Kublai Documentation

Release 0.6.0

Top Free Games

Oct 26, 2017

Contents

Contents:

CHAPTER 1

Overview

What is Kublai? Kublai is a pomelo plugin for the Khan service.

It allows easy integration with your gaming backend, thus enabling easy-to-use clan management operations.

Features

- **Pomelo Plug-in** - Just point Kublai Plug-in in the direction of your Khan instance and you are good to go;
- **Additional Data** - Kublai builds on top of Khan to ensure that you make as least requests as possible to achieve what you want;
- **Up-To-Date** - Kublai is managed by the same team working on Khan, thus the parity with Khan's features is ensured.

Who's Using it

Well, right now, only us at TFG Co, are using it, but it would be great to get a community around the project. Hope to hear from you guys soon!

How To Contribute?

Just the usual: Fork, Hack, Pull Request. Rinse and Repeat. Also don't forget to include tests and docs (we are very fond of both).

CHAPTER 2

Using Kublai

Installing Kublai

You can install Kublai using npm:

```
npm install Kublai-plugin
```

Integrating Kublai plugin with your Pomelo application

Our sandbox application is a reference example of how to integrate Kublai into your pomelo application.

Initializing Kublai in your app

In your `app.js` file, add the following lines:

```
const kublaiPlugin = require('kublai-plugin')

// app configuration
app.configure('production|development', '<server-type>', () => {

  // your app configuration

  app.use(kublaiPlugin, {
    kublai: {
      khanUrl: 'http://localhost:8888/',  // you need to set this to your khan API url
      timeout: 500,                       // request timeout in milliseconds (default
↪value is 500ms)
    },
  })
})
```

Using Kublai in your handlers

Using it in your handlers is as easy as asking the app for it:

```
// in your handler initialization
const Handler = function (app) {
  this.app = app
  this.kublaiService = this.app.get('kublai')  // this gets a configured kublai␣
↪service instance
}
```

Then in your handler methods:

```
Handler.prototype.getPlayer = function (msg, session, next) {
  this.kublaiService.getPlayer(msg.gameID, msg.publicID, (error, res) => {
    if (error) {
      next(error, null)
    } else {
      next(null, res)
    }
  }
}
```

For a complete example, our test sandbox has a handler with all available methods.

CHAPTER 3

Kublai API

Callbacks

All callbacks in Kublai are called with an error and a response and have the form of:

```
function(error, response) {
  if (error) {
    // do something with error
    return
  }
  // do something with response
}
```

Whenever a response is returned it will match the same response as that of the operation in Khan's API.

If additional details are added to the response, those will be detailed in the specific method's docs.

Errors

For error reasons and payloads, please refer to Khan's API docs.

All messages received from Khan are wrapped into a KhanError object with the following fields

- message: `Could not process request: ${reason}. Operation: ${operation}.`
  - reason and operation are the fields received from Khan
- khan: Always true to help you identify khan errors
- meta: possible metadata related to the error (e.g. the error object that originated the khan error)

Healthcheck Methods

Healthcheck

Calls Khan's Healthcheck Route.

Signature

```
kublaiService.healthcheck(callback);
```

Game Methods

Create Game

Creates a new game with the given parameters, using Khan's Create Game Route.

Signature

```
kublaiService.createGame(
  publicId,
  name,
  metadata,
  membershipLevels,
  minLevelToAcceptApplication,
  minLevelToCreateInvitation,
  minLevelToRemoveMember,
  minLevelOffsetToRemoveMember,
  minLevelOffsetToPromoteMember,
  minLevelOffsetToDemoteMember,
  maxMembers,
  maxClansPerPlayer,
  cooldownAfterDelete,
  cooldownAfterDeny,
  options,                        // optional
  callback
);
```

Arguments

- `publicId`: game's public id;
- `name`: name for this game;
- `metadata`: any meta-data that needs to be stored for this game;
- `membershipLevels`: object with the available membership levels for this game (refer to khan Docs for more details);
- `minLevelToAcceptApplication`: a member cannot accept a player's application to join the clan unless their level is greater or equal to this parameter;

