Playgama Bridge

ℹ️ Find the complete SDK documentation with the latest updates, API references, and guides here.

⚠️ Steps marked with an asterisk (*) are required.

Intro

Playgama Bridge is a unified SDK for publishing HTML5 games on various platform.

SDK integration is mandatory, but during the game submission, you can choose the platforms you want to publish on.

Currently, our SDK is integrated with the following platforms:

- Yandex Games
- Facebook
- Crazy Games
- Game Distribution
- ClickJogos
- DigitalWIll
- Success-Corp
- Telegram Playdeck
- Y8
- Lagged
- Poki

Setup

* Installation

Download the latest `.unitypackage` from the GitHub release page and import it into your project. To ensure proper functionality, import all files except for the `Examples` folder, which can be imported optionally. The `Examples` folder contains scenes demonstrating usage.

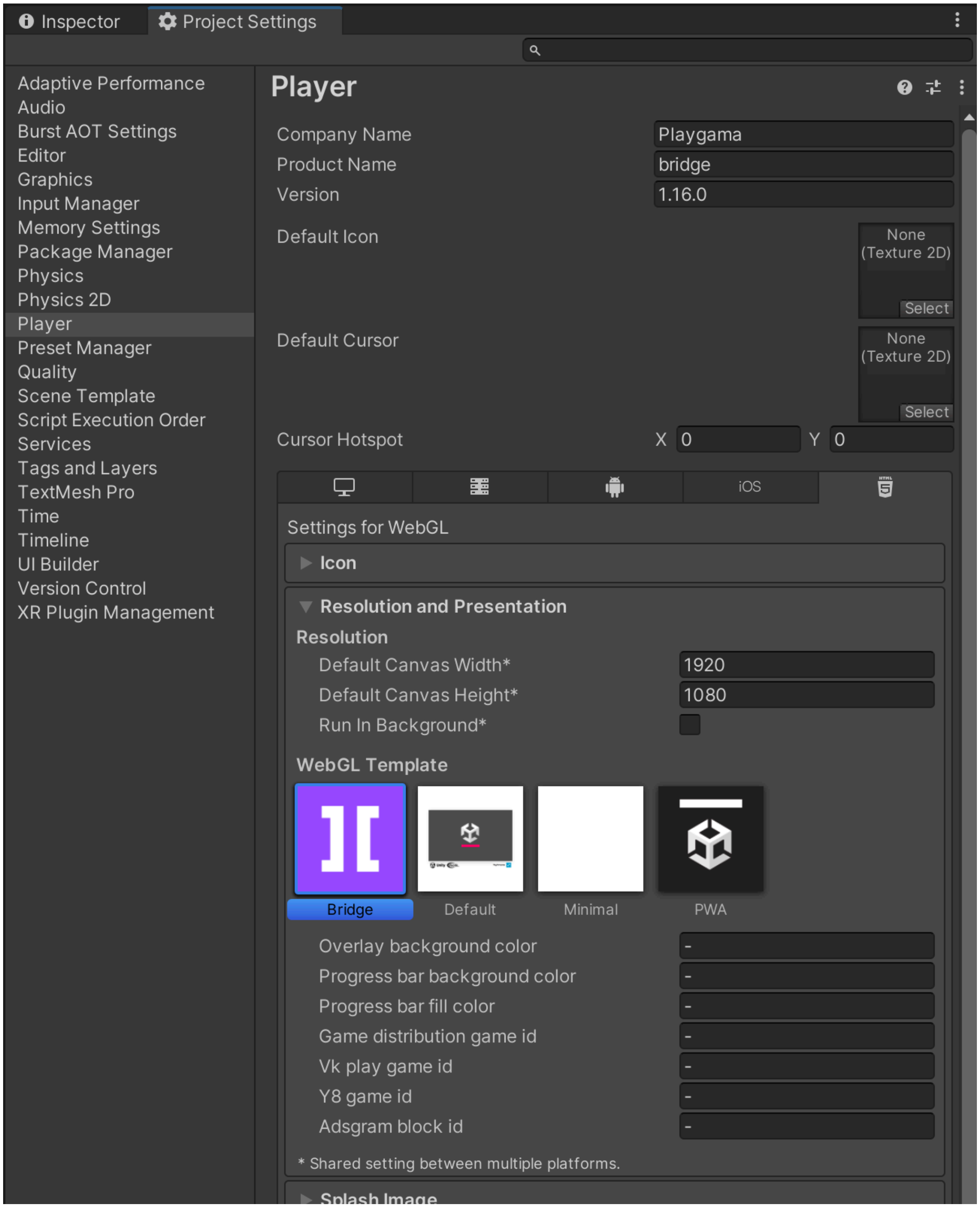
If you get an error "Namespace name Newtonsoft could not be found" – add Newtonsoft JSON package via Package Manager → Add Package By Name → `com.unity.nuget.newtonsoft-json`.

