Unity

Table of Contents:

Unity is a versatile and powerful game engine widely used for creating both 2D and 3D games. It offers a comprehensive set of tools, including a robust asset store, visual editor, and extensive documentation, which makes it suitable for both beginners and experienced developers. Unity's ability to deploy to multiple platforms, including web browsers, enhances its appeal. Its graphical capabilities and performance optimization features are highly regarded, allowing for the creation of visually stunning and smooth-running games.

Intro

We strive to make our SDK as simple as possible to integrate and use while providing all the tools to make your game engaging and immersive. The integration guide is a comprehensive manual covering all options and player interactions on all supported platforms we work with.

While our full guide covers all features, most games use only the essentials. Below you'll find the required steps and an API reference for quick setup.

In addition to the documentation, you can rely on our dedicated GPT assistant to guide your integration and answer any questions about the Playgama Bridge SDK.

Required Steps:

1. Install and initialize SDK.
2. Identify the current language and display the required text labels.

3. Save and retrieve user progress using our storage methods.
4. Send Game Ready message.
5. Display interstitial ads.

API Reference:

1. Setup: Set up the SDK and connect your game.
2. Platform Parameters: Use platform parameters to manage your game, send messages, check language, and more.
3. User Data: Handle player data, progress, and preferences.
4. Advertising: Integrate and control ad placements using this API. It supports multiple ad formats: interstitials, rewarded ads, and banners.
5. User Parameters: If you need to authorize the user, check device type, or retrieve user info, check out the User Parameters section.
6. Social Interactions: If you support social interactions (inviting friends, sharing on social networks, adding the game to favorites, etc.), implement calls from the Social Interactions section.
7. Leaderboards: If your game has player rankings, implement calls from the Leaderboards section (and remember to save user data as well).
8. Achievements: If you plan to add achievements or milestones, consider implementing calls from the Achievements section.
9. In-Game Purchases: If your game has in-app purchases, implement calls from the In-Game Purchases section.
10. Remote Configuration: If you want to manage game settings remotely without releasing updates, check out Remote Configuration.

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.

Config

In the `WebGLTemplates/Bridge/playgama-bridge-config.json` config file, you can set up various identifiers and in-game purchases.

You can see an explanation of the file structure here: **Playgama Bridge Config** (documentation for this JSON config file).

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.

Unity WebGL Build Settings showing the "Decompression Fallback" option.

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 include: `playgama`, `vk`, `ok`, `yandex`, `facebook`, `crazy_games`, `game_distribution`, `playdeck`, `telegram`, `y8`, `lagged`, `msn`, `poki`, `qa_tool`, `discord`, `gamepush`, `mock`.

Language

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 (e.g. `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. For example:

Platform	URL Format
VK	`http://vk.com/game_id#your-info`

Platform	URL Format
Yandex	`http://yandex.com/games/app/game_id?payload=your-info`
CrazyGames	`https://crazygames.com/game/game_name?payload=your-info`
Mock	`https://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, returns `null`. If the data is available, it might return values like `com`, `ru`, etc.

Is GetAllGames Supported

Verify whether the platform supports the `GetAllGames` method to retrieve the correct links to the developer's other games.

```
Bridge.platform.isGetAllGamesSupported
```

Is GetGameById Supported

Verify whether the platform supports the `GetGameById` method to retrieve the correct link to a specific game.

```
Bridge.platform.isGetGameByIdSupported
```

Get All Games

This method retrieves the correct links to the developer's other games.

```
private void Start()
{
    Bridge.platform.GetAllGames((success, games) => {
        Debug.Log($"OnGetAllGamesCompleted, success: {success}, games:");

        if (success) {
            switch (Bridge.platform.id)
            {
```

```
                case "yandex":
                    foreach (var game in games) {
                        Debug.Log($"App ID: {game["appID"]}");
                        Debug.Log($"Title: {game["title"]}");
                        Debug.Log($"URL: {game["url"]}");
                        Debug.Log($"Cover URL: {game["coverURL"]}");
                        Debug.Log($"Icon URL: {game["iconURL"]}");
                    }
                    break;
            }
        }
    });
}
```

Get Game By ID

This method retrieves the correct link to a specific game from the developer.

