On This Page

API Reference v6.1

Changelog

1. **Bot Subscription API** The Bot Subscription API allows games to subscribe the player to the game's Messenger bot, if they are not subscribed.
2. **[Android Only] Home Screen Shortcut API** The Home Screen Shortcut API allows developers to surface a dialog in-game for players to save the game to their home screen.
3. **Matchmaking API (Update)** This release also includes an update for the Matchmaking API. We've added an additional parameter to specify whether you want the game to switch into the newly created context right after the player is matched, or wait until the player has clicked 'Play' in the toast.

FBInstant

Top level namespace for the Instant Games SDK.

player

Contains functions and properties related to the current player.

getID( )
A unique identifier for the player. A Facebook user's player ID will remain constant, and is scoped to a specific game. This means that different games will have different player IDs for the same user. This function should not be called until FBInstant.initializeAsync() has resolved.

Examples

```
// This function should be called after FBInstant.initializeAsync()
// resolves.
var playerID = FBInstant.player.getID();
```

Returns **string?** A unique identifier for the player.

getSignedPlayerInfoAsync( )

Fetch the player's unique identifier along with a signature that verifies that the identifier indeed comes from Facebook without being tampered with. This function should not be called until FBInstant.initializeAsync() has resolved.

Parameters

- `requestPayload` **string?** A developer-specified payload to include in the signed response.

Examples

```
// This function should be called after FBInstant.initializeAsync()
// resolves.
FBInstant.player.getSignedPlayerInfoAsync('my_metadata')
  .then(function (result) {
    // The verification of the ID and signature should happen on server side.
    SendToMyServer(
      result.getPlayerID(), // same value as FBInstant.player.getID()
      result.getSignature(),
      'GAIN_COINS',
      100);
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise<SignedPlayerInfo>** A promise that resolves with a #signedplayerinfo object.

canSubscribeBotAsync( )

Returns a promise that resolves with whether the player can subscribe to the game bot or not.

Examples

```
// This function should be called before FBInstant.player.subscribeBotAsync()
FBInstant.player.canSubscribeBotAsync().then(
  can_subscribe => console.log(can_subscribe)
);
// 'true'
```

Returns **Promise<boolean>** Whether a player can subscribe to the game bot or not. Developer can only call subscribeBotAsync() after checking canSubscribeBotAsync(), and the player will only be able to see this bot subscription dialog once for a specific game.

subscribeBotAsync( )

Request that the player subscribe the bot associated to the game. The API will reject if the subscription fails - else, the player will subscribe the game bot.

Examples

```
FBInstant.player.subscribeBotAsync().then(
  // Player is subscribed to the bot
).catch(function (e) {
  // Handle subscription failure
});
```

- Throws **INVALID_PARAM**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_REQUIRES_UPDATE**

Returns **Promise** A promise that resolves if player successfully subscribed to the game bot, or rejects if request failed or player chose to not subscribe.

getName( )

The player's localized display name. This function should not be called until FBInstant.startGameAsync() has resolved.

Examples

```
// This function should be called after FBInstant.startGameAsync()
// resolves.
var playerName = FBInstant.player.getName();
```

Returns **string?** The player's localized display name.

getPhoto( )

A url to the player's public profile photo. The photo will always be a square, and with dimensions of at least 200x200. When rendering it in the game, the exact dimensions should never be assumed to be constant. It's recommended to always scale the image to a desired size before rendering. The value will always be null until FBInstant.startGameAsync() resolves.

WARNING: Due to CORS, using these photos in the game canvas can cause it to be tainted, which will prevent the canvas data from being extracted. To prevent this, set the cross-origin attribute of the images you use to 'anonymous'.

Examples

```
var playerImage = new Image();
playerImage.crossOrigin = 'anonymous';
// This function should be called after FBInstant.startGameAsync()
// resolves.
playerImage.src = FBInstant.player.getPhoto();
```

Returns **string?** Url to the player's public profile photo.

getDataAsync( )

Retrieve data from the designated cloud storage of the current player.

Parameters

- `keys` **Array<string>** An array of unique keys to retrieve data for.

Examples

```
FBInstant.player
  .getDataAsync(['achievements', 'currentLife'])
  .then(function(data) {
    console.log('data is loaded');
    var achievements = data['achievements'];
    var currentLife = data['currentLife'];
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise<Object>** A promise that resolves with an object which contains the current key-value pairs for each key specified in the input array, if they exist.

setDataAsync( )

Set data to be saved to the designated cloud storage of the current player. The game can store up to 1MB of data for each unique player.

Parameters

- `data` **Object** An object containing a set of key-value pairs that should be persisted to cloud storage. The object must contain only serializable values - any non-serializable values will cause the entire modification to be rejected.

Examples

```
FBInstant.player
  .setDataAsync({
    achievements: ['medal1', 'medal2', 'medal3'],
    currentLife: 300,
  })
  .then(function() {
    console.log('data is set');
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves when the input values are set. NOTE: The promise resolving *does not* necessarily mean that the input has already been persisted. Rather, it means that the data was valid and has been scheduled to be saved. It also guarantees that all values that were set are now available in player.getDataAsync.

flushDataAsync( )

Immediately flushes any changes to the player data to the designated cloud storage. This function is expensive, and should primarily be used for critical changes where persistence needs to be immediate and known by the game. Non-critical changes should rely on the platform to persist them in the background. NOTE: Calls to player.setDataAsync will be rejected while this function's result is pending.

Examples

```
FBInstant.player
  .setDataAsync({
    achievements: ['medal1', 'medal2', 'medal3'],
    currentLife: 300,
  })
  .then(FBInstant.player.flushDataAsync)
  .then(function() {
    console.log('Data persisted to FB!');
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves when changes have been persisted successfully, and rejects if the save fails.

getStatsAsync( )

Retrieve stats from the designated cloud storage of the current player.

Parameters

- `keys` **Array<string>?** An optional array of unique keys to retrieve stats for. If the function is called without it, it will fetch all stats.

