath' attribute.

Inside the class we will want to store the retrieved text data and do something with it. So let's define a public field that we

can access later. To make is super easy, we will add an attribute to the field from the 'hg.ApiWebKit.core.attributes' namespace.
The 'HttpResponseTextBody' attribute will let WAK know to take the response text value and assign it to the marked up field.

Ok so here is our first HTTP operation:

```
[hg.ApiWebKit.core.attributes.HttpGET]
[hg.ApiWebKit.core.attributes.HttpPath(null,"http://some.domain.com/textFile.txt")]
public class MyFirstWakOperation: hg.ApiWebKit.core.http.HttpOperation
{
[hg.ApiWebKit.core.attributes.HttpResponseTextBody]
public string RetrievedString;
}
```

So let's try to use this operation from one of our application's scripts.
Inside the Start method we will create and instance of the operation and call Send.
The Send method accepts 3 callback functions, similar to JavaScript and jQuery in web programming.
If the execution is successful then OnSuccess will be called, otherwise OnFailure will be called.
OnComplete fires regardless of success or failure.
The callback method signature has 2 parameters. The first one is of the HTTP operation type you are calling and the second one is 'core.http.HttpResponse' which gives you access to HTTP protocol information.

```
public class MyAppScript: MonoBehaviour
{
void Start()
{
  new MyFirstWakOperation.Send(OnSuccess,OnFailure,OnComplete);
}

private void OnSuccess(MyFirstWakOperation operation, core.http.HttpResponse response)
{
  Debug.Log("Success!");
  Debug.Log(operation.RetrievedString);
}

private void OnFailure(MyFirstWakOperation operation, core.http.HttpResponse response)
{
  Debug.Log("Failed");

  Debug.Log("Faulted because: " + string.Join(" ; ", operation.FaultReasons.ToArray()));
}

private void OnComplete(MyFirstWakOperation operation, core.http.HttpResponse response)
{
  Debug.Log("Completed with status code : " + response.StatusCode);
}
}
```

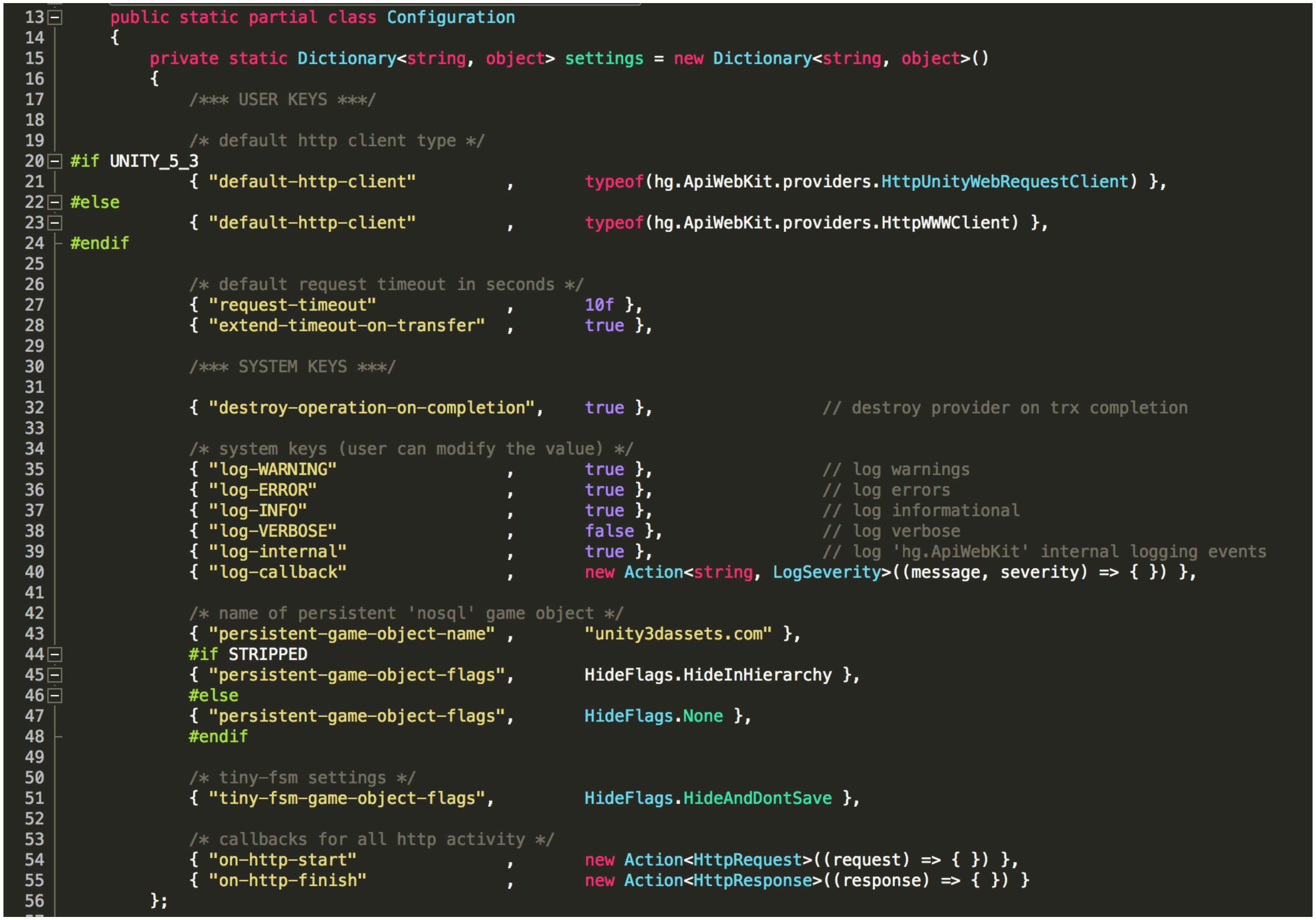
You have just created your first HTTP operation.
Don't forget to watch our tutorial series for more advanced topics.