* Initialization and Build

When you build the project, please enable "Decompression Fallback" in the Unity build settings

Once the game is loaded, the plugin is already initialized. No additional actions are required.

To ensure the plugin functions correctly, select the appropriate WebGL Template during the build process.



Player

Company Name	Playgama
Product Name	bridge
Version	1.16.0
Default Icon	None (Texture 2D)
Default Cursor	None (Texture 2D)
Cursor Hotspot	X 0 Y 0

Settings for WebGL

Resolution and Presentation

Resolution

Default Canvas Width*	1920
Default Canvas Height*	1080
Run In Background*	☐

WebGL Template: [✓ Bridge] Default Minimal PWA

Overlay background color	-
Progress bar background color	-
Progress bar fill color	-
Game distribution game id	-
Vk play game id	-
Y8 game id	-
Adsgram block id	-

* Shared setting between multiple platforms.

There are a few important settings here

- Overlay Background Color: The background color of the overlay.

- Progress Bar Background Color: The background color of the loading bar.
- Progress Bar Fill Color: The color of the loading bar itself.

Colors should be specified in HEX format. For example, black is `#000000`. Use the `-` symbol for default values.

To replace the icon, simply replace the file `WebGLTemplates/Bridge/icon.png`.

Platform Parameters

At any time, you can retrieve values for specific parameters that you might use in your game, such as the user's browser language.

Platform ID

Identify the platform on which the game is currently running to customize features and settings accordingly.

```
Bridge.platform.id
```

Returns the platform ID on which the game is currently running. Possible values: `playgama`, `vk`, `ok`, `yandex`, `facebook`, `crazy_games`, `game_distribution`, `wortal`, `playdeck`, `telegram`, `y8`, `lagged`, `msn`, `poki`, `mock`.

* Language

ℹ️ Check the language to display proper text labels.

Get the language set by the user on the platform or the browser language if not provided by the platform, to localize game content.

```
Bridge.platform.language
```

Returns the language set by the user on the platform. If the platform does not provide this data, it returns the browser language. Format: ISO 639-1. Example: `ru`, `en`.

URL Parameter

Embed auxiliary information into the game URL to pass additional data or settings when launching the game.

```
Bridge.platform.payload
```

Allows embedding auxiliary information into the game URL.

Platform	URL Format
VK	http://vk.com/game_id**#your-info**
Yandex	http://yandex.com/games/app/game_id**?payload=your-info**
Crazy Games	crazygames.com/game/game_name**?payload=your-info**
Mock	site.com/game_name**?payload=your-info**

Domain Information

Retrieve the top-level domain of the platform to handle domain-specific configurations and behavior.

```
Bridge.platform.tld
```

Returns the top-level domain (TLD) of the platform. If there is no data – `null`. If the data is available – `com`, `ru`, etc.

* Sending a Message to the Platform

ℹ️ The call to Bridge.platform.SendMessage with the parameter PlatformMessage.GameReady is mandatory!

Don't forget to implement it.

Send predefined messages to the platform to trigger specific actions or events, such as signaling that the game is ready.

```
Bridge.platform.SendMessage(PlatformMessage.GameReady)
```

Message	Description
GameReady	The game has loaded, all loading screens have passed, and the player can interact with the game.
InGameLoadingStarted	Some loading inside the game has started. For example, when a level is loading.
InGameLoadingStopped	Loading inside the game is finished.
GameplayStarted	Gameplay has started. For example, the player has entered a level from the main menu.
GameplayStopped	Gameplay has ended/paused. For example, when exiting from a level to the main menu or opening the pause menu.
PlayerGotAchievement	The player has reached a significant milestone. For example, defeating a boss or setting a new record.

Server Time

```
private void Start()
{
    Bridge.platform.GetServerTime(OnGetServerTimeCompleted);
}

private void OnGetServerTimeCompleted(DateTime? result)
{
    if (result.HasValue)
    {
        Debug.Log(result.Value); // UTC time
    }
}
```

Current Visibility State

Check if the game tab is visible or hidden, and adjust game behavior accordingly, such as muting sound when hidden.

```
Bridge.game.visibilityState
```

Returns the current visibility state of the game (the tab with the game). Possible values: `visible`, `hidden`.

```
// To track visibility state changes, subscribe to the event
private void Start()
{
    Bridge.game.visibilityStateChanged += OnGameVisibilityStateChanged;
}

private void OnGameVisibilityStateChanged(VisibilityState state)
{
    switch (state)
    {
        case VisibilityState.Visible:
            // The game tab is visible
            break;
        case VisibilityState.Hidden:
            // The game tab is hidden
            break;
    }
}
```

ℹ️ React to changes in visibility state. For example, mute the game sound when the state is `hidden` and unmute when the state is `visible`.