Examples

```
FBInstant.player
  .getStatsAsync(['level', 'zombiesSlain'])
  .then(function(stats)) {
    console.log('stats are loaded');
    var level = stats['level'];
```

```
    var zombiesSlain = stats['zombiesSlain'];
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise<Object>** A promise that resolves with an object which contains the current key-value pairs for each key specified in the input array, if they exist.

setStatsAsync( )

Set stats to be saved to the designated cloud storage of the current player.

Parameters

- `stats` **Object** An object containing a set of key-value pairs that should be persisted to cloud storage as stats, which can be surfaced or used in a variety of ways to benefit player engagement. The object must contain only numerical values - any non-numerical values will cause the entire modification to be rejected.

Examples

```
FBInstant.player
  .setStatsAsync({
    level: 5,
    zombiesSlain: 27,
  })
  .then(function() {
    console.log('data is set');
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves when the input values are set. NOTE: The promise resolving *does not* necessarily mean that the input has already been persisted. Rather, it means that the data was validated and has been scheduled to be saved. It also guarantees that all values that were set are now available in player.getStatsAsync.

incrementStatsAsync( )

Increment stats saved in the designated cloud storage of the current player.

Parameters

- `increments` **Object** An object containing a set of key-value pairs indicating how much to increment each stat in cloud storage. The object must contain only numerical values - any non-numerical values will cause the entire modification to be rejected.

Examples

```
FBInstant.player
  .incrementStatsAsync({
    level: 1,
    zombiesSlain: 17,
    rank: -1,
  })
 .then(function(stats)) {
    console.log('increments have been made! New values');
    var level = data['level'];
    var zombiesSlain = data['zombiesSlain'];
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise<Object>** A promise that resolves with an object which contains the updated key-value pairs for each key specified in the input dictionary. NOTE: The promise resolving *does not* necessarily mean that the changes have already been persisted. Rather, it means that the increments were valid and have been scheduled to be performed. It also guarantees that all values that were incremented are now available in player.getStatsAsync.

getConnectedPlayersAsync( )

Fetches an array of ConnectedPlayer objects containing information about players that are connected to the current player.

Examples

```
var connectedPlayers = FBInstant.player.getConnectedPlayersAsync()
  .then(function(players) {
    console.log(players.map(function(player) {
      return {
        id: player.getID(),
        name: player.getName(),
      }
    }));
  });
// [{id: '123456789', name: 'Paul Atreides'}, {id: '987654321', name: 'Duncan Idaho'}]
```

- Throws **NETWORK_FAILURE**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise<Array<ConnectedPlayer>>** A promise that resolves with a list of connected player objects. NOTE: This promise will not resolve until FBInstant.startGameAsync() has resolved.

context

Contains functions and properties related to the current game context.

getID( )

A unique identifier for the current game context. This represents a specific context that the game is being played in (for example, a particular messenger conversation or facebook post). The identifier will be null if game is being played in a solo context. This function should not be called until FBInstant.startGameAsync has resolved.

Examples

```
// This function should be called after FBInstant.startGameAsync()
// resolves.
var contextID = FBInstant.context.getID();
```

Returns **string?** A unique identifier for the current game context.

getType( )

The type of the current game context. POST - A facebook post. THREAD - A messenger thread. GROUP - A facebook group. SOLO - Default context, where the player is the only participant.

This function should not be called until FBInstant.startGameAsync has resolved.

Examples

```
// This function should be called after FBInstant.startGameAsync()
// resolves.
var contextType = FBInstant.context.getType();
```

Returns (**"POST"** | **"THREAD"** | **"GROUP"** | **"SOLO"**) Type of the current game context.

isSizeBetween( )

This function determines whether the number of participants in the current game context is between a given minimum and maximum, inclusive. If one of the bounds is null only the other bound will be checked against. It will always return the original result for the first call made in a context in a given game play session. Subsequent calls, regardless of arguments, will return the answer to the original query until a context change occurs and the query result is reset.

This function should not be called until FBInstant.startGameAsync has resolved.

This will be null if one or both of the supplied arguments are not valid, if we do not have a size available for the current context, or if the API is called before startGameAsync() resolves.

Parameters

- `minSize` **number?** The minimum bound of the context size query.
- `minSize` **number?** The maximum bound of the context size query.
- `maxSize` **number?**

Examples

```
console.log(FBInstant.context.isSizeBetween(3, 5)); (Context size = 4)
// {answer: true, minSize: 3, maxSize: 5}
```

```
console.log(FBInstant.context.isSizeBetween(5, 7)); (Context size = 4)
// {answer: false, minSize: 5, maxSize: 7}
```

```
console.log(FBInstant.context.isSizeBetween(2, 10)); (Context size = 3)
// {answer: true, minSize: 2, maxSize: 10}
console.log(FBInstant.context.isSizeBetween(4, 8)); (Still in same context)
// {answer: true, minSize: 2, maxSize: 10}
```

```
console.log(FBInstant.context.isSizeBetween(3, null)); (Context size = 4)
// {answer: true, minSize: 3, maxSize: null}
```

```
console.log(FBInstant.context.isSizeBetween(null, 3)); (Context size = 4)
// {answer: false, minSize: null, maxSize: 3}
```

```
console.log(FBInstant.context.isSizeBetween("test", 5)); (Context size = 4)
// null
```

```
console.log(FBInstant.context.isSizeBetween(0, 100)); (Context size = null)
// null
```

Returns **ContextSizeResponse?**

switchAsync( )

Request a switch into a specific context. If the player does not have permission to enter that context, or if the player does not provide permission for the game to enter that context, this will reject. Otherwise, the promise will resolve when the game has switched into the specified context.

Parameters

- `id` **string** ID of the desired context.

Examples

```
console.log(FBInstant.context.getID());
// 1122334455
FBInstant.context
  .switchAsync('1234567890')
  .then(function() {
    console.log(FBInstant.context.getID());
```

```
    // 1234567890
  });
```

- Throws **INVALID_PARAM**
- Throws **SAME_CONTEXT**
- Throws **NETWORK_FAILURE**
- Throws **USER_INPUT**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves when the game has switched into the specified context, or rejects otherwise.

chooseAsync( )

Opens a context selection dialog for the player. If the player selects an available context, the client will attempt to switch into that context, and resolve if successful. Otherwise, if the player exits the menu or the client fails to switch into the new context, this function will reject.

Parameters

- `options` **Object?** An object specifying conditions on the contexts that should be offered.
- `options.filters` **Array<ContextFilter>?** The set of filters to apply to the context suggestions.
- `options.maxSize` **number?** The maximum number of participants that a suggested context should ideally have.
- `options.minSize` **number?** The minimum number of participants that a suggested context should ideally have.

Examples