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

    switch (Bridge.platform.id)
    {
        case "yandex":
            options.Add("gameId", "111111");
            break;
    }

    Bridge.platform.GetGameById(options, (success, game) => {
        Debug.Log($"OnGetGameByIdCompleted, success: {success}, game:");

        if (success) {
            switch (Bridge.platform.id)
            {
                case "yandex":
                    Debug.Log($"App ID: {game["appID"]}");
                    Debug.Log($"Title: {game["title"]}");
                    Debug.Log($"URL: {game["url"]}");
                    Debug.Log($"Cover URL: {game["coverURL"]}");
                    Debug.Log($"Icon URL: {game["iconURL"]}");
                    Debug.Log($"Is Available: {game["isAvailable"]}");
                    break;
            }
        }
```

```
    });
}
```

Sending a Message to the Platform

The call to `Bridge.platform.SendMessage(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 or 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 (for example, muting sound when hidden).

```
Bridge.game.visibilityState
```

Returns the current visibility state of the game's window/tab. 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;
    }
}
```

React to changes in visibility state. For example, you might 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 the 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 sequence, make sure you wait for the previous call to finish so there are 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.

```
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.

Do not call SDK methods that require a callback (like `Bridge.storage.Get`) during the Unity `Awake` phase. Call them in `Start` instead.

```
// 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" },
OnStorageGetCompleted);
}

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.LocalStorage);
}

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,
StorageType.LocalStorage);
}
```

Delete Data

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

```
// 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 players away. Instead, ads should be shown at natural breakpoints, such as during level transitions, between stages, or when the player's character dies.

Banner

There are some advertisement settings related to banners in the configuration file (`playgama-bridge-config.json`). Refer to the **Playgama Bridge Config** documentation for details on configuration options.

Is Banner Supported

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

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

Ensure that in-game banners are not displayed during gameplay on CrazyGames. Please refer to the [CrazyGames Ads Documentation] for platform-specific guidelines.

Show Banner

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

```
private void Start()
{
    var position =
BannerPosition.Bottom; // optional, 'Top' | 'Bottom' (default = Bottom)
    var placement = "test_placement";     // optional
    Bridge.advertisement.ShowBanner(position, placement);
}
```

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 (for example, during level loading or after a game over).

Minimum Interval Between Displays

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

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

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

There should be delays between interstitial ad displays. For convenience, the SDK includes a built-in timer mechanism between ad displays. You only need to specify the required interval, and then you can call the ad display method as often as you like (the SDK will handle the timing).

Interstitial State

Check the `interstitialState` at the start of the game. If an ad is already `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 += OnInterstitialStateChanged;
}

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

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

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

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

Do not call `ShowInterstitial()` *right at the start of the game. On platforms where an initial interstitial is allowed, the ad will be shown automatically by the platform.*

Rewarded

Rewarded ads are advertisements that players can choose to watch in exchange for in-game rewards.

Offering players optional rewards for watching ads can incentivize engagement with ads and increase ad revenue.

Rewarded State

Monitor the state of the rewarded ad (Loading, Opened, Closed, Rewarded, Failed) to manage the reward process appropriately.

```
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 += OnRewardedStateChanged;
}

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

React to changes in ad state. For example, mute the game sound when the ad is `Opened` and unmute when it is `Closed` or `Failed`. Only reward the player when the state is `Rewarded` (indicating the ad was watched to completion).

Show Rewarded Ad

Display a rewarded ad and provide the promised incentive to the player if they watch the entire ad.

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

AdBlock

Check if an ad blocker is enabled on the player's device or browser.

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

private void OnCheckAdBlockCompleted(bool result)
{
```

```
    Debug.Log(result); // true if an ad blocker is detected; false otherwise
}
```

User Parameters

You can 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 accordingly.

```
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. This is useful to enable features that require user authentication (such as saving game progress to the cloud or accessing social features).

```
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.

```
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, purchases, etc.

```
Bridge.player.id
```

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

Player Name

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

```
Bridge.player.name
```

If there is no data (name not available), this returns `null`. If the data is available, it returns the name as a `string`.

Player Avatar

Get the player's avatar(s). Some platforms provide an array of URLs or images for the player's avatars.

```
Bridge.player.photos
```

Possible values: an array of player avatar URLs (sorted by increasing resolution), or an empty array if no avatar is available.

Player Authorization

Initiate the authorization (login) process for the player on the platform, in order to access protected features and personalize the game experience. For example, you might prompt the player to log in so you can save their progress or enable social features.