User Data

Store and manage player data to enhance gameplay experience and retain progress.

There are two types of storage: local (`LocalStorage`) and internal (`PlatformInternal`). When writing to local storage, data is saved on the player's device. When writing to internal storage, data is saved on the platform's servers.

If you need to call storage methods in a sequence, make sure you wait for previous call to finish, so there is no potential data collisions.

Use List<string> parameters for batch operations

Default Storage Type

Identify the default storage type to understand where data is being saved (local or server).

```
Bridge.storage.defaultType
```

Used automatically if no specific storage type is specified when working with data. Possible values: `LocalStorage`, `PlatformInternal`.

Support Check

Verify if the specified storage type is supported on the platform to ensure compatibility.

```
Bridge.storage.IsSupported(StorageType.LocalStorage)
Bridge.storage.IsSupported(StorageType.PlatformInternal)
```

Availability Check

Check if the specified storage type is currently available for use to manage data storage effectively.

```
Bridge.storage.IsAvailable(StorageType.LocalStorage)
Bridge.storage.IsAvailable(StorageType.PlatformInternal)
```

* Load Data

Retrieve stored data based on a key or multiple keys to restore player progress or settings.

Don't call SDK methods that require a callback (like `Bridge.storage.Get`) when starting the game in `Awake`. Call them in `Start`.

```
// Get data by key
private void Start()
{
    Bridge.storage.Get("level", OnStorageGetCompleted);
}

private void OnStorageGetCompleted(bool success, string data)
{
    // Loading succeeded
    if (success)
    {
        if (data != null)
        {
            Debug.Log(data);
        }
        else
        {
            // No data for the key 'level'
        }
    }
    else
    {
        // Error, something went wrong
    }
}

// Get data by multiple keys
private void Start()
{
    Bridge.storage.Get(new List<string>() { "level", "coins" }, OnStorageGetCo
mpleted);
}

private void OnStorageGetCompleted(bool success, List<string> data)
{
    // Loading succeeded
    if (success)
    {
        if (data[0] != null)
        {
            Debug.Log($"Level: {data[0]}");
        }
        else
        {
            // No data for the key 'level'
        }

        if (data[1] != null)
        {
            Debug.Log($"Coins: {data[1]}");
        }
        else
        {
            // No data for the key 'coins'
        }
    }
    else
    {
        // Error, something went wrong
    }
}

// Get data from a specific storage type
private void Start()
{
    Bridge.storage.Get("level", OnStorageGetCompleted, StorageType.LocalSto
rage);
}

private void OnStorageGetCompleted(bool success, string data)
{
    // Loading succeeded
    if (success)
    {
        if (data != null)
        {
            Debug.Log(data);
        }
        else
        {
            // No data for the key 'level'
        }
    }
    else
    {
        // Error, something went wrong
    }
}
```

* Save Data

Save data to the specified storage with a key to retain player progress or settings.

```
// Save data by key
private void Start()
{
    Bridge.storage.Set("level", "dungeon_123", OnStorageSetCompleted);
}

private void OnStorageSetCompleted(bool success)
{
    Debug.Log($"OnStorageSetCompleted, success: {success}");
}

// Save data by multiple keys
private void Start()
{
    var keys = new List<string>() { "level", "is_tutorial_completed", "coins" };
    var data = new List<object>() { "dungeon_123", true, 12 };
    Bridge.storage.Set(keys, data, OnStorageSetCompleted);
}

// Save data to a specific storage type
private void Start()
{
    Bridge.storage.Set("level", "dungeon_123", OnStorageSetCompleted, Storag
eType.LocalStorage);
}
```

Delete Data

Remove data from the specified storage by key to manage player data and settings effectively.

```
// Delete data by key
private void Start()
{
    Bridge.storage.Delete("level", OnStorageDeleteCompleted);
}

private void OnStorageDeleteCompleted(bool success)
{
    Debug.Log($"OnStorageDeleteCompleted, success: {success}");
}

// Delete data by multiple keys
private void Start()
{
    var keys = new List<string>() { "level", "is_tutorial_completed", "coins" };
    Bridge.storage.Delete(keys, OnStorageDeleteCompleted);
}

// Delete data from a specific storage type
private void Start()
{
    Bridge.storage.Delete("level", OnStorageDeleteCompleted, StorageType.LocalStorage);
}
```

If no specific storage type is passed as the third argument when working with data, the default storage type `Bridge.storage.defaultType` is used.

Advertising

Monetize your game by integrating various types of advertisements, including banners, interstitials, and rewarded ads.

Advertisements should not be displayed during gameplay to avoid disrupting the user experience. Disruptive ads can drive users away. Instead, ads should be shown at natural breakpoints, such as during level transitions, map changes, or when the player dies

Banner

Is Banner Supported

Check if the platform supports displaying banner ads. Use this to determine if you can include banner advertisements in your game.

```
// Possible values: true, false
Bridge.advertisement.isBannerSupported
```

Ensure that in-game banners are not displayed during gameplay on CrazyGames. Please refer to Advertisement - CrazyGames Documentation.