- `minLevelToCreateInvitation`: a member cannot invite a player to join the clan unless their level is greater or equal to this parameter;
- `minLevelToRemoveMember`: minimum membership level required to remove another member from the clan;
- `minLevelOffsetToRemoveMember`: a member cannot remove another member unless their level is at least `minLevelOffsetToRemoveMember` levels greater than the level of the member they wish to promote;
- `minLevelOffsetToPromoteMember`: a member cannot promote another member unless their level is at least `minLevelOffsetToPromoteMember` levels greater than the level of the member they wish to promote;
- `minLevelOffsetToDemoteMember`: a member cannot demote another member unless their level is at least `minLevelOffsetToDemoteMember` levels greater than the level of the member they wish to demote;
- `maxMembers`: maximum number of members a clan of this game can have;
- `maxClansPerPlayer`: maximum numbers of clans a player can be an approved member of.
- `cooldownAfterDelete`: a membership cannot be recreated after being deleted unless cooldown seconds have passed.
- `cooldownAfterDeny`: a membership cannot be recreated after being denied unless cooldown seconds have passed.
- `options`: optional object. Properties:
  - `maxPendingInvites`: a member cannot be invited if they have more than maxPendingInvites. Default value is -1 (unlimited).
  - `cooldownBeforeInvite`: a member cannot be invited to the clan after a previous application/invite unless cooldown seconds have passed. Default value is 0.
  - `cooldownBeforeApply`: a member cannot apply to the clan after a previous application/invite unless cooldown seconds have passed. Default value is 3600.

Update Game

Updates a game. If the game does not exist it gets created with the given parameters. This operation uses Khan's Update Game Route.

Signature

```
kublaiService.updateGame(
  publicId,
  name,
  metadata,
  membershipLevels,
  minLevelToAcceptApplication,
  minLevelToCreateInvitation,
  minLevelToRemoveMember,
  minLevelOffsetToRemoveMember,
  minLevelOffsetToPromoteMember,
  minLevelOffsetToDemoteMember,
  maxMembers,
  maxClansPerPlayer,
  cooldownAfterDelete,
  cooldownAfterDeny,
```

```
  options,                          // optional
  callback
)
```

Arguments

- `publicId`: game's public id;
- `name`: name for this game;
- `metadata`: any metadata that needs to be stored for this game;
- `membershipLevels`: object with the available membership levels for this game (refer to khan Docs for more details);
- `minLevelToAcceptApplication`: a member cannot accept a player's application to join the clan unless their level is greater or equal to this parameter;
- `minLevelToCreateInvitation`: a member cannot invite a player to join the clan unless their level is greater or equal to this parameter;
- `minLevelToRemoveMember`: minimum membership level required to remove another member from the clan;
- `minLevelOffsetToRemoveMember`: a member cannot remove another member unless their level is at least `minLevelOffsetToRemoveMember` levels greater than the level of the member they wish to promote;
- `minLevelOffsetToPromoteMember`: a member cannot promote another member unless their level is at least `minLevelOffsetToPromoteMember` levels greater than the level of the member they wish to promote;
- `minLevelOffsetToDemoteMember`: a member cannot demote another member unless their level is at least `minLevelOffsetToDemoteMember` levels greater than the level of the member they wish to demote;
- `maxMembers`: maximum number of members a clan of this game can have;
- `maxClansPerPlayer`: maximum numbers of clans a player can be an approved member of.
- `cooldownAfterDelete`: a membership cannot be recreated after being deleted unless cooldown seconds have passed.
- `cooldownAfterDeny`: a membership cannot be recreated after being denied unless cooldown seconds have passed.
- `options`: optional object. Properties:
  - `maxPendingInvites`: a member cannot be invited if they have more than maxPendingInvites. Default value is -1 (unlimited).
  - `cooldownBeforeInvite`: a member cannot be invited to the clan after a previous application/invite unless cooldown seconds have passed. Default value is 0.
  - `cooldownBeforeApply`: a member cannot apply to the clan after a previous application/invite unless cooldown seconds have passed. Default value is 3600.

Player Methods

Create Player

Creates a new player in a specific game. This operation uses Khan's Create Player Route.

Warning

This operation is not idempotent. If you want to create or update a player, please use the Update Player operation described below. If you try to create a player for which the public ID already exists in the specified game, you will get an error.

Signature

```
kublaiService.createPlayer(gameId, publicId, name, metadata, callback);
```

Arguments

- `gameId`: public ID for the player's game;
- `publicId`: public ID for the player;
- `name`: player's name;
- `metadata`: any player meta-data the game wants to store.

Update Player

Updates a player in a specific game. If the player does not exist, the player gets created. This operation uses Khan's Update Player Route.