```
async function authorize() {
    const options = {};
    if (Bridge.platform.id === "yandex") {
        options.scopes = true;
    }

    try {
        await Bridge.player.authorize(options);
        // player successfully authorized
    } catch (error) {
        // error, something went wrong
    }
}
```

Social Interactions

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

Share

Allow players to share game content or achievements on social media platforms.

```
Bridge.social.isShareSupported
```

Check if the share functionality is supported on the current platform.

```
const options = {};

switch (Bridge.platform.id) {
    case 'vk':
        options.link = 'YOUR_LINK';
        break;
    case 'facebook':
        options.image = 'A base64 encoded image to be shared';
        options.text = 'A text message to be shared.';
        break;
    case 'msn':
        options.title = 'A title to display';
        options.image = 'A base64 encoded image or image URL to be shared';
        options.text = 'A text message to be shared.';
        break;
}

Bridge.social.share(options)
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

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.

```
const options = {};
```

```
switch (Bridge.platform.id) {
    case 'vk':
        options.groupId = YOUR_GROUP_ID;
        break;
    case 'ok':
        options.groupId = YOUR_GROUP_ID;
        break;
}

Bridge.social.joinCommunity(options)
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

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.

```
const options = {};

switch (Bridge.platform.id) {
    case 'ok':
        options.text = 'Hello World!';
        break;
    case 'facebook':
        options.image = 'A base64 encoded image to be shared';
        options.text = 'A text message';
        break;
}

Bridge.social.inviteFriends(options)
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

Create Post

Let players create social media posts about their achievements or game updates directly from the game.

```
Bridge.social.isCreatePostSupported
```

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

```
const options = {};

switch (Bridge.platform.id) {
    case 'ok':
        options.media = [
            {
                type: 'text',
                text: 'Hello World!'
            },
            {
                type: 'link',
                url: 'https://apiok.ru'
            },
            {
                type: 'poll',
                question: 'Do you like our API?',
                answers: [
                    { text: 'Yes' },
                    { text: 'No' }
                ],
                options: 'SingleChoice,AnonymousVoting'
            }
        ];
        break;
}

Bridge.social.createPost(options)
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

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.

```
Bridge.social.addToFavorites()
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

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.

```
Bridge.social.addToHomeScreen()
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

Rate Game

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

```
Bridge.social.isRateSupported
```

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

```
Bridge.social.rate()
    .then(() => {
        // success
    })
```

```
    .catch(error => {
        // error
    });
```

External Links

Allow players to follow links to external websites (for example, your game's official site or related resources).

```
Bridge.social.isExternalLinksAllowed
```

Check if external links are allowed on the platform.

(If external links are not allowed, you should avoid using them or handle the restriction appropriately.)

Leaderboards

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

Leaderboards Type

Retrieve the type of leaderboards supported on the current platform.

```
Bridge.leaderboards.type
```

Type	Game Logic
`not_available`	Leaderboards are not available. Any leaderboard functionality must be disabled in the game.
`in_game`	Leaderboards are available. The game must use the `setScore` method to submit the player's score. The game should display custom in-game leaderboards using data from the `getEntries` method.
`native`	Leaderboards are available. The game must use the `setScore` method to submit the player's score. The game should **not** display custom in-game leaderboards because the leaderboards are shown via the platform's native interface (and the `getEntries` method will not return data).

Setup

Set up leaderboards in the **playgama-bridge-config.json** config file. For each leaderboard, add an `id`. You can override which ID is sent to the platform's native SDK if needed.

```
{
    "...": "...",
    "leaderboards": [
        {
            "id": "test_leaderboard",            // use this id in game logic
            "<ANY_PLATFORM_ID_HERE>": "<OVERRIDDEN_ID_FOR_PLATFORM_HERE>"
        }
    ]
}
```

Set Score

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

```
let leaderboardId =
"YOUR_LEADERBOARD_ID"; // the ID you specified in the config file
let score = 42;

Bridge.leaderboards.setScore(leaderboardId, score)
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

Get Entries

(Works only when `Bridge.leaderboards.type` *is* `in_game`*.)*

Retrieve entries from the leaderboard (including the player's rank and score) to display a custom leaderboard in your game.

```
let leaderboardId =
"YOUR_LEADERBOARD_ID"; // the ID you specified in the config file

Bridge.leaderboards.getEntries(leaderboardId)
    .then(entries => {
        entries.forEach(entry => {
            console.log('ID: ' + entry.id);
            console.log('Name: ' + entry.name);
            console.log('Photo: ' + entry.photo);
            console.log('Score: ' + entry.score);
            console.log('Rank: ' + entry.rank);
```

```
        });
    })
    .catch(error => {
        // error
    });
```