Show Banner

Display a banner ad within your game to generate revenue through advertising.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "vk":
            options.Add("position", "bottom");
            options.Add("layoutType", "resize");
            options.Add("canClose", false);
            break;
        case "crazy_games":
            options.Add("position", "top"); // optional, default = "bottom"
            break;
        case "game_distribution":
            options.Add("position", "top"); // optional, default = "bottom"
            break;
    }

    Bridge.advertisement.ShowBanner(options);
}
```

Hide Banner

Hide the currently displayed banner ad when it is no longer needed.

```
Bridge.advertisement.HideBanner()
```

Banner State

Monitor the state of the banner ad (loading, shown, hidden, failed) to manage its display and troubleshoot issues.

```
Bridge.advertisement.bannerState
```

Possible values: `Loading`, `Shown`, `Hidden`, `Failed`.

```
// To track banner state changes, subscribe to the event
private void Start()
{
    Bridge.advertisement.bannerStateChanged += OnBannerStateChanged;
}

private void OnBannerStateChanged(BannerState state)
{
    Debug.Log(state);
}
```

Interstitial

Interstitial ads typically appear during transitions in the game, such as level loading or after game over.

Minimum Interval Between Displays

Set the minimum time interval between interstitial ad displays to comply with platform requirements and improve user experience.

```
// Default value = 60 seconds
Bridge.advertisement.minimumDelayBetweenInterstitial

private void Start()
{
    // Set minimum interval
    Bridge.advertisement.SetMinimumDelayBetweenInterstitial(30);
}
```

There should be time intervals between interstitial ad displays. For convenience, this SDK includes a built-in timer mechanism between ad displays. You just need to specify the required interval, and you can call the ad display method as often as you like.

* Interstitial State

ℹ️ Check the `interstitialState` at the start of the game, and if the ad is `Opened`, perform the necessary actions (mute sounds/pause the game/etc.).

Track the state of the interstitial ad (loading, opened, closed, failed) to manage ad display and user experience.

```
Bridge.advertisement.interstitialState
```

Possible values: `Loading`, `Opened`, `Closed`, `Failed`.

```
// To track interstitial state changes, subscribe to the event
private void Start()
{
    Bridge.advertisement.interstitialStateChanged += OnInterstitialStateChange
d;
}

private void OnInterstitialStateChanged(InterstitialState state)
{
    Debug.Log(state);
}
```

React to changes in ad state. For example, mute the game sound when `Opened` and unmute when `Closed` and `Failed`.

* Show Interstitial

Display an interstitial ad at appropriate moments, such as during level transitions or game over screens.

```
private void Start()
{
    Bridge.advertisement.ShowInterstitial();
}
```

Do not call `ShowInterstitial()` method at the start of the game. On platforms where this is allowed, the ad will be shown automatically.

Rewarded

Rewarded ads are a type of advertisement where players have the option to watch an ad in exchange for in-game rewards

Offer players rewards in exchange for watching ads, incentivizing ad engagement and increasing ad revenue.

Rewarded State

Monitor the state of the rewarded ad (loading, opened, closed, rewarded, failed) to manage the reward process.

```
Bridge.advertisement.rewardedState
```

Possible values: `Loading` , `Opened` , `Closed` , `Rewarded` , `Failed` .

```
// To track rewarded ad state changes, subscribe to the event
private void Start()
{
    Bridge.advertisement.rewardedStateChanged += OnRewardedStateChange
d;
}

private void OnRewardedStateChanged(RewardedState state)
{
    Debug.Log(state);
}
```

React to changes in ad state. For example, mute the game sound when `Opened` and unmute when `Closed` and `Failed`.

Reward the player only when the state is `Rewarded`.

Show Rewarded Ad

Display a rewarded ad and provide incentives to players for watching the entire ad.

```
Bridge.advertisement.ShowRewarded()
```

AdBlock

Check if the ad blocker is enabled.

```
private void Start()
{
    Bridge.advertisement.CheckAdBlock(OnCheckAdBlockCompleted);
}

private void OnCheckAdBlockCompleted(bool result)
{
    Debug.Log(result); // true or false
}
```

User Parameters

You can also retrieve various information about the player and their device.

Device Type

Determine the type of device (mobile, tablet, desktop, tv) the game is being played on to adjust the game’s interface and performance settings.

```
Bridge.device.type
```

Returns the type of device the user launched the game from. Possible values: `Mobile`, `Tablet`, `Desktop`, `TV`.

Authorization Support

Check if the platform supports player authorization to enable features that require user authentication, such as saving game progress or accessing social features.

