

Commandes, documents et persistance BrainCloud

Architecture serveur commune

Le backend du Game repose principalement sur les Cloud Scripts dans

`node.js/backend/braincloud/server_scripts`. Les dossiers `_command`, `_admin_command`, `_job` et `_utils` constituent le socle partagé.

Familles de scripts

Dossier	Rôle
<code>_command/</code>	Entrées appelées par le client : simulations, activités, équipes, évaluations, logbook, chat, rankings, static data, etc.
<code>_admin_command/</code>	Outils administratifs comme création de documents en bulk et récupération de joueurs.
<code>_job/</code>	Travaux planifiés : génération et migrations/versioning de documents.
<code>_utils/</code>	Logique réutilisable : documents, messages, eval, rankings, RNG, By Year et By Module.
<code><gameUid>/command/</code>	Wrappers game-specific.

utils_documents.js

Ce fichier est la couche d'accès commune aux Custom Entities. Ses fonctions structurantes sont

`getDocumentData`, `gatherDocumentsData`, `gatherDocumentsDataWithQuery`, `saveDocumentData`, `saveDocumentDataFields`, `saveDocumentIncrementData`, `createDocumentData`, `deleteDocumentData`, `initializeDocument` et `updateDocumentVersion`.

```
sequenceDiagram
participant CMD as Cloud Script
participant UD as utils_documents
participant CE as Custom Entity
participant VER as version/*
CMD->>UD: getDocumentData(type, uid)
UD->>CE: lecture
CE-->>UD: data + version
UD->>VER: initialize / migrate
UD-->>CMD: objet métier
CMD->>CMD: modifications
CMD->>UD: saveDocumentData(type, uid)
UD->>CE: écriture
```

Versioning et migrations

Les scripts `_utils/version/version_*.js` portent les migrations par type de document. Chaque type peut enregistrer une fonction d'initialisation et une suite de transformations successives.

`version_teams_modules.js` est un bon exemple : il contient plusieurs migrations `teamModuleDataToV*`. Un vieux document peut donc être transformé jusqu'au schéma actuel avant d'être resauvegardé.

Documents principaux

- `simulation` : état/configuration de simulation;
- `activity` : activité en cours;
- `actDataGameSpec` : données propres au jeu pour l'activité;
- `teams` : état commun d'équipe;
- `teamsModules` : données By Module d'une équipe et d'un module;
- `results`, `evaluation`, `evaluationTeam`;
- `teamsLogbooks`, `userDetails`, `simComments`.

Règle lorsqu'un schéma change

1. Définir le nouveau schéma attendu.
2. Ajouter la migration dans le bon `version_<entity>.js`.

3. Garantir le passage de toutes les versions antérieures supportées.
4. Tester lecture, migration et resauvegarde.
5. Ne lancer un job de migration globale qu'après validation sur des documents représentatifs.

Static data serveur

`_command/update_static_data.js` est l'entrée utilisée par les outils d'export pour publier les JSON dans BrainCloud. La chaîne complète est documentée dans **Données statiques et pipeline import/export**.

Diagnostic concurrence

Un bug qui n'apparaît qu'avec plusieurs utilisateurs peut venir d'écritures concurrentes et de versions de Custom Entities. Dans ce cas, suivre le document concerné, sa version lue, la fonction de sauvegarde utilisée et les éventuels retries avant d'accuser l'UI.

Ajouter ou faire évoluer une Custom Entity

Le point critique n'est pas seulement de créer un nouveau JSON : il faut définir son UID, son propriétaire logique, son lifecycle, son versioning et tous les documents qui le référencent.

Nouvelle entity type

1. Choisir un `entityType` stable et un schéma de `data.uid` déterministe.
2. Créer un helper d'initialisation si le document peut être absent.
3. Passer les lectures/écritures par `utils_documents.js`.
4. Ajouter un fichier `version_<entity>.js` si le schéma doit survivre à plusieurs versions du code.
5. Brancher `initializeDocument()` / `updateDocumentVersion()` pour ce type.
6. Ajouter le document aux scripts de copie/migration si une simulation copiée doit l'emporter.

Faire évoluer une entity existante

Ne pas écrire du code qui suppose simplement que le nouveau champ existe. Ajouter une migration `Vn → Vn+1` qui :

- tolère les documents anciens;
- initialise les nouveaux champs avec une valeur métier correcte;
- ne détruit pas les données inconnues;
- peut être appliquée dans l'ordre après toutes les migrations précédentes.

Concurrence et version BrainCloud

Une Custom Entity possède une version serveur. Une écriture issue d'une lecture ancienne peut être rejetée. Lorsqu'un bug n'apparaît qu'avec deux utilisateurs, logger au minimum : `entityType`, `uid`, version lue, version sauvegardée et commande à l'origine de l'écriture.

Copie entre environnements

`exports/data_importer_braincloud.js` copie plusieurs documents liés à une simulation : simulation, activity, results, actDataGameSpec, teams, teamsLogbooks, evaluationTeam, teamsModules, userDetails et evaluation. **Si un nouveau type de document fait partie d'une simulation copiée, ce script doit être audité et potentiellement étendu.**

Checklist migration

1. Créer un vieux document représentatif.
2. Le charger avec le code actuel.
3. Vérifier chaque transformation de version.
4. Resauvegarder puis recharger.
5. Tester une simulation active avec plusieurs clients.
6. Tester le job global de migration seulement après validation ciblée.
7. Tester la copie Dev → autre environnement si le document doit suivre une simulation.

Revision #3

Created 2026-09-18 19:09:33 UTC by Mathis

Updated 2026-09-18 20:45:46 UTC by Mathis