```
console.log(FBInstant.context.getID());
// 1122334455
FBInstant.context
  .chooseAsync()
  .then(function() {
    console.log(FBInstant.context.getID());
    // 1234567890
  });
```

```
console.log(FBInstant.context.getID());
// 1122334455
FBInstant.context
  .chooseAsync({
    filters: ['NEW_CONTEXT_ONLY'],
    minSize: 3,
  })
  .then(function() {
    console.log(FBInstant.context.getID());
    // 1234567890
  });
```

- Throws **INVALID_PARAM**
- Throws **SAME_CONTEXT**
- Throws **NETWORK_FAILURE**
- Throws **USER_INPUT**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves when the game has switched into the context chosen by the user. Otherwise, the promise will reject (if the user cancels out of the dialog, for example).

createAsync( )

Attempts to create or switch into a context between a specified player and the current player. The returned promise will reject if the player listed is not a Connected Player of the current player or if the player does not provide permission to enter the new context. Otherwise, the promise will resolve when the game has switched into the new context.

Parameters

- `playerID` **string** ID of the player

Examples

```
console.log(FBInstant.context.getID());
// 1122334455
FBInstant.context
  .createAsync('12345678')
  .then(function() {
    console.log(FBInstant.context.getID());
```

```
    // 5544332211
  });
```

- Throws **INVALID_PARAM**
- Throws **SAME_CONTEXT**
- Throws **NETWORK_FAILURE**
- Throws **USER_INPUT**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves when the game has switched into the new context, or rejects otherwise.

getPlayersAsync( )

Gets an array of #contextplayer objects containing information about active players — people who actively played the game in the current context in the last 90 days. This may include the current player.

Examples

```
var contextPlayers = FBInstant.context.getPlayersAsync()
  .then(function(players) {
    console.log(players.map(function(player) {
      return {
        id: player.getID(),
        name: player.getName(),
      }
    }));
  });
// [{id: '123456789', name: 'Luke'}, {id: '987654321', name: 'Leia'}]
```

- Throws **NETWORK_FAILURE**
- Throws **CLIENT_UNSUPPORTED_OPERATION**
- Throws **INVALID_OPERATION**

Returns **Promise<Array<ContextPlayer>>**

payments

[IN CLOSED BETA] Contains functions and properties related to payments and purchases of game products.

getCatalogAsync( )

Fetches the game's product catalog.

Examples

```
FBInstant.payments.getCatalogAsync().then(function (catalog) {
  console.log(catalog); // [{productID: '12345', ...}, ...]
});
```

- Throws **CLIENT_UNSUPPORTED_OPERATION**
- Throws **PAYMENTS_NOT_INITIALIZED**
- Throws **NETWORK_FAILURE**

Returns **Promise<Array<Product>>** The set of products that are registered to the game.

purchaseAsync( )

Begins the purchase flow for a specific product. Will immediately reject if called before FBInstant.startGameAsync() has resolved.

Parameters

- `purchaseConfig` **PurchaseConfig** The purchase's configuration details.

Examples

```
FBInstant.payments.purchaseAsync({
  productID: '12345',
  developerPayload: 'foobar',
}).then(function (purchase) {
  console.log(purchase);
  // {productID: '12345', purchaseToken: '54321', developerPayload: 'foobar', ...}
});
```

- Throws **CLIENT_UNSUPPORTED_OPERATION**
- Throws **PAYMENTS_NOT_INITIALIZED**
- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **INVALID_OPERATION**

Returns **Promise<Purchase>** A Promise that resolves when the product is successfully purchased by the player. Otherwise, it rejects.

getPurchasesAsync( )

Fetches all of the player's unconsumed purchases. As a best practice, the game should fetch the current player's purchases as soon as the client indicates that it is ready to perform payments-related operations. The game can then process and consume any purchases that are waiting to be consumed.

Examples

```
FBInstant.payments.getPurchasesAsync().then(function (purchases) {
  console.log(purchase);
  // [{productID: '12345', ...}, ...]
});
```

- Throws **CLIENT_UNSUPPORTED_OPERATION**
- Throws **PAYMENTS_NOT_INITIALIZED**
- Throws **NETWORK_FAILURE**

Returns **Promise<Array<Purchase>>** The set of purchases that the player has made for the game.

consumePurchaseAsync( )

Consumes a specific purchase belonging to the current player. Before provisioning a product's effects to the player, the game should request the consumption of the purchased product. Once the purchase is successfully consumed, the game should immediately provide the player with the effects of their purchase.

Parameters

- `purchaseToken` **string** The purchase token of the purchase that should be consumed.

Examples

```
FBInstant.payments.consumePurchaseAsync('54321').then(function () {
  // Purchase successfully consumed!
  // Game should now provision the product to the player
});
```

- Throws **CLIENT_UNSUPPORTED_OPERATION**

- Throws **PAYMENTS_NOT_INITIALIZED**
- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**

Returns **Promise** A Promise that resolves when the purchase has been consumed successfully.

onReady( )

Sets a callback to be triggered when Payments operations are available.

Parameters

- `callback` **Function** The callback function to be executed when Payments are available.

Examples

```
FBInstant.payments.onReady(function () {
  console.log('Payments Ready!')
});
```

Returns **void**

getLocale( )

The current locale. Use this to determine what language the current game should be localized with. The value will not be accurate until FBInstant.startGameAsync() resolves.

Examples

```
// This function should be called after FBInstant.startGameAsync()
// resolves.
var locale = FBInstant.getLocale(); // 'en_US'
```

Returns **string?** The current locale.

getPlatform( )

The platform on which the game is currently running. The value will always be null until FBInstant.initializeAsync() resolves.

Examples

```
// This function should be called after FBInstant.initializeAsync()
// resolves.
var platform = FBInstant.getPlatform(); // 'IOS'
```

Returns **Platform?**

getSDKVersion( )

The string representation of this SDK version.

Examples

```
// This function should be called after FBInstant.initializeAsync()
// resolves.
var sdkVersion = FBInstant.getSDKVersion(); // '2.0'
```

Returns **string** The SDK version.

initializeAsync( )

Initializes the SDK library. This should be called before any other SDK functions.

Examples