```
// Possible values: true, false
Bridge.player.isAuthorizationSupported
```

Is the Player Currently Authorized

Verify if the player is currently authorized on the platform. This allows you to enable personalized features, such as saving high scores or providing user-specific content.

```
// Possible values: true, false
Bridge.player.isAuthorized
```

Player ID

Get the player’s unique ID on the platform to manage user-specific data and settings. Use this ID to track player progress, achievements, and purchases

```
Bridge.player.id
```

If the platform supports authorization and the player is currently authorized, this returns their platform ID. Otherwise, it returns `null`.

Player Name

Retrieve the player's name to personalize the game experience. Display the name in leaderboards, friend lists, or when sending notifications and messages.

```
Bridge.player.name
```

If there is no data – `null`. If the data is available, the data is in `string` format.

Player Photos

Get the count of player avatars available. Use this to manage and display user profile images effectively, such as showing the avatar in multiplayer lobbies or profile screens.

```
Bridge.player.photos
```

Possible values: an array of player avatars (sorted by increasing resolution), empty array.

Player Authorization

Authorize the player on the platform to access protected features and personalize the game experience. For example, prompting the player to log in to save their progress or unlock social sharing features.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "yandex":
            options.Add("scopes", true);
            break;
    }

    Bridge.player.Authorize(options, OnAuthorizePlayerCompleted);
}

private void OnAuthorizePlayerCompleted(bool success)
{
    if (success)
    {
        // Player successfully authorized
    }
    else
    {
        // Error, something went wrong
    }
}
```

Social Interactions

Enable social features to enhance player engagement by allowing them to share, join communities, invite friends, and more.

Share

Use this to allow players to share game content or achievements on social media platforms.

```
Bridge.social.isShareSupported
```

Check if the share functionality is supported on the platform.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "vk":
            options.Add("link", "YOUR_LINK");
            break;
    }

    Bridge.social.Share(options, OnShareCompleted);
}

private void OnShareCompleted(bool success)
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

Join Community

Enable players to join social communities related to your game, enhancing engagement and loyalty.

```
Bridge.social.isJoinCommunitySupported
```

Check if the join community functionality is supported on the platform.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "vk":
            options.Add("groupId", YOUR_GROUP_ID);
            break;
        case "ok":
            options.Add("groupId", YOUR_GROUP_ID);
            break;
    }

    Bridge.social.JoinCommunity(options, OnJoinCommunityCompleted);
}
```

```
private void OnJoinCommunityCompleted(bool success)
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

Invite Friends

Allow players to invite their friends to play the game, helping to grow your player base organically.

```
Bridge.social.isInviteFriendsSupported
```

Check if the invite friends functionality is supported on the platform.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "ok":
            options.Add("text", "Hello World!");
            break;
    }

    Bridge.social.InviteFriends(options, OnInviteFriendsCompleted);
}

private void OnInviteFriendsCompleted(bool success)
```

```
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

Create Post

Use this to let players create posts about their achievements or updates directly from the game.

```
Bridge.social.isCreatePostSupported
```

Check if the create post functionality is supported on the platform.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "ok":
            var media = new object[]
            {
                new Dictionary<string, object>
                {
                    { "type", "text" },
                    { "text", "Hello World!" },
                },
                new Dictionary<string, object>
                {
```

```
                { "type", "link" },
                { "url", "https://apiok.ru" },
            },
            new Dictionary<string, object>
            {
                { "type", "poll" },
                { "question", "Do you like our API?" },
                {
                    "answers",
                    new object[]
                    {
                        new Dictionary<string, object>
                        {
                            { "text", "Yes" },
                        },
                        new Dictionary<string, object>
                        {
                            { "text", "No" },
                        }
                    }
                },
                { "options", "SingleChoice,AnonymousVoting" },
            },
        };

        options.Add("media", media);
        break;
    }

    Bridge.social.CreatePost(options, OnCreatePostCompleted);
}

private void OnCreatePostCompleted(bool success)
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

Add to Favorites

Allow players to bookmark your game for easy access in the future.

```
Bridge.social.isAddToFavoritesSupported
```

Check if the add to favorites functionality is supported on the platform.

```
private void Start()
{
    Bridge.social.AddToFavorites(OnAddToFavoritesCompleted);
}

private void OnAddToFavoritesCompleted(bool success)
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

Add to Home Screen

Enable players to add a shortcut to your game on their home screen for quick access.

```
Bridge.social.isAddToHomeScreenSupported
```

Check if the add to home screen functionality is supported on the platform.

```
private void Start()
{
    Bridge.social.AddToHomeScreen(OnAddToFavoritesCompleted);
}

private void OnAddToHomeScreenCompleted(bool success)
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

Rate Game

Encourage players to rate your game, providing valuable feedback and improving visibility.

```
Bridge.social.isRateSupported
```

Check if the rate game functionality is supported on the platform.

```
private void Start()
{
    Bridge.social.Rate(OnRateCompleted);
}

private void OnRateCompleted(bool success)
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

External Links

Allow players to follow links to external websites, such as your game's official site or related resources.