For a practical example please take a look here: http://www.reddit.com/r/Unity3D/comments/2at4pa/www_and_rest_apis/

-Haptix Games Inc

>>>>> How can I set specific settings?

The asset doesn't have many general settings but everything is done through Configuration static class. These settings are not persisted and most can be overridden within specific HTTP operations.



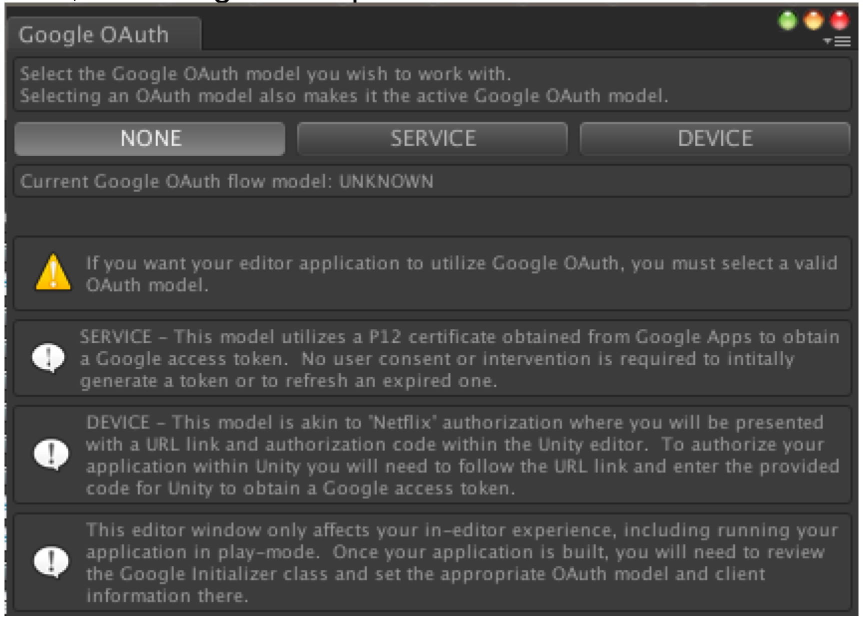
```
    public static partial class Configuration
    {
        private static Dictionary<string, object> settings = new Dictionary<string, object>()
        {
            /*** USER KEYS ***/

            /* default http client type */
#if UNITY_5_3
            { "default-http-client"         ,       typeof(hg.ApiWebKit.providers.HttpUnityWebRequestClient) },
#else
            { "default-http-client"         ,       typeof(hg.ApiWebKit.providers.HttpWWWClient) },
#endif

            /* default request timeout in seconds */
            { "request-timeout"             ,       10f },
            { "extend-timeout-on-transfer"  ,       true },

            /*** SYSTEM KEYS ***/

            { "destroy-operation-on-completion",    true },                 // destroy provider on trx completion

            /* system keys (user can modify the value) */
            { "log-WARNING"                 ,       true },                 // log warnings
            { "log-ERROR"                   ,       true },                 // log errors
            { "log-INFO"                    ,       true },                 // log informational
            { "log-VERBOSE"                 ,       false },                // log verbose
            { "log-internal"                ,       true },                 // log 'hg.ApiWebKit' internal logging events
            { "log-callback"                ,       new Action<string, LogSeverity>((message, severity) => { }) },

            /* name of persistent 'nosql' game object */
            { "persistent-game-object-name" ,       "unity3dassets.com" },
            #if STRIPPED
            { "persistent-game-object-flags",       HideFlags.HideInHierarchy },
            #else
            { "persistent-game-object-flags",       HideFlags.None },
            #endif

            /* tiny-fsm settings */
            { "tiny-fsm-game-object-flags",         HideFlags.HideAndDontSave },

            /* callbacks for all http activity */
            { "on-http-start"               ,       new Action<HttpRequest>((request) => { }) },
            { "on-http-finish"              ,       new Action<HttpResponse>((response) => { }) }
        };
```