```
FBInstant.initializeAsync().then(function() {
  // Many properties will be null until the initialization completes.
  // This is a good place to fetch them:
  var locale = FBInstant.getLocale(); // 'en_US'
  var platform = FBInstant.getPlatform(); // 'IOS'
  var sdkVersion = FBInstant.getSDKVersion(); // '3.0'
  var playerID = FBInstant.player.getID();
});
```

- Throws **INVALID_OPERATION**

Returns **Promise** A promise that resolves when the SDK is ready to use.

setLoadingProgress( )

Report the game's initial loading progress.

Parameters

- `percentage` **number** A number between 0 and 100.

Examples

```
FBInstant.setLoadingProgress(50); // Assets are 50% loaded
```

Returns **void**

getSupportedAPIs( )

Provides a list of API functions that are supported by the client.

Examples

```
// This function should be called after FBInstant.initializeAsync()
// resolves.
FBInstant.getSupportedAPIs();
// ['getLocale', 'initializeAsync', 'player.getID', 'context.getType', ...]
```

Returns **Array<string>** List of API functions that the client explicitly supports.

getEntryPointData( )

Returns any data object associated with the entry point that the game was launched from.

The contents of the object are developer-defined, and can occur from entry points on different platforms. This will return null for older mobile clients, as well as when there is no data associated with the particular entry point.

Examples

```
// This function should be called after FBInstant.initializeAsync()
// resolves.
const entryPointData = FBInstant.getEntryPointData();
```

Returns **Object?** Data associated with the current entry point.

getEntryPointAsync( )

Returns the entry point that the game was launched from

Examples

```
// This function should be called after FBInstant.startGameAsync()
// resolves.
FBInstant.getEntryPointAsync().then(entrypoint => console.log(entrypoint));
// 'admin_message'
```

Returns **string** The name of the entry point from which the user started the game.

This function should be called after FBInstant.startGameAsync() resolves.

setSessionData( )

Sets the data associated with the individual gameplay session for the current context.

This function should be called whenever the game would like to update the current session data. This session data may be used to populate a variety of payloads, such as game play webhooks.

Parameters

- `sessionData` **Object** An arbitrary data object, which must be less than or equal to 1000 characters when stringified.

Examples

```
FBInstant.setSessionData({coinsEarned: 10, eventsSeen: ['start', ...]});
```

Returns **void**

startGameAsync( )

This indicates that the game has finished initial loading and is ready to start. Context information will be up-to-date when the returned promise resolves.

Examples

```
FBInstant.startGameAsync().then(function() {
  myGame.start();
});
```

- Throws **INVALID_PARAM**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves when the game should start.

shareAsync( )

This invokes a dialog to let the user share specified content, either as a message in Messenger or as a post on the user's timeline. A blob of data can be attached to the share which every game session launched from the share will be able to access from FBInstant.getEntryPointData(). This data must be less than or equal to 1000 characters when stringified. The user may choose to cancel the share action and close the dialog, and the returned promise will resolve when the dialog is closed regardless if the user actually shared the content or not.

Parameters

- `payload` **SharePayload** Specify what to share. See example for details.

Examples

```
FBInstant.shareAsync({
  intent: 'REQUEST',
  image: base64Picture,
  text: 'X is asking for your help!',
  data: { myReplayData: '...' },
}).then(function() {
  // continue with the game.
});
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**
- Throws **INVALID_OPERATION**

Returns **Promise** A promise that resolves when the share is completed or cancelled.

updateAsync( )

Informs Facebook of an update that occurred in the game. This will temporarily yield control to Facebook and Facebook will decide what to do based on what the update is. The returned promise will resolve/reject when Facebook returns control to the game.

Parameters

- `payload` **UpdatePayload** A payload that describes the update.

Examples

```
// This will post a custom update. If the game is played in a messenger
// chat thread, this will post a message into the thread with the specified
// image and text message. And when people launch the game from this
// message, those game sessions will be able to access the specified blob
// of data through FBInstant.getEntryPointData().
FBInstant.updateAsync({
  action: 'CUSTOM',
  cta: 'Join The Fight',
  image: base64Picture,
```

```
  text: {
    default: 'X just invaded Y\'s village!',
    localizations: {
      ar_AR: 'X \u0641\u0642\u0637 \u063A\u0632\u062A ' +
      '\u0642\u0631\u064A\u0629 Y!',
      en_US: 'X just invaded Y\'s village!',
      es_LA: '\u00A1X acaba de invadir el pueblo de Y!',
    }
  }
  template: 'VILLAGE_INVASION',
  data: { myReplayData: '...' },
  strategy: 'IMMEDIATE',
  notification: 'NO_PUSH',
}).then(function() {
  // closes the game after the update is posted.
  FBInstant.quit();
});
```

- Throws **INVALID_PARAM**
- Throws **PENDING_REQUEST**
- Throws **INVALID_OPERATION**

Returns **Promise** A promise that resolves when Facebook gives control back to the game.

switchGameAsync( )

Request that the client switch to a different Instant Game. The API will reject if the switch fails - else, the client will load the new game.

Parameters

- `appID` **string** The Application ID of the Instant Game to switch to. The application must be an Instant Game, and must belong to the same business as the current game. To associate different games with the same business, you can use Business Manager: https://developers.facebook.com/docs/apps/business-manager#update-business.
- `data` **string?** An optional data payload. This will be set as the entrypoint data for the game being switched to. Must be less than or equal to 1000 characters when stringified.

Examples

```
FBInstant.switchGameAsync('12345678').catch(function (e) {
  // Handle game change failure
});
```

- Throws **USER_INPUT**
- Throws **INVALID_PARAM**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_REQUIRES_UPDATE**

Returns **Promise**

canCreateShortcutAsync( )

Returns whether or not the user is eligible to have shortcut creation requested.

Will return false if createShortcutAsync was already called this session or the user is ineligible for shortcut creation.