```
Bridge.social.isExternalLinksAllowed
```

Check if external links are allowed on the platform.

Leaderboards

Enhance competitiveness by integrating leaderboards, allowing players to compare their scores and achievements.

Support

Use this to determine if you can implement leaderboards for your game on the current platform.

Check if the leaderboard feature is supported on the platform.

```
Bridge.leaderboard.isSupported
```

Check if built-in popup is supported.

```
Bridge.leaderboard.isNativePopupSupported
```

Check if multiple boards are supported.

```
Bridge.leaderboard.isMultipleBoardsSupported
```

Check if user can set score to leaderboard.

```
Bridge.leaderboard.isSetScoreSupported
```

Check if user can retrieve their score.

```
Bridge.leaderboard.isGetScoreSupported
```

Check if user can access leaderboard entries including other players' scores.

```
Bridge.leaderboard.isGetEntriesSupported
```

Player Scores. Set

Submit the player's score to the leaderboard to update their rank and position.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "yandex":
            options.Add("score", 42);
            options.Add("leaderboardName", "YOUR_LEADERBOARD_NAME");
            break;
        case "facebook":
            options.Add("score", 42);
            options.Add("leaderboardName", "YOUR_LEADERBOARD_NAME");
            break;
        case "lagged":
            options.Add("score", 42);
            options.Add("boardId", "YOUR_LEADERBOARD_ID");
            break;
        case "y8":
            options.Add("points", 42);
            options.Add("table", "YOUR_LEADERBOARD_NAME");
            break;
    }

    Bridge.leaderboard.SetScore(options, OnSetScoreCompleted);
}

private void OnSetScoreCompleted(bool success)
{
    if (success)
    {
        // Operation succeeded
    }
    else
    {
        // An error occurred
    }
}
```

Player Scores. Get

Retrieve the player's score from the leaderboard to display their current standing.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "yandex":
            options.Add("leaderboardName", "YOUR_LEADERBOARD_NAME");
            break;
        case "facebook":
            options.Add("leaderboardName", "YOUR_LEADERBOARD_NAME");
            break;
        case "y8":
            options.Add("table", "YOUR_LEADERBOARD_NAME");
            break;
    }

    Bridge.leaderboard.GetScore(options, OnGetScoreCompleted);
}

private void OnGetScoreCompleted(bool success, int score)
{
    if (success)
    {
        // Data successfully retrieved
        Debug.Log(score);
    }
    else
    {
        // Something went wrong
    }
}
```

Get Full Leaderboard

Retrieve entries from the leaderboard, including the player's rank and score, to display a comprehensive leaderboard.

Don't call SDK methods that require a callback (like `Bridge.leaderboard.GetEntries`) when starting the game in `Awake`. Call them in `Start`.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "yandex":
            options.Add("leaderboardName", "YOUR_LEADERBOARD_NAME");
            options.Add("includeUser", true);
            options.Add("quantityAround", 10);
            options.Add("quantityTop", 10);
            break;
        case "facebook":
            options.Add("leaderboardName", "YOUR_LEADERBOARD_NAME");
            options.Add("count", 10);
            options.Add("offset", 10);
            break;
        case "y8":
            options.Add("table", "YOUR_LEADERBOARD_NAME");
            options.Add("page", 1);
            options.Add("perPage", 10);
            break;
    }

    Bridge.leaderboard.GetEntries(options, OnGetEntriesCompleted);
}

private void OnGetEntriesCompleted(bool success, List<Dictionary<string, stri
ng>> entries)
{
    Debug.Log($"OnGetEntriesCompleted, success: {success}, entries:");

    if (success)
    {
        switch (Bridge.platform.id)
        {
            case "yandex":
                foreach (var entry in entries)
                {
                    Debug.Log("ID: " + entry["id"]);
                    Debug.Log("Score: " + entry["score"]);
                    Debug.Log("Rank: " + entry["rank"]);
                    Debug.Log("Name: " + entry["name"]);
                    Debug.Log("Photo: " + entry["photo"]);
                }
                break;
            case "facebook":
                foreach (var entry in entries)
                {
                    Debug.Log("ID: " + entry["playerId"]);
                    Debug.Log("Score: " + entry["score"]);
                    Debug.Log("Rank: " + entry["rank"]);
                    Debug.Log("Name: " + entry["playerName"]);
                    Debug.Log("Photo: " + entry["playerPhoto"]);
                }
                break;
            case "y8":
                foreach (var entry in entries)
                {
                    Debug.Log("ID: " + entry["playerid"]);
                    Debug.Log("Score: " + entry["points"]);
                    Debug.Log("Rank: " + entry["rank"]);
                    Debug.Log("Name: " + entry["playername"]);
                }
                break;
        }
    }
}
```