>>>>> How can I use it to login via username and password?

Here is an example on basic auth.
http://haptixgames.com/forum/viewtopic.php?f=13&t=1086&start=10

>>>>> How can I use it to login via access token?

So the currenlty implemented OAuth flow is authorization code. This follows the flow you found in the PDF. Here is a screenshot of an interceptor.
Also, the Google examples have a JWT flow as well as a Netflix Bluray player type flow.


Google OAuth

Select the Google OAuth model you wish to work with.
Selecting an OAuth model also makes it the active Google OAuth model.

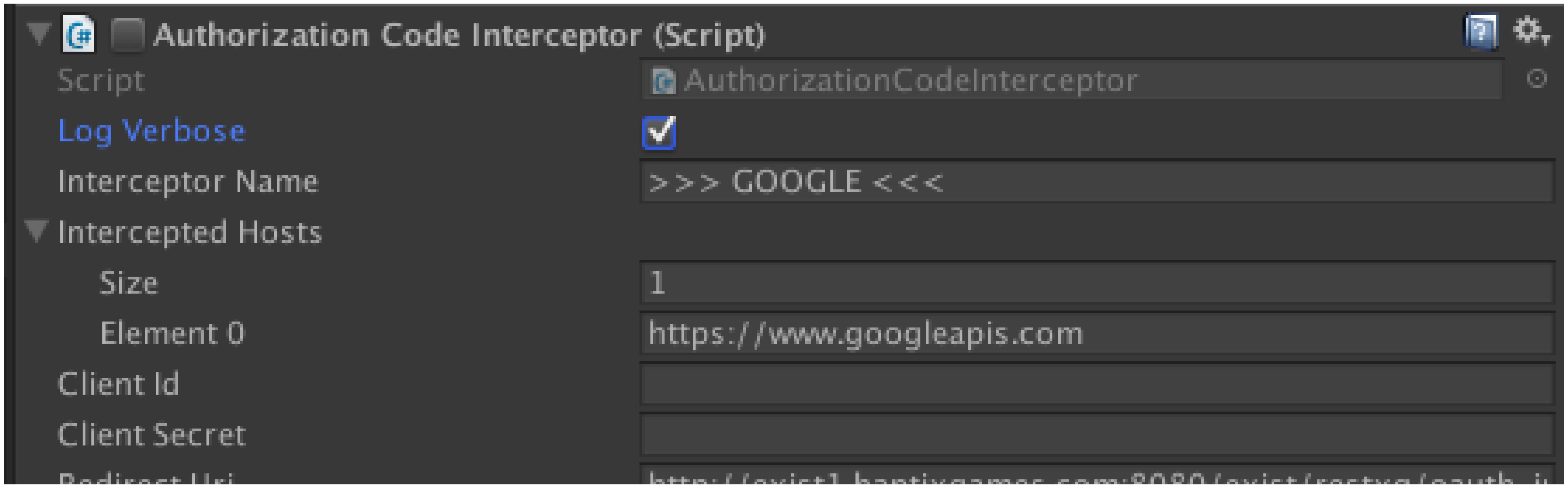
Current Google OAuth flow model: UNKNOWN

If you want your editor application to utilize Google OAuth, you must select a valid OAuth model.