Achievements

Support

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

```
Bridge.achievements.isSupported
```

Check if retrieving a list of achievements is supported.

```
Bridge.achievements.isGetListSupported
```

Check if a built-in achievements UI (popup/overlay) is supported on the platform.

```
Bridge.payments.isNativePopupSupported
```

Unlock Achievement

Unlock an achievement for a player.

```
const options = {};

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

Bridge.achievements.unlock(options)
    .then(result => {
        // success
    })
```

```
    .catch(error => {
        // error
    });
```

Get List

Retrieve the list of achievements (in JSON format).

```
const options = {};

Bridge.achievements.getList(options)
    .then(list => {
        // success
        switch (Bridge.platform.id) {
            case 'y8':
                list.forEach(item => {
                    console.log('achievementid: ' + item.achievementid);
                    console.log('achievement: ' + item.achievement);
                    console.log('description: ' + item.description);
                    console.log('achievementkey: ' + item.achievementkey);
                    console.log('icon: ' + item.icon);
                    console.log('playerid: ' + item.playerid);
                    console.log('playername: ' + item.playername);
                });
                break;
        }
    })
    .catch(error => {
        // error
    });
```

Show Native Popup

Some platforms support a built-in achievement list displayed as an overlay.

```
const options = {};

Bridge.achievements.showNativePopup(options)
    .then(() => {
        // success
    })
    .catch(error => {
        // error
    });
```

In-Game Purchases

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

There are two types of purchases: **permanent** (e.g. ad removal, one-time unlocks) and **consumable** (e.g. in-game coins, consumable power-ups).

Support

Check if in-game purchases are supported on the platform.

```
Bridge.payments.isSupported
```

Setup

Set up in-game purchases in the **playgama-bridge-config.json** config file. For each product, add an `id` and fill in the required information for each platform. For example, for one product:

```
{
    "...": "...",
    "payments": [
        {
            "id": "test_product",
            "playgama": {
                "amount": 1           // int price in GAM (Playgama's currency)
            },
            "playdeck": {
                "amount": 1,          // int price in Telegram Stars (Playdeck
platform)
                "description": "TEST PRODUCT"
            }
        }
    ]
}
```

Purchase

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

```
Bridge.payments.purchase("test_product") // "test_product" is the product id
from the config
    .then(purchase => {
        // success
```

```
        console.log('Purchase completed, id:', purchase.id);
    })
    .catch(error => {
        // error
    });
```

Consume Purchase

For consumable purchases, "consume" the purchased item after it's used, to free it up for repurchase (and to update the player's inventory).

```
Bridge.payments.consumePurchase("test_product")
    .then(purchase => {
        // success
        console.log('Consume completed, id:', purchase.id);
    })
    .catch(error => {
        // error
    });
```

Catalog of All Items

Retrieve a list of all available in-game items that players can purchase, so you can display them in your game's store.

```
Bridge.payments.getCatalog()
    .then(catalogItems => {
        // success
        catalogItems.forEach(item => {
            console.log('ID: ' + item.id);
            console.log('Price: ' + item.price);
            console.log('Price Currency Code: ' + item.priceCurrencyCode);
            console.log('Price Value: ' + item.priceValue);
        });
    })
    .catch(error => {
        // error
    });
```

List of Purchased Items

Retrieve a list of items that the player has purchased (and not consumed, if consumable), to manage their inventory and provide access to purchased content.

If a user loses internet connection during a purchase, the purchase might remain unprocessed. To avoid this, check for any unprocessed purchases each time the game launches using this method.

```
Bridge.payments.getPurchases()
    .then(purchases => {
        // success
        purchases.forEach(purchase => {
            console.log('ID: ' + purchase.id);
        });
    })
    .catch(error => {
        // error
    });
```

Remote Configuration

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

Support

Check if remote configuration is supported on the platform.

```
Bridge.remoteConfig.isSupported
```

Load Values

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

```
const options = {};

switch (Bridge.platform.id) {
    case 'yandex':
        options.clientFeatures = [ // optional parameter
            { name: 'levels', value: '5' }
        ];
        break;
}

Bridge.remoteConfig.get(options)
    .then(data => {
        // example data: { showFullscreenAdOnStart: 'no', difficulty: 'hard' }
        console.log(data);
```

```
    })
    .catch(error => {
        // error
    });
```
