Documentation

Firebase for Unity WebGL

This is dedicated implementation of the most popular **Firebase** modules for Web apps and games made by **Unity Engine**.
Google does not provide support for the **Unity WebGL** plugin, so we decided to do it by ourselves.

What the package includes:

- **Editor scripts:** build post-process to made all required JavaScript code injections into `index.html` file.
- **Runtime scripts:** significant amount of **Firebase** modules that can be integrated in your project.
- **Demo scenes:** a demonstration scene how works **Firebase** modules

Table of content:

Installation

1. Importing the package into Unity project

- Open your Unity Project.
- Go to `Window → Package Manager → Firebase for WebGL`.
- Click **Download** button if you haven't downloaded package yet.

- Click **Import** button to select all the package's files to be imported.

2. Creating **Firebase** settings

- Open your Unity Project
- Select any `Resources` folder in `Project Window`
- Go to `Assets → Create → Firebase` and click `WebGL Settings`
- `Firebase.asset` will be created (attention: this file must be placed in any `Resources` folder)

3. Configuring **Firebase** settings

- Select `Firebase.asset` file in `Project Window`
- In the `[General]` section, fill the following text fields by data from your `Firebase` project: *API Key, Auth Domain, Project Id, Storage Bucket, Messaging Sender Id, App Id and Measurement Id/*.
- In the `[Options]` section, tick the checkboxes of the modules you want to enable. It means that only these selected modules will be included in the game build.

User Guide

All modules placed under `FirebaseWebGL` namespace.

Initialization

The main class is `FirebaseApp` class.

You can access the default instance of `FirebaseApp` using the following code:

```
var app = FirebaseWebGL.FirebaseApp.DefaultInstance();
```

Also you have to check that `FirebaseApp` instance is initialized:

```
if (!app.isInitialized)
{
    //Firebase SDK is not initialized
    //It means that Firebase JavaScript code not loaded in browser (please check the logs first)
    //CORS policies are often the cause of initialization failures.
    return;
}
```

Every module injected into `FirebaseApp` instance and have to be initialized.

Use the following code to initialize any module. For example:

```
IEnumerator Start()
{
    //It will be not null if the module injected successfully.
    if (_app.Analytics ! null)
    {
        bool? initialized = null;
        _app.Analytics.Initialize((callback) =>
        {
            if (callback.success == false)
            {
                Debug.LogError($"Initialize: {callback.error}");
                return;
            }
            initialized = callback.result;
         });
         yield return new WaitUntil(() => initialized != null);
         if (initialized.Value == false)
         {
             Debug.LogError("Initialize: not initialized");
             yield break;
         }
    }
}
```

Firebase Analytics

To set user id:

```
app.Analytics.SetUserId("YOUR_USER_ID");
```

To set user properties:

```
app.Analytics.SetUserProperties(new Dictionary<string, string>()
{
    { "property_name", "property_value" },
});
```

To set enable of analytics collection:

```
app.Analytics.SetAnalyticsCollectionEnabled(true);
```

To set custom default event parameters:

```
app.Analytics.SetDefaultEventParameters(new Dictionary<string, string>
()
{
    { "platform", Application.platform.ToString() },
    { "version", Application.version },
    { "unity_version", Application.unityVersion },
});
```

To set mass-consent for different consent types:

```
app.Analytics.SetConsent(new Dictionary<FirebaseAnalyticsConsentType,
FirebaseAnalyticsConsentValue>
{
    { FirebaseAnalyticsConsentType.AdPersonalization,
FirebaseAnalyticsConsentValue.Granted },
    { FirebaseAnalyticsConsentType.AdStorage,
FirebaseAnalyticsConsentValue.Granted },
    { FirebaseAnalyticsConsentType.AdUserData,
FirebaseAnalyticsConsentValue.Granted },
    { FirebaseAnalyticsConsentType.AnalyticsStorage,
FirebaseAnalyticsConsentValue.Granted },
    {
FirebaseAnalyticsConsentType.FunctionalityStorage,FirebaseAnalyticsCon
sentValue.Granted },
    { FirebaseAnalyticsConsentType.PersonalizationStorage,
FirebaseAnalyticsConsentValue.Granted },
    { FirebaseAnalyticsConsentType.SecurityStorage,
FirebaseAnalyticsConsentValue.Granted },
});
```

To log event without params:

```
app.Analytics.LogEvent("test_event_no_params");
```

To log event with any kind of params:

```
app.Analytics.LogEvent("test_event_1_params", new Dictionary<string,
object> { { "param1", "value1" } });
```

Full list of API methods you can find here:

https://firebase.google.com/docs/reference/js/analytics.md

Firebase AppCheck

To set token auto-refresh:

```
app.AppCheck.SetTokenAutoRefreshEnabled(true);
```

To subscribe to `OnToken` callback:

```
app.AppCheck.OnTokenChanged((token) =>
{
    //do something with "token" here
}));
```

To get limited-use token:

```
app.AppCheck.GetLimitedUseToken((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetLimitedUseToken: {callback.error}");
        return;
    }
    var limitedUseToken = callback.result;
    //do something with "limitedUseToken" here
});
```