Examples

```
FBInstant.canCreateShortcutAsync()
  .then(function(canCreateShortcut) {
    if (canCreateShortcut) {
      FBInstant.createShortcutAsync()
        .then(function() {
          // Shortcut created
        })
        .catch(function() {
          // Shortcut not created
        });
    }
  });
```

- Throws **PENDING_REQUEST**
- Throws **CLIENT_REQUIRES_UPDATE**
- Throws **INVALID_OPERATION**

Returns **Promise<boolean>** Promise that resolves with true if the game can request the player create a shortcut to the game, and false otherwise

createShortcutAsync( )

Prompts the user to create a shortcut to the game if they are eligible to Can only be called once per session. (see canCreateShortcutAsync)

Examples

```
FBInstant.canCreateShortcutAsync()
  .then(function(canCreateShortcut) {
    if (canCreateShortcut) {
      FBInstant.createShortcutAsync()
        .then(function() {
          // Shortcut created
        })
        .catch(function() {
          // Shortcut not created
        });
    }
  });
```

- Throws **USER_INPUT**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_REQUIRES_UPDATE**
- Throws **INVALID_OPERATION**

Returns **Promise**

quit( )

Quits the game.

Examples

```
FBInstant.quit();
```

Returns **void**

logEvent( )

Log an app event with FB Analytics. See https://developers.facebook.com/docs/javascript/reference/v2.8#app_events for more details about FB Analytics.

Parameters

- `eventName` **string** Name of the event. Must be 2 to 40 characters, and can only contain '_', '-', ' ', and alphanumeric characters.
- `valueToSum` **number** An optional numeric value that FB Analytics can calculate a sum with.
- `parameters` **Object** An optional object that can contain up to 25 key-value pairs to be logged with the event. Keys must be 2 to 40 characters, and can only contain '_', '-', ' ', and alphanumeric characters. Values must be less than 100 characters in length.

Examples

```
var logged = FBInstant.logEvent(
  'my_custom_event',
  42,
  {custom_property: 'custom_value'},
);
```

Returns **APIError?** The error if the event failed to log; otherwise returns null.

onPause( )

Set a callback to be fired when a pause event is triggered.

Parameters

- `func` **Function** A function to call when a pause event occurs.

Examples

```
FBInstant.onPause(function() {
  console.log('Pause event was triggered!');
})
```

Returns **void**

getInterstitialAdAsync( )

Attempt to create an instance of interstitial ad. This instance can then be preloaded and presented.

Parameters

- `placementID` **string** The placement ID that's been setup in your Audience Network settings.

Examples

```
FBInstant.getInterstitialAdAsync(
  'my_placement_id',
).then(function(interstitial) {
  interstitial.getPlacementID(); // 'my_placement_id'
});
```

- Throws **ADS_TOO_MANY_INSTANCES**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves with a #adinstance, or rejects with a #apierror if it couldn't be created.

getRewardedVideoAsync( )

Attempt to create an instance of rewarded video. This instance can then be preloaded and presented.

Parameters

- `placementID` **string** The placement ID that's been setup in your Audience Network settings.

Examples

```
FBInstant.getRewardedVideoAsync(
  'my_placement_id',
).then(function(rewardedVideo) {
```

```
  rewardedVideo.getPlacementID(); // 'my_placement_id'
});
```

- Throws **ADS_TOO_MANY_INSTANCES**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise** A promise that resolves with a #adinstance, or rejects with a #apierror if it couldn't be created.

matchPlayerAsync( )

Attempts to match the current player with other users looking for people to play with. If successful, a new Messenger group thread will be created containing the matched players and the player will be context switched to that thread. The default minimum and maximum number of players in one matched thread are 2 and 20 respectively, depending on how many players are trying to get matched around the same time. The values can be changed in fbapp-config.json. See the [Bundle Config documentation]https://developers.facebook.com/docs/games/instant-games/bundle-config for documentation about fbapp-config.json.

Parameters

- `matchTag` **string?** Optional extra information about the player used to group them with similar players. Players will only be grouped with other players with exactly the same tag. The tag must only include letters, numbers, and underscores and be 100 characters or less in length.
- `switchContextWhenMatched` **boolean** Optional extra parameter that specifies whether the player should be immediately switched to the new context when a match is found. By default this will be false which will mean the player needs explicitly press play after being matched to switch to the new context.

Examples

```
FBInstant
  .matchPlayerAsync('level1')
  .then(function() {
    console.log(FBInstant.context.getID());
    // 12345
  });
```

```
FBInstant
  .matchPlayerAsync()
  .then(function() {
```

```
    console.log(FBInstant.context.getID());
    // 3456
  });
```

```
FBInstant
  .matchPlayerAsync(null, true)
  .then(function() {
    console.log(FBInstant.context.getID());
    // 3456
  });
```

- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **USER_INPUT**
- Throws **PENDING_REQUEST**
- Throws **CLIENT_UNSUPPORTED_OPERATION**
- Throws **INVALID_OPERATION**

Returns **Promise** A promise that resolves when the player has been added to a group thread and switched into the thread's context.

checkCanPlayerMatchAsync( )

Checks if the current player is eligible for the matchPlayerAsync API.

Examples

```
FBInstant
  .checkCanPlayerMatchAsync()
  .then(canMatch => {
    if (canMatch) {
      FBInstant.matchPlayerAsync('level1');
    }
  });
```

- Throws **NETWORK_FAILURE**
- Throws **CLIENT_UNSUPPORTED_OPERATION**

Returns **Promise<boolean>** A promise that resolves with true if the player is eligible to match with other players and false otherwise.

getLeaderboardAsync( )

Fetch a specific leaderboard belonging to this Instant Game.

Parameters

- `name` **string** The name of the leaderboard. Each leaderboard for an Instant Game must have its own distinct name.

Examples

```
FBInstant.getLeaderboardAsync('my_awesome_leaderboard')
  .then(leaderboard => {
    console.log(leaderboard.getName()); // 'my_awesome_leaderboard'
  });
```

- Throws **LEADERBOARD_NOT_FOUND**
- Throws **NETWORK_FAILURE**
- Throws **CLIENT_UNSUPPORTED_OPERATION**
- Throws **INVALID_OPERATION**
- Throws **INVALID_PARAM**

Returns **Promise<Leaderboard>** A promise that resolves with the matching leaderboard, rejecting if one is not found.

LocalizationsDict

Represents a mapping from locales to translations of a given string. Each property is an optional five-character Facebook locale code of the form xx_XX. See https://www.facebook.com/translations/FacebookLocales.xml for a complete list of supported locale codes.