Signature

```
kublaiService.updatePlayer(gameId, publicId, name, metadata, callback);
```

Arguments

- `gameId`: public ID for the player's game;
- `publicId`: public ID for the player;
- `name`: player's name;
- `metadata`: any player meta-data the game wants to store.

Get Player

Gets details about a player in a specific game. This operation uses Khan's Retrieve Player Route.

Signature

```
kublaiService.getPlayer(gameId, playerId, callback);
```

Arguments

- `gameId`: public ID for the player's game.
- `playerId`: public ID for the player.

Clan Routes

Create Clan

Creates a new clan. This operation uses Khan's Create Clan Route.

Signature

```
kublaiService.createClan(
  gameId,
  publicId,
  name,
  metadata,
  ownerPublicId,
  allowApplication,
  autoJoin,
  callback
);
```

Arguments

- `gameId`: public ID for the clan's game;
- `publicId`: public ID for the clan;
- `name`: clan's name;
- `metadata`: a JSON object representing any metadata required for the clan;
- `ownerPublicId`: clan's owner player public id;
- `allowApplication`: does this clan allow players to apply to it;
- `autoJoin`: do players that apply to this clan get automatically accepted;

Update Clan

Updates a clan. This operation uses Khan's Update Clan Route.

Signature

```
kublaiService.updateClan(
  gameId,
  publicId,
  name,
  metadata,
  ownerPublicId,
  allowApplication,
  autoJoin,
  callback
);
```

Arguments

- `gameId`: public ID for the clan's game;
- `publicId`: public ID for the clan;
- `name`: clan's name;
- `metadata`: a JSON object representing any meta-data required for the clan;
- `ownerPublicId`: clan's owner player public id;
- `allowApplication`: does this clan allow players to apply to it;
- `autoJoin`: do players that apply to this clan get automatically accepted;

Get Clan

Gets detailed information about a clan. This operation uses Khan's Retrieve Clan Route.

Signature

```
kublaiService.getClan(gameId, clanId, callback);
```

Arguments

- `gameId`: public ID for the clan's game.
- `clanId`: public ID for the clan.

Get Clan Summary

Gets summarized information about a clan. This operation uses Khan's Clan Summary Route.

Signature

```
kublaiService.getClanSummary(gameId, clanId, callback);
```

Arguments

- `gameId`: public ID for the clan's game.
- `clanId`: public ID for the clan.

List Clans Summary

Lists summarized information about the clans with the given IDs. This operation uses Khan's Clans Summary Route.

Signature

```
kublaiService.listClansSummary(gameId, clanIds, callback);
```

Arguments

- `gameId`: public ID for the clan's game.
- `clanIds`: list of clans public IDs.

List Clans

Gets a list of all clans in a game. This operation uses Khan's List Clans Route.

Warning

Depending on the number of clans in your game this can be a **VERY** expensive operation! Be wary of using this. A better way of getting clans is using clan search.

Signature

```
kublaiService.listClans(gameId, callback);
```

Arguments

- `gameId`: public ID for the clan's game.

Search Clans

Searches a clan by a specific term. This operation uses Khan's Search Clans Route.

Signature

```
kublaiService.searchClans(gameId, term, callback);
```

Arguments

- `gameId`: public ID for the clan's game.
- `term`: partial term to search for public ID or name.

Leave Clan

This operation should be used when the clan's owner decides to leave the clan. If there are no clan members left, the clan will be deleted. This operation uses Khan's Leave Clan Route.

Signature

```
kublaiService.leaveClan(gameId, clanId, callback);
```

Arguments

- `gameId`: public ID for the clan's game;
- `clanId`: public ID for the clan.

Transfer Clan Ownership

Allows the owner to transfer the clan's ownership to another clan member of their choice. The previous owner will then be a member with the maximum level allowed for the clan. This operation uses Khan's Transfer Clan Ownership Route.

Signature

```
kublaiService.transferClanOwnership(gameId, clanId, playerPublicId, callback);
```

Arguments

- `gameId`: public ID for the clan's game;
- `clanId`: public ID for the clan;
- `playerPublicId`: public ID for the player to be the new owner of the clan.

Membership Routes

Apply for Membership

Allows a player to ask to join the clan with the given `publicID`. If the clan's `autoJoin` property is true the member will be automatically approved. Otherwise, the membership must be approved by the clan owner or one of the clan members.

This operation uses Khan's Apply For Membership Route.