To get token:

```
var forceRefresh = false;
app.AppCheck.GetToken(forceRefresh, (callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetToken: {callback.error}");
        return;
    }
    _appCheckToken = callback.result;
});
```

Full list of API methods you can find here:
https://firebase.google.com/docs/reference/js/app-check.md

Firebase Auth

To subscribe to `OnAuthStateChanged` callback that indicates about change of current user

```
app.Auth.OnAuthStateChanged((user) =>
{
    //do something with "user" here
});
```

To subscribe to `OnIdTokenChanged` callback:

```
app.Auth.OnIdTokenChanged((idToken) =>
{
    //do something with "idToken" here
});
```

To create user with email and password:

```
app.Auth.CreateUserWithEmailAndPassword(authEmail, authPassword,
(callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"CreateUserWithEmailAndPassword:
{callback.error}");
        return;
    }

    var user = callback.result.user;
    //do something with "user" here
});
```

To sign-in anonymously:

```
app.Auth.SignInAnonymously((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"SignInAnonymously: {callback.error}");
        return;
    }

    var user = callback.result.user;
    //do something with "user" here
}
```

To sign-in by email and password:

```
app.Auth.SignInWithEmailAndPassword(authEmail, authPassword,
(callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"SignInWithEmailAndPassword:
{callback.error}");
        return;
    }

    var user = callback.result.user;
    //do something with "user" here
});
```

To sign-in with custom provider, for example with `Sign-in with Google`:

```
var providerId = FirebaseAuthProviders.google;
app.Auth.SignInWithPopup(providerId, (callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"SignInWithPopup: providerId={providerId},
error={callback.error}");
        return;
    }

    var user = callback.result.user;
    //do something with "user" here
});
```

To get access logged user:

```
var user = app.Auth.loggedUser;
```

To get user’s id token:

```
user.GetIdToken(forceRefresh: true, (callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetIdToken: {callback.error}");
        return;
    }

    _authIdToken = callback.result;
    Debug.Log($"GetIdToken: token={_authIdToken}");
});
```

To reload user:

```
user.Reload((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"Reload: {callback.error}");
        return;
    }

    var reloaded = callback.result;
    Debug.Log($"Reload: reloaded={reloaded}");
});
```

To sign-out:

```
user.SignOut((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"SignOut: {callback.error}");
        return;
    }

    var signedOut = callback.result;
    Debug.Log($"SignOut: signedOut={signedOut}");
});
```

To update user’s email:

```
user.UpdateEmail("YOUR_USER_EMAIL", (callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"UpdateEmail: {callback.error}");
        return;
    }

    var updated = callback.result;
    Debug.Log($"UpdateEmail: updated={updated}");
});
```

To update user’s profile:

```
user.UpdateProfile("YOUR_USER_NAME", "URL_TO_USER_PICTURE", (callback)
=>
{
    if (callback.success == false)
    {
        Debug.LogError($"UpdateProfile: {callback.error}");
        return;
    }

    var updated = callback.result;
    Debug.Log($"UpdateProfile: updated={updated}");
});
```

To update user’s password:

```
user.UpdatePassword("YOUR_NEW_PASSWORD", (callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"UpdatePassword: {callback.error}");
        return;
    }

    var updated = callback.result;
    Debug.Log($"UpdatePassword: updated={updated}");
});
```

To delete user:

```
user.DeleteUser((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"DeleteUser: {callback.error}");
        return;
    }

    var deleted = callback.result;
    Debug.Log($"DeleteUser: deleted={deleted}");
});
```

Full list of API methods you can find here:

https://firebase.google.com/docs/reference/js/auth.md

Firebase Functions

Before you can invoke functions over network, you have to get `IFirebaseFunctionsHttpsCallable` instance. To get it:

```
var callable = app.Functions.HttpsCallable("METHOD_NAME", new FirebaseFunctionsHttpsCallableOptions{ limitedUseAppCheckTokens = false });
//or
var callable = app.Functions.HttpsCallableFromURL("https://PATH_TO_FIREBASE_FUNCTION", new FirebaseFunctionsHttpsCallableOptions{ limitedUseAppCheckTokens = false });
```

Then you got a callable instance, you are able to invoke it with or without input data:

```
callable.Request((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"Request ({nameof(callable)}): error={callback.error}");
        return;
    }

    Debug.Log($"Request ({nameof(callable)}): {callback.result}");
});
```

Full list of API methods you can find here: https://firebase.google.com/docs/reference/js/functions.md

Firebase Messaging

Before receiving messages you have to get **Firebase Messaging** token:

```
app.Messaging.GetToken((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetToken: {callback.error}");
        return;
    }
    var messagingToken = callback.result;
    //do something with 'messagingToken' here
});
```

To delete this token:

```
app.Messaging.DeleteToken((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"DeleteToken: {callback.error}");
        return;
    }

    if (callback.result == false)
    {
        Debug.LogError($"DeleteToken: token isn't deleted");
        return;
    }

    Debug.Log("DeleteToken: token deleted");
});
```

To subscribe to `OnMessage` callback:

```
app.Messaging.OnMessage((message) =>
{
    //do something with "message" here
}));
```