Show Native Popup

Shows leaderboard entries built-in popup

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "y8":
            options.Add("table", "YOUR_LEADERBOARD_NAME");
            break;
    }

    Bridge.leaderboard.ShowNativePopup(options, OnShowNativePopupCompl
eted);
}

private void OnShowNativePopupCompleted(bool success)
{
    if (success)
    {
        // Data successfully retrieved
        Debug.Log(score);
    }
    else
    {
        // Something went wrong
    }
}
```

Achievements

Achievements in HTML5 games are an exciting and rewarding feature that adds an extra layer of engagement for players. They serve as milestones, celebrating a player's progress, skill, and dedication.

}

Support

Use this to determine if you can implement achievements for your game on the current platform.

```
Bridge.achievements.isSupported
```

Check if getting list of achievements is supported.

```
Bridge.achievements.isGetListSupported
```

Check if built-in popup is supported.

```
Bridge.payments.isNativePopupSupported
```

Unlock achievement

Unlocks achievement for a player.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "y8":
            options.Add("achievement", "ACHIEVEMENT_NAME");
            options.Add("achievementkey", "ACHIEVEMENT_KEY");
            break;
        case "lagged":
            options.Add("achievement", "ACHIEVEMENT_ID");
            break;
    }

    Bridge.achievements.Unlock(options, OnAchievementsUnlockCompleted);
}
```

```
private void OnAchievementsUnlockCompleted(bool success, Dictionary<strin
g, string> result)
{
    Debug.Log(success);
}
```

Get List

Returns the achievement list in JSON

```
private void Start()
{
    Bridge.achievements.GetList(OnGetListCompleted);
}

private void OnGetListCompleted(bool success, List<Dictionary<string, string
>> list)
{
    Debug.Log($"OnGetListCompleted, success: {success}, items:");

    if (success)
    {
        switch (Bridge.platform.id)
        {
            case "y8":
                foreach (var item in list)
                {
                    Debug.Log("achievementid:" + item["achievementid"]);
                    Debug.Log("achievement:" + item["achievement"]);
                    Debug.Log("achievementkey:" + item["achievementkey"]);
                    Debug.Log("description:" + item["description"]);
                    Debug.Log("icon:" + item["icon"]);
                    Debug.Log("difficulty:" + item["difficulty"]);
                    Debug.Log("secret:" + item["secret"]);
                    Debug.Log("awarded:" + item["awarded"]);
                    Debug.Log("game:" + item["game"]);
                    Debug.Log("link:" + item["link"]);
                    Debug.Log("playerid:" + item["playerid"]);
                    Debug.Log("playername:" + item["playername"]);
                    Debug.Log("lastupdated:" + item["lastupdated"]);
                    Debug.Log("date:" + item["date"]);
                    Debug.Log("rdate:" + item["rdate"]);
                }
                break;
        }
    }
}
```

Show Native Popup

Some platforms support built-in achievement list which is shown in overlay

```
private void Start()
{
    Bridge.achievements.ShowNativePopup(options, (success) ⇒
    {
        Debug.Log($"OnShowNativePopupCompleted, success: {success}");
    });
}
```

In-Game Purchases

Enable players to purchase items, upgrades, or currency within your game to enhance their experience and generate revenue.

There are two types of purchases — permanent (e.g., ad removal) and consumable (e.g., in-game coins).

Support

Check if in-game purchases are supported to offer items or upgrades within the game.

```
Bridge.payments.isSupported
```

Check if getting catalog is supported.

```
Bridge.payments.isGetCatalogSupported
```

Check if getting purchases list is supported.

```
Bridge.payments.isGetPurchasesSupported
```

Check if purchase consuming is supported.

```
Bridge.payments.isConsumePurchaseSupported
```

Purchase

Allow players to buy items or upgrades in your game to enhance their gameplay experience.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "yandex":
            options.Add("id", "PRODUCT_ID");
            break;
        case "facebook":
            options.Add("productID", "PRODUCT_ID");
            break;
        case "playdeck":
            options.Add("amount", 1); // integer
            options.Add("description", "DESCRIPTION");
            options.Add("externalId", "EXTERNAL_ORDER_ID"); // Unique order ide
ntifier in your system, which you can use to check in postback that payment w
as successful
            break;
    }

    Bridge.payments.Purchase(options, OnPurchaseCompleted);
}

private void OnPurchaseCompleted(bool success, Dictionary<string, string> p
urchase)
{
    Debug.Log($"OnPurchaseCompleted, success: {success}");

    if (success)
    {
        switch (Bridge.platform.id)
        {
            case "yandex":
                Debug.Log("Product ID: " + purchase["productID"]);
                Debug.Log("Purchase Token: " + purchase["purchaseToken"]);
                break;
            case "facebook":
                Debug.Log("Product ID: " + purchase["productID"]);
                Debug.Log("Purchase Token: " + purchase["purchaseToken"]);
                break;
            case "playdeck":
                Debug.Log("Purchase Status: " + purchase["status"]);
                break;
        }
    }