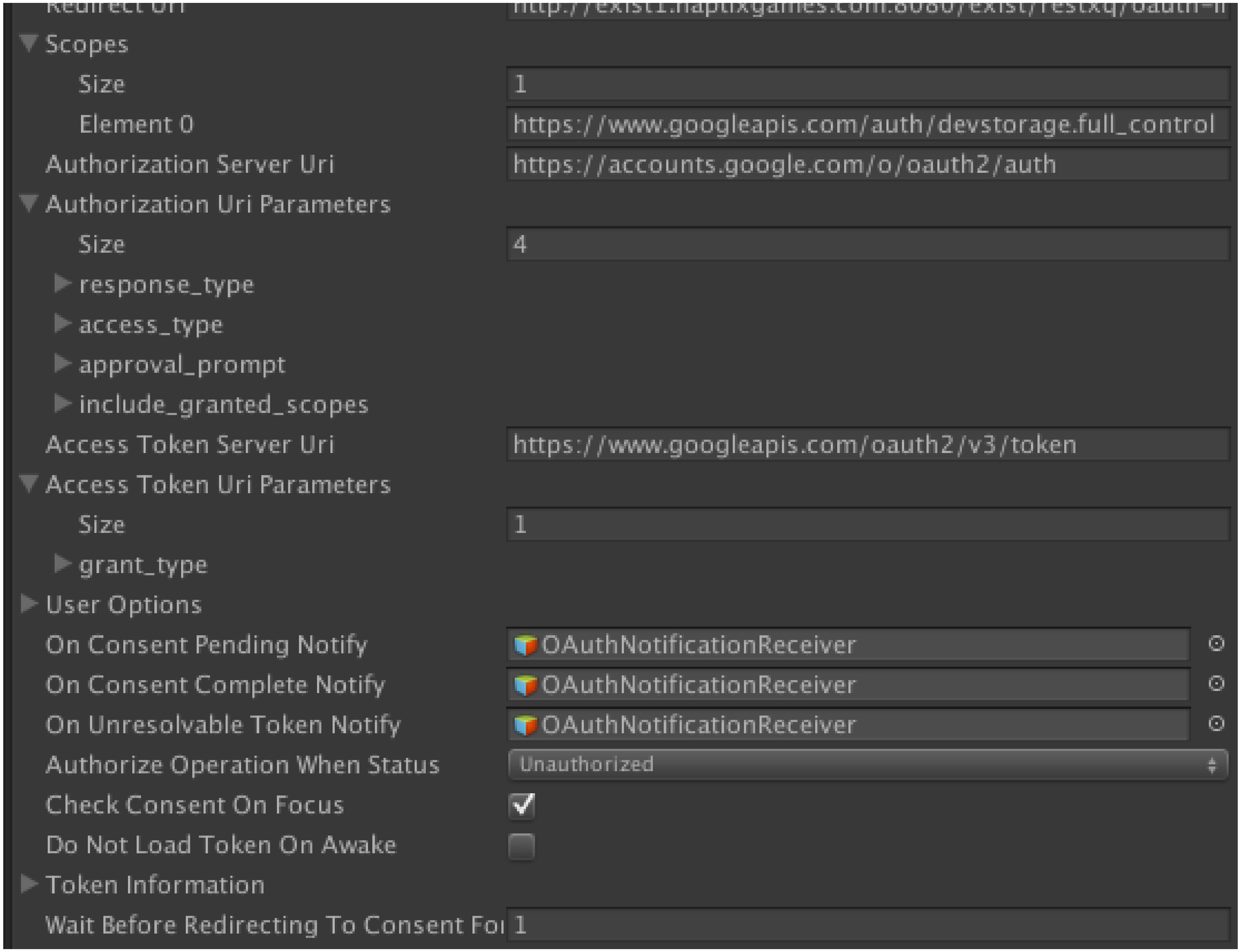
SERVICE - This model utilizes a P12 certificate obtained from Google Apps to obtain a Google access token. No user consent or intervention is required to intitally generate a token or to refresh an expired one.