Full list of API methods you can find here:
https://firebase.google.com/docs/reference/js/messaging_.md#@firebase/messaging

Firebase Installations

To get installation id:

```
app.Installations.GetId((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetId: {callback.error}");
        return;
    }
    var installationId = callback.result;
    //do something with 'installationId' here
});
```

To delete installation:

```
app.Installations.DeleteInstallations((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"DeleteInstallations: {callback.error}");
        return;
    }

    if (!callback.result)
    {
        Debug.LogError($"DeleteInstallations: {callback.error}");
        return;
    }

    Debug.Log($"DeleteInstallations: installation deleted");
});
```

To get token:

```
var forceRefresh = false;
app.Installations.GetToken(forceRefresh, (callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetToken: {callback.error}");
        return;
    }
    var installationToken = callback.result;
    //do something with 'installationToken' here
});
```

To subscribe to `OnIdChange` callback:

```
app.Installations.OnIdChange((id) =>
{
    //do something with "id" here
});
```

Full list of API methods you can find here:
https://firebase.google.com/docs/reference/js/installations.md

Firebase Performance

Before you can record metrics, you have to get `IFirebasePerformanceTrace` instance. To get it:

```
var trace = app.Performance.Trace("TRACE_NAME");
```

To start recoding:

```
trace.Start();
```

To stop recording:

```
trace.Stop();
```

To set metric:

```
trace.PutMetric("YOUR_METRIC_NAME", 5);
```

To increment metric:

```
trace.IncrementMetric("YOUR_METRIC_NAME");
```

To get metric:

```
var value = trace.GetMetric("YOUR_METRIC_NAME");
//do something with "value" here
```

Full list of API methods you can find here:
https://firebase.google.com/docs/reference/js/performance.md

Firebase Remote Config

To activate:

```
 app.RemoteConfig.Activate((callback) =>
 {
     if (callback.success == false)
     {
         Debug.LogError($"Activate: {callback.error}");
         return;
     }
     Debug.Log($"Activate: {callback.result}");
```

```
});
```

To fetch config:

```
app.RemoteConfig.FetchConfig((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"FetchConfig: {callback.error}");
        return;
    }
    Debug.Log($"FetchConfig: {callback.result}");
});
```

To fetch config and activate it:

```
app.RemoteConfig.FetchAndActivate((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"FetchAndActivate: {callback.error}");
        return;
    }
    Debug.Log($"FetchAndActivate: {callback.result}");
});
```

To get all registered keys:

```
var keys = _app.RemoteConfig.GetKeys();
Debug.Log($"GetKeys: {string.Join(", ", keys)}");
```

To get `Boolean` value from a remote config:

```
var boolKey = "boolKey";
var boolValue = app.RemoteConfig.GetBoolean(boolKey);
```

To get `Integer` value from a remote config:

```
var integerKey = "integerKey";
var integerValue = app.RemoteConfig.GetInteger(integerKey);
```

To get `String` value from a remote config:

```
var stringKey = "stringKey";
var stringValue = app.RemoteConfig.GetString(stringKey);
```

To subscribe to `onConfigUpdate` callback:

```
app.RemoteConfig.OnConfigUpdate((updatedKeys) =>
{
    //do something with "updatedKeys" here
}));
```

Full list of API methods you can find here:

https://firebase.google.com/docs/reference/js/remote-config.md

Before you can get data from Firebase Storage, you have to get storage `IFirebaseStorageReference` instance. To get it:

```
var itemRef = app.Storage.Ref("/path/to/item");
```

To get ref’s download url:

```
itemRef.GetDownloadURL((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetDownloadUrl: {itemRef.fullPath},
{callback.error}");
        return;
    }
    var downloadUrl callback.result;
    //do something with "downloadUrl" here
});
```

To get ref’s metadata:

```
itemRef.GetMetadata((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetMetadata: {callback.error}");
        return;
    }
    var metadata = callback.result;
    //do something with "metadata" here
});
```

To download ref’s data:

```
itemRef.GetBytes((callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"GetBytes: {fileRef.fullPath},
{callback.error}");
        return;
    }

    var downloadedBytes = callback.result;
    //do something with "downloadedBytes" here
});
```

To upload ref’s data:

```
var bytes = new byte[128];
itemRef.UploadBytes(bytes, (callback) =>
{
    if (callback.success == false)
    {
        Debug.LogError($"UploadBytes: {fileRef.fullPath},
{callback.error}");
        return;
    }

    var metadata = callback.result;
    //do something with "metadata" here
```

To delete ref:

```
itemRef.DeleteObject();
```

Full list of API methods you can find here:
https://firebase.google.com/docs/reference/js/storage.md

Remarks

This package created and maintained by **Pavel Shestakov** (founder of **Ritehook Games**) and distributed under the MIT license, the source code of the package you can find at: https://github.com/am1goo/FirebaseWebGL-Unity

Additionally, if you are using `com.cysharp.unitask` package, you can also use our UniTask extensions: https://github.com/am1goo/FirebaseWebGL-Unity-UniTask

For any questions or inquiries, feel free to contact us at: assetstore@ritehook.games