Type: Object

APIError

An API Error returned by the Instant Games SDK

code

The relevant error code

Type: ErrorCodeType

message

A message describing the error

Type: string

ConnectedPlayer

Represents information about a player who is connected to the current player.

getID( )

Get the id of the connected player.

Returns **string** The ID of the connected player

getName( )

Get the player's full name.

Returns **string?** The player's full name

getPhoto( )

Get the player's public profile photo.

Returns **string?** A url to the player's public profile photo

SignedPlayerInfo

Represents information about the player along with a signature to verify that it indeed comes from Facebook.

getPlayerID( )

Get the id of the player.

Examples

```
FBInstant.player.getSignedPlayerInfoAsync()
  .then(function (result) {
    result.getPlayerID(); // same value as FBInstant.player.getID()
  });
```

Returns **string** The ID of the player

getSignature( )

A signature to verify this object indeed comes from Facebook. The string is base64url encoded and signed with an HMAC version of your App Secret, based on the OAuth 2.0 spec.

You can validate it with the following 4 steps:

- Split the signature into two parts delimited by the '.' character.
- Decode the first part (the encoded signature) with base64url encoding.
- Decode the second part (the response payload) with base64url encoding, which should be a string representation of a JSON object that has the following fields: ** algorithm - always equals to HMAC-SHA256 ** issued_at - a unix timestamp of when this response was issued. ** player_id - unique identifier of the player. ** request_payload - the requestPayload string you specified when calling

FBInstant.player.getSignedPlayerInfoAsync.

- Hash the whole response payload string using HMAC SHA-256 and your app secret and confirm that it is equal to the encoded signature.
- You may also wish to validate the issued_at timestamp in the response payload to ensure the request was made recently.

Signature validation should only happen on your server. Never do it on the client side as it will compromise your app secret key.

Examples

```
FBInstant.player.getSignedPlayerInfoAsync()
  .then(function (result) {
    result.getSignature();
    // Eii6e636mz5J47sfqAYEK40jYAwoFqi3x5bxHkPG4Q4.eyJhbGdvcml0aG0iOiJITUFDLVNIQTI1NiIsIml
  });
```

Returns **string** The signature string.

ContextPlayer

Represents information about a player who is in the context that the current player is playing in.

getID( )

Get the id of the context player.

Returns **string** The ID of the context player

getName( )

Get the player's localized display name.

Returns **string?** The player's localized display name.

getPhoto( )

Get the player's public profile photo.

Returns **string?** A url to the player's public profile photo

AdInstance

Represents an instance of an ad.

getPlacementID( )

Return the Audience Network placement ID of this ad instance.

loadAsync( )

Preload the ad. The returned promise resolves when the preload completes, and rejects if it failed.

Examples

```
FBInstant.getInterstitialAdAsync(
  'my_placement_id',
).then(function(interstitial) {
  return interstitial.loadAsync();
}).then(function() {
  // Ad loaded
});
```

- Throws **ADS_FREQUENT_LOAD**
- Throws **ADS_NO_FILL**
- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**

Returns **Promise**

showAsync( )

Present the ad. The returned promise resolves when user finished watching the ad, and rejects if it failed to present or was closed during the ad.

Examples

```
var ad = null;
FBInstant.getRewardedVideoAsync(
  'my_placement_id',
).then(function(rewardedVideo) {
  ad = rewardedVideo;
  return ad.loadAsync();
}).then(function() {
  // Ad loaded
  return ad.showAsync();
}).then(function() {
  // Ad watched
});
```

- Throws **ADS_NOT_LOADED**
- Throws **INVALID_PARAM**
- Throws **NETWORK_FAILURE**
- Throws **INVALID_OPERATION**

Returns **Promise**

Product

Represents a game's product information.

Type: Object

Properties

- `title` **string** The title of the product
- `productID` **string** The product's game-specified identifier
- `description` **string?** The product description
- `imageURI` **string?** A link to the product's associated image
- `price` **string** The price of the product

- `priceCurrencyCode` **string** The currency code for the product

ContextFilter

A filter that may be applied to a Context Choose operation 'NEW_CONTEXT_ONLY' - Prefer to only surface contexts the game has not been played in before. 'INCLUDE_EXISTING_CHALLENGES' - Include the "Existing Challenges" section, which surfaces actively played-in contexts that the player is a part of. 'NEW_PLAYERS_ONLY' - In sections containing individuals, prefer people who have not played the game.

Type: (`"NEW_CONTEXT_ONLY"` | `"INCLUDE_EXISTING_CHALLENGES"` | `"NEW_PLAYERS_ONLY"`)

Platform

Represents the current platform that the user is playing on.

Type: (`"IOS"` | `"ANDROID"` | `"WEB"` | `"MOBILE_WEB"`)

ContextSizeResponse

The answer field is true if the current context size is between the minSize and maxSize values that are specified in the object, and false otherwise.

Type: {answer: boolean, minSize: number?, maxSize: number?}

Purchase

Represents an individual purchase of a game product.

Type: Object

Properties

- `developerPayload` **string?** A developer-specified string, provided during the purchase of the product
- `paymentID` **string** The identifier for the purchase transaction
- `productID` **string** The product's game-specified identifier
- `purchaseTime` **string** Unix timestamp of when the purchase occurred

- `purchaseToken` **string** A token representing the purchase that may be used to consume the purchase
- `signedRequest` **SignedPurchaseRequest** Server-signed encoding of the purchase request

Leaderboard

An Instant Game leaderboard

getName( )

The name of the leaderboard.

Examples

```
FBInstant.getLeaderboardAsync('my_leaderboard')
  .then(function(leaderboard) {
    console.log(leaderboard.getName()); // my_leaderboard
  });
```

Returns **string**

getContextID( )

The ID of the context that the leaderboard is associated with, or null if the leaderboard is not tied to a particular context.

Examples

```
FBInstant.getLeaderboardAsync('contextual_leaderboard')
  .then(function(leaderboard) {
    console.log(leaderboard.getContextID()); // 12345678
  });
```

```
FBInstant.getLeaderboardAsync('global_leaderboard')
  .then(function(leaderboard) {
```

```
    console.log(leaderboard.getContextID()); // null
  });
```

Returns **string?**

getEntryCountAsync( )

Fetches the total number of player entries in the leaderboard.

