CONSTRUCT GAME SERVICES SCRIPT INTERFACE

View online: https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/construct-game-services

The `ICGSObjectType` interface derives from IObjectType to add APIs specific to the Construct Game Services plugin. Construct Game Services is also written as CGS for short.

Using the REST APIs

Construct Game Services have a comprehensive set of REST APIs available. When writing code it is possible to use those entirely independently of this plugin. This option may be of interest to advanced users - see the Construct Game Services manual for details.

The CGS script interface also provides the `sessionKey` when signed in. This can be used to make REST API calls with the user's current authentication. This allows using the CGS script interface solely for authentication, and then using the resulting `sessionKey` to make REST API calls.

For completeness the CGS script interface does provide equivalent features to the plugin, but note this is a subset of all the available features in the REST API.

CGS events

"signinpopupblocked"

Triggered during sign in if the browser popup blocker prevented the sign in popup window from appearing. Use the `retryOpenSignInPopup()` method in a user input event to retry.

CGS APIs

isSignedIn

Read-only boolean indicating if the player is currently signed in.

canSignInPersistent

Read-only boolean indicating if persistent sign in with `signInPersistent()` is available.

playerId

A read-only string of the unique player ID when signed in.

playerName

A read-only string of the player display name when signed in.

gameId

A read-only string of the game ID signed in to when signed in.

sessionKey

When signed in, a string with the session key for this sign in. This can be used to make REST API calls using the same authentication.

async signInWithProvider(provider, gameId, options)

Attempt to sign the user in with a third-party identity service like Google or Microsoft. `provider` must be one of: `"Apple"`, `"BattleNet"`, `"BattleNetChina"`, `"Discord"`, `"Facebook"`, `"Github"`, `"Google"`, `"ItchIO"`, `"Microsoft"`, `"Reddit"`, `"Steam"`, `"X"`, `"Yandex"`. The game ID to sign in to from your Construct Game Services account must be provided. `options` is an optional object that may provide the following properties:

- `allowPersisting`: if `true` a successful sign in is remembered, and may be re-used in future with `signInPersistent()`. The default is `true`.
- `expiryMins`: the time in minutes that sessions remain active. The default is 1440 (24 hours).
- `popupWindowWidth` and `popupWindowHeight`: Signing in with a provider will open a popup window, and you can also specify the size of this popup window with these properties. The default is 800x800.

The method returns a Promise that resolves when sign in has completed successfully, or rejects if sign in fails or is cancelled.

> *Note that in some cases, sign in may be cancelled in a way that cannot be detected, and so the returned promise will not settle. Be sure to design your project with this in mind - for example do not block the user interface until sign in finishes, as you cannot reliably detect that.*

retryOpenSignInPopup()

After the `"signinpopupblocked"` event fires, make another attempt to open the sign in popup window to continue the sign in process. This should be used in a user input event such as a button click or mouse click, otherwise the browser popup blocker may still block the second attempt.

async signInPersistent(gameId)

Attempt to sign in re-using a previous successful sign in that allowed persisting. This can only be used when `canSignInPersistent` is true. This returns a Promise that resolves when the persistent sign in has completed.

async signOut()

Sign out of any account the user is currently signed in to, and also delete any remembered sign in if it allowed persisting. Locally this operation completes immediately. For completeness signing out will send a request in the background to ensure the session is ended on the server-side as well, but that is optional and if it fails the session will time out anyway. This method returns a Promise that resolves when the background request completes, but failure should not be considered significant.

async setPlayerName(name)

Set the name of the currently signed in player. In some cases the player name is obtained from the authentication provider, but in others a generic name will be used, in which case the player will likely want to change their name for features like leaderboards. Returns a Promise that resolves when the operation completes.

async submitScore(score, leaderboardId)

Submit a score to a leaderboard. `score` must be an integer (fractional scores are not supported). `leaderboardId` may be omitted to submit a score on the Construct Arcade, which does not require authentication. Otherwise it must be set to the leaderboard ID to submit the score to, in which case authentication is required. This method returns a Promise that resolves when the score has been successfully submitted.

async getLeaderboardScores(leaderboardId, options)

Request a page of scores from a given leaderboard ID. This does not require authentication. The returned scores can optionally be filtered with several options, which may include:

- `resultsPerPage`: the number of results to fetch. The default is 20.
- `page`: zero-based index of the page to fetch. The default is 0 to retrieve the first page.
- `country`: an ISO 3166-1 alpha-2 country code filter can be provided, e.g. "US" to only return scores submitted in the United States of America.
- `range`: a time range to filter results by, which if provided must be one of `"Daily"`, `"Weekly"`, `"Monthly"` or `"Yearly"`. For example using `"Daily"` will return today's

scores. Weekly leaderboards run from Monday to Sunday. If this property is omitted it returns all scores (which is the default).

- `rangeOffset`: an offset to return a prior time range when `range` is specified. For example specifying a `"Daily"` range with a `rangeOffset` of 1 will return yesterday's scores.

- `culture`: a locale to use for returned values. By default it will use the leaderboard's default culture.

This method returns a Promise that resolves with the following object upon success:

- `totalPageCount`: the total number of pages available

- `scores`: an array of returned scores, each being an object with the following properties:
  - `score`: a number representing this score.
  - `formattedScore`: a string with the score formatted according to the culture.
  - `rank`: a number representing the score rank.
  - `formattedRank`: a string with the rank formatted according to the culture.
  - `country`: a string of the ISO 3166-1 alpha-2 country code the score was submitted from.
  - `playerId`: a string of the unique player ID who submitted the score.
  - `playerName`: a string of the player display name who submitted the score.

---

async createCloudSave(opts)

Create or replace the data for a cloud save key. The `opts` parameter is an object which uses the following properties to specify options:

- `key` (required): the storage key to save to.

- `bucketId`: the bucket ID to save to. This can be omitted to use player private storage.

- `name`: an optional name to associate with the data, such as a file name.

- `data` (required): a string or Blob for the data to upload.

The method returns a Promise that resolves when the operation has completed.

---

async getCloudSave(opts)

Download the data previously uploaded for a key in a game bucket with `createCloudSave`. The `opts` parameter is an object which uses the following properties to specify options:

- `key` (required): the storage key to save to.
- `bucketId`: the bucket ID to save to. This can be omitted to use player private storage.
- `type`: a string of `"text"` or `"blob"` that determines the data type the returned Promise resolves with. The default is `"text"`.

The method returns a Promise that returns with a string when `type` is `"text"`, or a Blob if `type` is `"blob"`.