DEVICE - This model is akin to 'Netflix' authorization where you will be presented with a URL link and authorization code within the Unity editor. To authorize your application within Unity you will need to follow the URL link and enter the provided code for Unity to obtain a Google access token.

This editor window only affects your in-editor experience, including running your application in play-mode. Once your application is built, you will need to review the Google Initializer class and set the appropriate OAuth model and client information there.


Authorization Code Interceptor (Script)

Script	AuthorizationCodeInterceptor
Log Verbose	✓
Interceptor Name	>>> GOOGLE <<<
Intercepted Hosts	
Size	1
Element 0	https://www.googleapis.com
Client Id	
Client Secret	
Redirect Uri	http://exist1.haptixgames.com:8080/exist/restxq/oauth_i



Redirect Uri	http://exist1.naptixgames.com:8080/exist/restxq/oauth-ll
▼ Scopes	
Size	1
Element 0	https://www.googleapis.com/auth/devstorage.full_control
Authorization Server Uri	https://accounts.google.com/o/oauth2/auth
▼ Authorization Uri Parameters	
Size	4
▶ response_type	
▶ access_type	
▶ approval_prompt	
▶ include_granted_scopes	
Access Token Server Uri	https://www.googleapis.com/oauth2/v3/token
▼ Access Token Uri Parameters	
Size	1
▶ grant_type	
▶ User Options	
On Consent Pending Notify	OAuthNotificationReceiver
On Consent Complete Notify	OAuthNotificationReceiver
On Unresolvable Token Notify	OAuthNotificationReceiver
Authorize Operation When Status	Unauthorized
Check Consent On Focus	✓
Do Not Load Token On Awake	☐
▶ Token Information	
Wait Before Redirecting To Consent For	1

>>>>> How is the access token refreshed?

When a token expires, the operation fails, usually with a 401 (configured in Interceptor 'Authorize Operation When Status').

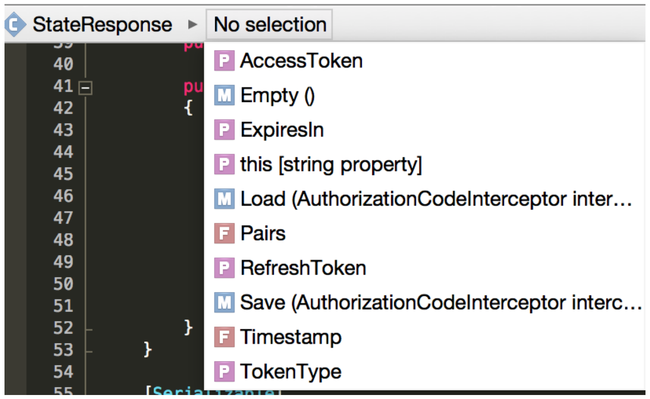
When this happens, the failed operation is queued for retry. Then the interceptor will try one of two things: use a refresh token to obtain a new access token or request consent (user login) to obtain a new access token. This was a very important consideration since Google tokens expire every 30 minutes. After an access token is obtained it is stored in PlayerPrefs and the failed operation is retried with the new token.

>>>>> Where is the access token and refresh token stored?

The tokens are stored in PlayerPrefs.

>>>>> How can I access them?

The interceptor provides a TokenInformation field that has access.



```
P AccessToken
M Empty ()
P ExpiresIn
P this [string property]
M Load (AuthorizationCodeInterceptor inter...
F Pairs
P RefreshToken
M Save (AuthorizationCodeInterceptor interc...
F Timestamp
P TokenType
```

>>>>> How do I setup the oauth process prior, so I do not have to care later?

You set up the interceptor in the editor.

>>>>> How do I set it up so when I request a resource the token is
>>>>> automatically refreshed or an event is raised to let the user
>>>>> login again via username and password?