Examples

```
FBInstant.getLeaderboardAsync('my_leaderboard')
  .then(function(leaderboard) {
    return leaderboard.getEntryCountAsync();
  })
  .then(function(count) { console.log(count); }); // 24
```

- Throws **NETWORK_FAILURE**
- Throws **RATE_LIMITED**

Returns **Promise<number>** A unique identifier for the player.

setScoreAsync( )

Updates the player's score. If the player has an existing score, the old score will only be replaced if the new score is better than it. NOTE: If the leaderboard is associated with a specific context, the game must be in that context to set a score for the player.

Parameters

- `score` **number** The new score for the player. Must be a 64-bit integer number.
- `extraData` **string?= // 2** Metadata to associate with the stored score. Must be less than 2KB in size.

Examples

```
FBInstant.getLeaderboardAsync('my_leaderboard')
  .then(function(leaderboard) {
```

```
    return leaderboard.setScoreAsync(42, '{race: "elf", level: 3}');
  })
  .then(function(entry) {
    console.log(entry.getScore()); // 42
    console.log(entry.getExtraData()); // '{race: "elf", level: 3}'
  });
```

- Throws **LEADERBOARD_WRONG_CONTEXT**
- Throws **NETWORK_FAILURE**
- Throws **INVALID_PARAM**
- Throws **INVALID_OPERATION**
- Throws **RATE_LIMITED**

Returns **Promise<LeaderboardEntry>** Resolves with the current leaderboard entry for the player after the update.

getPlayerEntryAsync( )

Retrieves the leaderboard's entry for the current player, or null if the player has not set one yet.

Examples

```
FBInstant.getLeaderboardAsync('my_leaderboard')
  .then(function(leaderboard) {
    return leaderboard.getPlayerEntryAsync();
  })
  .then(function(entry) {
    console.log(entry.getRank()); // 2
    console.log(entry.getScore()); // 42
    console.log(entry.getExtraData()); // '{race: "elf", level: 3}'
  });
```

- Throws **NETWORK_FAILURE**
- Throws **INVALID_OPERATION**
- Throws **RATE_LIMITED**

Returns **Promise<LeaderboardEntry>?** Resolves with the current leaderboard entry for the player.

getEntriesAsync( )

Retrieves a set of leaderboard entries, ordered by score ranking in the leaderboard.

Parameters

- `count` **number** The number of entries to attempt to fetch from the leaderboard. Defaults to 10 if not specified. Up to a maximum of 100 entries may be fetched per query.
- `offset` **number** The offset from the top of the leaderboard that entries will be fetched from.

Examples

```
FBInstant.getLeaderboardAsync('my_leaderboard')
  .then(function(leaderboard) {
    return leaderboard.getEntriesAsync();
  })
  .then(function(entries) {
    console.log(entries.length); // 10
    console.log(entries[0].getRank()); // 1
    console.log(entries[0].getScore()); // 42
    console.log(entries[1].getRank()); // 2
    console.log(entries[1].getScore()); // 40
  });
```

```
FBInstant.getLeaderboardAsync('my_leaderboard')
  .then(function(leaderboard) {
    return leaderboard.getEntriesAsync(5, 3);
  })
  .then(function(entries) {
    console.log(entries.length); // 5
    console.log(entries[0].getRank()); // 4
    console.log(entries[0].getScore()); // 34
    console.log(entries[1].getRank()); // 5
    console.log(entries[1].getScore()); // 31
  });
```

- Throws **NETWORK_FAILURE**
- Throws **RATE_LIMITED**

Returns **Promise<Array<LeaderboardEntry>>** Resolves with the leaderboard entries that match the query.

SharePayload

Represents content to be shared by the user.

Type: Object

Properties

- `intent` (**`"INVITE"` | `"REQUEST"` | `"CHALLENGE"` | `"SHARE"`**) Indicates the intent of the share.
- `image` **string** A base64 encoded image to be shared.
- `text` **string** A text message to be shared.
- `data` **Object?** A blob of data to attach to the share. All game sessions launched from the share will be able to access this blob through FBInstant.getEntryPointData().

ErrorCode

Error codes that may be returned by the Instant Games API

Properties

- `ADS_FREQUENT_LOAD` **string** Ads are being loaded too frequently.
- `ADS_NO_FILL` **string** We were not able to serve ads to the current user. This can happen if the user has opted out of interest-based ads on their device, or if we do not have ad inventory to show for that user.
- `ADS_NOT_LOADED` **string** Attempted to show an ad that has not been loaded successfully.
- `ADS_TOO_MANY_INSTANCES` **string** There are too many concurrent ad instances. Load and show existing ad instances before creating new ones.
- `ANALYTICS_POST_EXCEPTION` **string** The analytics API experienced a problem while attempting to post an event.
- `CLIENT_REQUIRES_UPDATE` **string** [Deprecated] - The client requires an update to access the feature that returned this result. If this result is returned on web, it means the feature is not supported by the web client yet. Deprecated in favor of CLIENT_UNSUPPORTED_OPERATION in v5.0 and above
- `CLIENT_UNSUPPORTED_OPERATION` **string** The client does not support the current operation. This may be due to lack of support on the client version or platform, or because the operation is not allowed for the game or player.
- `INVALID_OPERATION` **string** The requested operation is invalid or the current game state. This may include requests that violate limitations, such as exceeding storage thresholds, or are not available in a certain state, such as making a context-specific request in a solo context.
- `INVALID_PARAM` **string** The parameter(s) passed to the API are invalid. Could indicate an incorrect type, invalid number of arguments, or a semantic issue (for example, passing an unserializable object to a

serializing function).