Signature

```
kublaiService.applyForMembership(gameId, clanId, level, playerPublicId, message,␣
↪callback);
```

Arguments

- `gameId`: public ID for the desired clan's game;
- `clanId`: public ID for the desired clan;
- `level`: membership level for the application;
- `playerPublicId`: public id for the player filing the application for the clan;
- `message`: message sent by the player when applying for the membership (**optional**);

Approve or Deny Membership

Allows the clan owner or a clan member to approve or deny a player's application to join the clan. The member's membership level must be at least the game's `minLevelToAcceptApplication`.

This operation uses Khan's Approve Or Deny Membership Route.

Signature

```
kublaiService.approveDenyMembershipApplication(
  gameId, clanId, action, playerPublicId, requestorPublicID, callback
);
```

Arguments

- `gameId`: public ID for the desired clan's game;
- `clanId`: public ID for the desired clan;
- `action`: action to be executed. Can be either `approve` or `deny`;
- `playerPublicId`: public id for the player that must be approved or denied;
- `requestorPublicId`: the public id of the clan member who will approve or deny the application.

Invite for Membership

Allows the clan owner or a clan member to invite a player to join the clan with the given `publicID`. If the request is made by a member of the clan, their membership level must be at least the game's `minLevelToCreateInvitation`. The membership must be approved by the player being invited.

This operation uses Khan's Invite for Membership Route.

Signature

```
kublaiService.inviteForMembership(
  gameId, clanId, level, playerPublicId, requestorPublicId, callback
);
```

Arguments

- `gameId`: public ID for the desired clan's game;
- `clanId`: public ID for the desired clan;
- `level`: membership level for the application;
- `playerPublicId`: public id for the player that is being invited;
- `requestorPublicId`: the public id of the clan member who is inviting the player.
- `message`: message sent by the player when inviting for the membership (**optional**);

Approve or Deny Membership Invitation

Allows a player member to approve or deny a player's invitation to join a given clan.

This operation uses Khan's Approve or Deny Membership Invitation Route.

Signature

```
kublaiService.approveDenyMembershipInvitation(
  gameId, clanId, action, playerPublicId, callback
);
```

Arguments

- `gameId`: public ID for the desired clan's game;
- `clanId`: public ID for the desired clan;
- `action`: action to be executed. Can be either `approve` or `deny`;
- `playerPublicId`: public id for the player that is accepting the invitation.

Promote or Demote Member

Allows the clan owner or a clan member to promote or demote another member. When promoting, the member's membership level will be increased by one, when demoting it will be decreased by one. The member's membership level must be at least `minLevelOffsetToPromoteMember` or `minLevelOffsetToDemoteMember` levels greater than the level of the player being promoted or demoted.

This operation uses Khan's Promote or Demote Member Route.

Signature

```
kublaiService.promoteDemoteMember(
  gameId, clanId, action, playerPublicId, requestorPublicId, callback
);
```

Arguments

- `gameId`: public ID for the desired clan's game;
- `clanId`: public ID for the desired clan;
- `action`: action to be executed. Can be either `promote` or `demote`;
- `playerPublicId`: public id for the player that is being promoted/demoted;
- `requestorPublicId`: the public id of the clan member who is promoting/demoting the player.

Delete Membership

Allows the clan owner or a clan member to remove another member from the clan. The member's membership level must be at least `minLevelToRemoveMember`. A member can leave the clan by sending the same `playerPublicID` and `requestorPublicID`.

This operation uses Khan's Delete Membership Route.

Signature

```
kublaiService.deleteMembership(
  gameId, clanId, playerPublicId, requestorPublicId, callback
);
```

Arguments

- `gameId`: public ID for the desired clan's game;
- `clanId`: public ID for the desired clan;
- `playerPublicId`: public id for the player that is leaving the clan;
- `requestorPublicId`: the public id of the clan member who is kicking the player.

Hook Routes

Create Hook

Creates a hook for the specified game for the given event. The hook URL will be called with the payload specified in Khan's Docs.

Signature

```
kublaiService.createHook(gameId, hookType, hookURL, callback);
```

Arguments

- `gameId`: public ID for the desired game;
- `hookType`: integer specifying the type of hook being created;
- `hookURL`: URL to be called when POSTing the payload for the event.

Remove Hook By Public ID

Removes a hook for the specified game using its public ID.

Signature

```
kublaiService.removeHook(gameId, hookPublicId, callback);
```

Arguments

- `gameId`: public ID for the desired game;
- `hookPublicId`: Public ID for the Hook. This was returned when creating the Hook.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search