If an access token or refresh token is missing then the interceptor will force Application.OpenURL to your consent page. There also custom actions available depending on what happens during a code<=>token exchange.

```
public class OAuthStatusReceiverSample: MonoBehaviour
{
  /*
```

```
   your project can have multiple AuthorizationCodeInterceptor behaviors, each one
for a different public API.
   when your projects enters runtime each interceptor registers itself.
   as each HTTP operation is executed, the host defined in the HttpPath attribute of
the HTTP operation determines whether to send it through the interceptor or not.
   not to break existing implementations, a Send2() extension method has been added
to HttpOperation class.
   you may use Send2() for all operations, intercepted or not.
   HttpOperations that will be intercepted must also carry a
[HttpBearerAuthrorization] attribute on its class.

   Here is an example of an interceptable operation:
    [HttpBearerAuthorization]
    [HttpPath(null,"https://www.googleapis.com/storage/v1/b")]
    [HttpGET]
    [HttpTimeout(10f)]
    [HttpProvider(typeof(hg.ApiWebKit.providers.HttpWWWClient))]
    public class ListBuckets : HttpOperation
    {
     [HttpQueryString("project")]
     public string projectId = "perfect-tape-656";
    }

   Here is how you call it:
    new
ListBuckets().Send2(on_start,on_success,on_failure,on_completion,custom_params);

   invocation of the Send2() method will place the operation on the appropriate
interceptor's queue
   and attempt to send it.  if the interceptor has no current token the consent
process will begin.
   if the interceptor does have a current token then it will attempt to execute the
operation.
   if the token is valid the operation will succeed.  if the token is not valid, the
operation will
   result in a 401 status code.  the operation will then be placed on a fail queue to
be retried at
   a later time.  the interceptor will now attempt to refresh the expired token and
```

```
reprocess any
   failed operations then continue to process any new incoming operations.  if a
refresh token is not found
   the consent process will be invoked.
  */

  public void OnConsentPending(ConsentRequest request)
  {
   Debug.Log ("[OAuthStatusReceiverSample] OnConsentPending called");

   /*
    this method is invoked when the OAuth Interceptor starts the consent process.
    this involves opening the native web browser to the consent form.
    the delay between calling this method and pausing the application to go to the
    consent form is determined by
AuthorizationCodeInterceptor.WaitBeforeRedirectingToConsentForm.
    during this time you may display additional information to the user.
   */
  }

  public void OnConsentComplete(ConsentResponse response)
  {
   Debug.Log ("[OAuthStatusReceiverSample] OnConsentComplete called");

   /*
    this method is invoked once the application returned from a paused state after a
pending consent.
    the response contains the access token information and the status of the consent
process.

    there are several possible outcomes of the consent flow:
    1.
     the browser opens up to the consent form and the user does nothing
     he simply returns to the application
     this will result in a 404 error

    2.
     the browser opens up to the consent form and the user clicks cancel
```

```
      then returns to the application
      this will result in a 1 error

     3.
      the browser opens up to the consent form and the user clicks approve
      then he returns to the application
      this will result in a 200 success or 403 error if the access token failed to be
retrieved

    other possible error code is a 410.  the application has only one chance to
redeem the access token from
    the redirect server.  after that any requests for the same state will result in a
410.

    upon a 1, 403, 404 error the OnUnresolvableToken will be called.
    there you can choose how to handle above failures.
   */
  }

  public void OnUnresolvableToken(AuthorizationCodeInterceptor interceptor)
  {
   Debug.Log ("[OAuthStatusReceiverSample] OnUnresolvableToken called");

   /*
    a token could fail to be obtained because of the conditions described in
OnConsentComplete.
    a failure could also be caused by missing information on the interceptor such as
clientid or secret,
    or any other information required but missing.  remember that OAuth is a
framework and each public API
    interprets it differently resulting in different implementations.

    you may choose to restart the consent process by invoking the following.
    Remember that after resolving an unresolvable token condition you must Resume()
the interceptor manually.
   */

   interceptor.RequestConsent(null, (response) => {
    if(response.Status==OAUTH_RESPONSE_STATUS.SUCCESS)
    {
     interceptor.Resume ();
    }
    else
    {
     // you got a big problem
    }
   });
  }
}
}
```

/c

Attachments

- Screen Shot 2016-01-17 at 8.20.36 PM.png (95.66KB)
- Screen Shot 2016-01-17 at 8.06.42 PM.png (127.06KB)
- Screen Shot 2016-01-17 at 8.29.20 PM.png (78.46KB)
- Screen Shot 2016-01-17 at 7.53.19 PM.png (434.00KB)