

Backend Unity ↔ BrainCloud

Contrat côté Unity

`BackEndBase` définit le contrat backend utilisé par le reste du client. Les controllers ne doivent pas connaître l'implémentation BrainCloud : ils appellent des opérations métier comme `GetTeamModule`, `SaveModule`, `GenerateResults`, `GetEvaluation`, etc.

`BackEndBrainCloud` implémente ce contrat avec le SDK BrainCloud et les Cloud Scripts du dépôt Node.

```
flowchart LR
    CTRL[Controller ou State Unity] --> BASE[BackEndBase]
    BASE --> BC[BackEndBrainCloud]
    BC -->|commun| C[_command/script]
    BC -->|game-specific| G[gameUid/command/gameUid_script]
    C --> U[_utils]
    G --> GU[utils game-specific]
    GU --> U
```

Convention des Cloud Scripts

Le helper privé `BackEndBrainCloud.SendCommand()` construit le chemin du script :

- commande commune : `_command/<scriptName>`;
- commande game-specific : `<gameUid>/command/<gameUid>_<scriptName>`.

Exemple : `SendCommand(true, "generate_results", ...)` dans Acc1 exécute `acc1/command/acc1_generate_results`.

Authentification et RTT

```
sequenceDiagram
    participant G as Game
    participant B as BackEndBrainCloud
    participant BC as BrainCloud
    G->>B: Login(user, pass, gameId...)
    B->>BC: AuthenticateUniversal
    BC-->>B: profile + session
    B->>B: EnableRTT()
    B-->>G: utilisateur authentifié
```

Après authentification réussie, `EnableRTT()` est appelé. Le client conserve la fonction d'authentification pour pouvoir réagir à une perte de session et réactive RTT lorsqu'une connexion se ferme.

GetDocument()

`GetDocument(entityType, uid, forceCreate)` appelle `_command/get_document`. `GetTeamModule()` l'utilise pour lire `teamsModules` avec l'UID produit par `UtilsBase.GetTeamModuleUid(teamUid, actModuleUid)`.

Gestion des appels

- `SendCommand()` incrémente un compteur et pilote le loading circle.
- Les réponses sont désérialisées vers `MessageData`.
- Les erreurs sont transmises au callback d'échec.
- Une session invalide ou une connexion RTT morte peut déclencher une réauthentification ou une resynchronisation.

Ajouter une commande

1. Créer l'appel métier au bon controller/service.
2. Ajouter le contrat à `BackEndBase` si l'opération doit faire partie de l'abstraction.
3. Implémenter dans `BackEndBrainCloud`.
4. Choisir commande commune ou game-specific.
5. Créer le Cloud Script wrapper.
6. Déplacer la logique réutilisable dans `_utils`.
7. Sauvegarder les documents via `utils_documents`.
8. Envoyer un message RTT si d'autres clients doivent être synchronisés.

Ajouter une commande Unity → BrainCloud de A à Z

Une nouvelle opération métier traverse normalement quatre couches : **controller Unity** → **BackEndBase/BackEndBrainCloud** → **Cloud Script** → **utils/documents**. Si plusieurs clients doivent voir le changement, ajouter ensuite RTT.

1. Définir le contrat Unity

Ajouter la méthode dans `BackEndBase` lorsque l'opération appartient réellement au contrat backend. Utiliser des types simples/`MessageData` compatibles avec les deux backends historiques.

2. Implémenter BackEndBrainCloud

Construire la payload avec les UUIDs minimaux nécessaires et appeler `SendCommand()`.

- `isGameSpecific = false` → `_command/<scriptName>`;
- `isGameSpecific = true` → `<gameUid>/command/<gameUid>_<scriptName>`.

3. Créer un wrapper Cloud Script mince

Le fichier command doit valider/extraire `data`, inclure les utils nécessaires et déléguer le calcul. Une logique de centaines de lignes n'a pas sa place dans le wrapper.

4. Lire les documents avec utils_documents

Utiliser `getDocumentData()` / gather helpers. Une modification doit conserver la version de Custom Entity et passer par `saveDocumentData()` ou l'helper adapté.

5. Sauvegarder avant d'envoyer RTT

Le message temps réel doit représenter un état déjà persisté. Sinon un autre client peut recevoir le message, relire le document et obtenir encore l'ancienne valeur.

6. Choisir la stratégie RTT

- petite payload : envoyer directement les champs nécessaires;
- grosse payload : envoyer une instruction de refresh / `get_data` / `execute_request` selon le système;
- choisir user/team/team+teacher/sim/activity comme audience.

7. Mettre à jour le controller Unity

Le callback de la commande est utile pour l'appelant local, mais les autres clients doivent passer par le listener RTT. Éviter deux chemins de mise à jour qui appliquent deux fois la même mutation.

Exemple de checklist avant merge

1. Contrat `BackEndBase` ajouté si requis.
2. `BackEndBrainCloud` utilise le bon namespace common/game-specific.
3. Wrapper command créé avec le nom exact attendu.
4. Logic métier dans `_utils` ou game-specific utils.
5. Document sauvegardé avec gestion de version.
6. RTT envoyé après la sauvegarde.
7. `MessageType` et listener Unity ajoutés si nécessaire.
8. Chemin testé avec deux clients si la feature est collaborative.
9. Server Script importé vers le bon environnement BrainCloud.

Debug d'une commande

Si Unity reçoit une erreur : logger `scriptName` et payload, exécuter le wrapper dans **Debug local des Server Scripts BrainCloud**, puis inspecter le document avant/après. Si la commande réussit mais l'autre client ne bouge pas, le problème est probablement dans la chaîne RTT plutôt que dans la commande.

Revision #2

Created 2026-09-18 19:09:16 UTC by Mathis

Updated 2026-09-18 20:07:26 UTC by Mathis