- `LEADERBOARD_NOT_FOUND` **string** No leaderboard with the requested name was found. Either the leaderboard does not exist yet, or the name did not match any registered leaderboard configuration for the game.
- `LEADERBOARD_WRONG_CONTEXT` **string** Attempted to write to a leaderboard that's associated with a context other than the one the game is currently being played in.
- `NETWORK_FAILURE` **string** The client experienced an issue with a network request. This is likely due to a transient issue, such as the player's internet connection dropping.
- `PENDING_REQUEST` **string** Represents a rejection due an existing request that conflicts with this one. For example, we will reject any calls that would surface a Facebook UI when another request that depends on a Facebook UI is pending.
- `RATE_LIMITED` **string** Some APIs or operations are being called too often. This is likely due to the game calling a particular API an excessive amount of times in a very short period. Reducing the rate of requests should cause this error to go away.
- `SAME_CONTEXT` **string** The game attempted to perform a context switch into the current context.
- `UNKNOWN` **string** An unknown or unspecified issue occurred. This is the default error code returned when the client does not specify a code.
- `USER_INPUT` **string** The user made a choice that resulted in a rejection. For example, if the game calls up the Context Switch dialog and the player closes it, this error code will be included in the promise rejection.

Examples

```
FBInstant.startGameAsync().catch(function(e) {
  console.log(e);
});
// {code: 'CLIENT_UNSUPPORTED_OPERATION', message: '...'}
```

UpdateAction

Represents the type of the update action to perform.

Properties

- `CUSTOM` **string** A custom update, with all content specified by the game.
- `LEADERBOARD` **string** An update associated with an Instant Game leaderboard.

ErrorCodeType

An Instant Games error code, one of #errorcode

Type: string

SignedPurchaseRequest

Type: string

Examples

```
Eii6e636mz5J47sfqAYEK40jYAwoFqi3x5bxHkPG4Q4.eyJhbGdvcml0aG0iOiJITUFDLVNIQTI1NiIsImlzc3VlZF
```

PurchaseConfig

The configuration of a purchase request for a product registered to the game.

Type: Object

Properties

- `productID` **string** The identifier of the product to purchase
- `developerPayload` **string?** An optional developer-specified payload, to be included in the returned purchase's signed request.

CustomUpdatePayload

Represents a custom update for FBInstant.updateAsync.

Type: Object

Properties

- `action` **UpdateAction** For custom updates, this should be 'CUSTOM'.
- `template` **string** ID of the template this custom update is using. Templates should be predefined in fbapp-config.json. See the [Bundle Config documentation]https://developers.facebook.com/docs/games/instant-

games/bundle-config for documentation about fbapp-config.json.

- `cta` **(string? | LocalizableContent?)** Optional call-to-action button text. By default we will use a localized 'Play' as the button text. To provide localized versions of your own call to action, pass an object with the default cta as the value of 'default' and another object mapping locale keys to translations as the value of 'localizations'.
- `image` **string** Data URL of a base64 encoded image.
- `text` **(string | LocalizableContent)** A text message, or an object with the default text as the value of 'default' and another object mapping locale keys to translations as the value of 'localizations'.
- `data` **Object?** A blob of data to attach to the update. All game sessions launched from the update will be able to access this blob through FBInstant.getEntryPointData(). Must be less than or equal to 1000 characters when stringified.
- `strategy` **string?** Specifies how the update should be delivered. This can be one of the following: 'IMMEDIATE' - The update should be posted immediately. 'LAST' - The update should be posted when the game session ends. The most recent update sent using the 'LAST' strategy will be the one sent. 'IMMEDIATE_CLEAR' - The update is posted immediately, and clears any other pending updates (such as those sent with the 'LAST' strategy). If no strategy is specified, we default to 'IMMEDIATE'.
- `notification` **string?** Specifies notification setting for the custom update. This can be 'NO_PUSH' or 'PUSH', and defaults to 'NO_PUSH'. Use push notification only for updates that are high-signal and immediately actionable for the recipients. Also note that push notification is not always guaranteed, depending on user setting and platform policies.

LeaderboardUpdatePayload

Represents a leaderboard update for FBInstant.updateAsync.

Type: Object

Properties

- `action` **UpdateAction** For a leaderboard update, this should be 'LEADERBOARD'. text. By default we will use a localized 'Play Now' as the button text.
- `name` **string** The name of the leaderboard to feature in the update.
- `text` **string?** Optional text message. If not specified, a localized fallback message will be provided instead.

LocalizableContent

Represents a string with localizations and a default value to fall back on.

Type: Object

Properties

- `default` **string** The default value of the string to use if the viewer's locale is not a key in the localizations object.
- `localizations` **LocalizationsDict** Specifies what string to use for viewers in each locale. See https://www.facebook.com/translations/FacebookLocales.xml for a complete list of supported locale values.

LeaderboardEntry

A score entry for an Instant Game leaderboard

getScore( )

Gets the score associated with the entry.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getScore()); // 9001
  });
```

Returns **number** Returns an integer score value.

getFormattedScore( )

Gets the score associated with the entry, formatted with the score format associated with the leaderboard.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getFormattedScore()); // '90.01 meters'
  });
```

Returns **string** Returns a formatted score.

getTimestamp( )

Gets the timestamp of when the leaderboard entry was last updated.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getTimestamp()); // 1515806355
  });
```

Returns **number** Returns a Unix timestamp.

getRank( )

Gets the rank of the player's score in the leaderboard.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getRank()); // 2
  });
```

Returns **number** Returns the entry's leaderboard ranking.

getExtraData( )

Gets the developer-specified payload associated with the score, or null if one was not set.

Examples

```
leaderboard.setScoreAsync(42, '{race: "elf", level: 3}');
  .then(function(entry) {
    console.log(entry.getExtraData()); // '{race: "elf", level: 3}'
  });
```

Returns **string?** An optional developer-specified payload associated with the score.

getPlayer( )

Gets information about the player associated with the entry.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getPlayer().getName()); // Sally
  });
```

Returns **LeaderboardPlayer**

LeaderboardPlayer

Details about the player associated with a score entry.

getName( )

Gets the player's localized display name.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getPlayer().getName()); // Sally
  });
```

Returns **string** The player's localized display name.

getPhoto( )

Returns a url to the player's public profile photo.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getPlayer().getPhoto()); // <photo_url>
  });
```

Returns **string?** Url to the player's public profile photo.

getID( )

Gets the game's unique identifier for the player.

Examples

```
leaderboard.setScoreAsync(9001)
  .then(function(entry) {
    console.log(entry.getPlayer().getID()); // 12345678
  });
```

Returns **string?** The game-scoped identifier for the player.

Platform Status

Developers Group

Marketing Partners

Bugs

News

Blog

Success Stories

Videos