}
```

Consume Purchase

Consume purchased items, such as in-game currency, once they are used, to manage inventory and player progression.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "yandex":
            options.Add("purchaseToken", "PURCHASE_TOKEN");
            break;
        case "facebook":
            options.Add("purchaseToken", "PURCHASE_TOKEN");
            break;
    }

    Bridge.payments.ConsumePurchase(options, OnConsumePurchaseComplet
ed);
}

private void OnConsumePurchaseCompleted(bool success)
{
    Debug.Log(success);
}
```

Catalog of All Items

Retrieve a list of all available in-game items that players can purchase to display in the game store.

```
private void Start()
{
    Bridge.payments.GetCatalog(OnGetCatalogCompleted);
}
```

```
private void OnGetCatalogCompleted(bool success, List<Dictionary<string, str
ing>> catalog)
{
    Debug.Log($"OnGetCatalogCompleted, success: {success}, items:");

    if (success)
    {
        switch (Bridge.platform.id)
        {
            case "yandex":
                foreach (var item in catalog)
                {
                    Debug.Log("ID: " + item["id"]);
                    Debug.Log("Title: " + item["title"]);
                    Debug.Log("Description: " + item["description"]);
                    Debug.Log("Image URI: " + item["imageURI"]);
                    Debug.Log("Price: " + item["price"]);
                    Debug.Log("Price Currency Code: " + item["priceCurrencyCod
e"]);
                    Debug.Log("Price Currency Image: " + item["priceCurrencyImag
e"]);
                    Debug.Log("Price Value: " + item["priceValue"]);
                }
                break;
            case "facebook":
                foreach (var item in catalog)
                {
                    Debug.Log("ID: " + item["productID"]);
                    Debug.Log("Title: " + item["title"]);
                    Debug.Log("Description: " + item["description"]);
                    Debug.Log("Image URI: " + item["imageURI"]);
                    Debug.Log("Price: " + item["price"]);
                    Debug.Log("Price Currency Code: " + item["priceCurrencyCod
e"]);
                    Debug.Log("Price Currency Image: " + item["priceCurrencyImag
e"]);
```

```
                Debug.Log("Price Amount: " + item["priceAmount"]);
            }
            break;
        }
    }
}
```

List of Purchased Items

Retrieve a list of items that the player has purchased to manage their inventory and provide access to purchased content.

Don't call SDK methods that require a callback (like `Bridge.payments.GetPurchases`) when starting the game in `Awake`. Call them in `Start`.

```
private void Start()
{
    Bridge.payments.GetPurchases(OnGetPurchasesCompleted);
}

private void OnGetPurchasesCompleted(bool success, List<Dictionary<string,
string>> purchases)
{
    Debug.Log($"OnGetPurchasesCompleted, success: {success}, items:");

    if (success)
    {
        switch (Bridge.platform.id)
        {
            case "yandex":
                foreach (var purchase in purchases)
                {
                    Debug.Log("Product ID: " + purchase["productID"]);
                    Debug.Log("Purchase Token: " + purchase["purchaseToken"]);
                }
                break;
            case "facebook":
                foreach (var purchase in purchases)
                {
                    Debug.Log("Product ID: " + purchase["productID"]);
                    Debug.Log("Purchase Token: " + purchase["purchaseToken"]);
                }
                break;
        }
    }
}
```

Remote Configuration

Manage your game settings remotely without releasing updates, allowing for dynamic adjustments and feature toggling.

Support

Check if remote configuration is supported to manage game settings without releasing updates.

```
Bridge.remoteConfig.isSupported
```

Load Values

Load configuration settings from the server to dynamically adjust game parameters based on real-time data.

Don't call SDK methods that require a callback (like `Bridge.remoteConfig.Get`) when starting the game in `Awake`. Call them in `Start`.

```
private void Start()
{
    var options = new Dictionary<string, object>();

    switch (Bridge.platform.id)
    {
        case "yandex":
            var clientFeatures = new object[]
            {
                new Dictionary<string, object>
                {
                    { "name", "levels" },
                    { "value", "5" },
                }
            };

            options.Add("clientFeatures", clientFeatures);
            break;
    }

    Bridge.remoteConfig.Get(options, OnRemoteConfigGetCompleted);
}

private void OnRemoteConfigGetCompleted(bool success, Dictionary<string,
string> data)
{
    if (success)
    {
        foreach (var keyValuePair in data)
        {
            Debug.Log($"key: { keyValuePair.Key }, value: { keyValuePair.Value }");
        }
    }
}
```