

Logbook — sauvegarde, historique et évaluation

Structure des données

Le logbook est stocké par équipe dans `teamsLogbooks`, avec `teamUid` comme UID du document. Chaque activité possède deux rubriques : `student` et `teacher`. Chaque rubrique contient une entrée courante par `yearId` et un historique de versions.

```
logbook[actUid][role][yearId] = { text, date }  
logbookHistory[actUid][role][yearId] = [{ text, date }, ...]
```

Côté Unity, `Logbook.logbookRoles` indexe les activités et les rôles. `LogbookRole.entries` contient les entrées courantes; `historyEntries` contient les listes de versions. `LogEntry` porte le texte et une date en millisecondes depuis l'époque, convertie en heure locale pour l'affichage.

Chargement et cache

`TeamBaseController.Logbook` expose le modèle chargé. `LoadLogbook()` appelle `BackEndBrainCloud.GetLogbook()`, puis `_command/get_logbook.js`, qui retourne le document `teamsLogbooks`.

Le controller conserve un cache. Des demandes concurrentes de chargement sont regroupées par callbacks et `_logbookIsLoading`. Si le cache existe, le callback est appelé sans relire le serveur.

Édition et sauvegarde

`LogbookRoleControl` gère l'édition, la sauvegarde, l'annulation, le collage, Undo/Redo et la sélection d'une version. Le bouton Save applique `Trim()` au texte et évite l'envoi si le texte est identique à l'entrée actuelle.

L'accès étudiant à l'édition dépend du rôle : avec `HasLogbookDir`, le directeur du logbook est requis; sinon, en équipe, le rôle temporaire `logbook_editor` sert à coordonner l'édition. Il peut être repris après confirmation et est libéré à la sortie du mode édition.

1. `TeamBaseController.SaveLogbookEntry()` normalise `actUid` en asynchrone via `UtilsBase.GetActUid()`.
2. Le rôle envoyé au serveur est déterminé par l'utilisateur : professeur → `teacher`, étudiant → `student`.
3. `BackEndBrainCloud.SaveLogbook()` envoie `teamUid`, `actUid`, `yearId`, `date`, `text` et `role` à `_command/save_logbook`.
4. Le controller met aussi à jour son entrée locale immédiatement. Cette mise à jour locale ne constitue pas un accusé de sauvegarde serveur.
5. Le script persiste les champs de `teamsLogbooks`, puis envoie `logbook_changed` à l'équipe et au professeur.

Historique : 10 versions précédentes

`save_logbook.js` copie l'entrée précédente dans l'historique si son texte ou sa date diffère de la nouvelle entrée. La version précédente est insérée en tête. `MAX_LOGBOOK_HISTORY = 10` limite la liste; la version la plus ancienne est retirée au-delà de cette limite. La première sauvegarde n'a pas d'entrée précédente à archiver.

Les piles Undo/Redo du champ d'édition sont locales et distinctes de cet historique persistant.

Restauration

La liste déroulante présente un choix initial, puis les versions historiques. Sélectionner une version ouvre `HistoryPreviewPopupControl`; la confirmation appelle `RestoreLogbookHistory()` avec l'index de version, après normalisation de l'activité en asynchrone.

`restore_logbook_history.js` retire la version choisie de l'historique, place l'entrée courante en tête, puis remplace l'entrée courante par le texte et la date d'origine de la version restaurée. Il sauvegarde le document et émet `logbook_changed`.

Synchronisation entre clients

Le message contient `teamUid`, `actUid`, `yearId` et `role`. Le texte complet et l'historique sont relus depuis le serveur.

`TeamBaseController.OnLogbookChanged()` ignore les autres équipes et les logbooks qui ne sont pas encore chargés. Pour un logbook chargé de la bonne équipe, il invalide le cache, appelle `LoadLogbook()`, puis déclenche `onLogbookChanged(role)`. Le chargement initial reste donc nécessaire.

Évaluation distincte du texte

`save_logbook_eval.js` enregistre la note dans le document `evaluation` de chaque étudiant concerné. `simulation.logbookPonderationType` choisit un mode annuel ou global, en équipe ou individuel. En équipe, la note est appliquée aux membres; en individuel, elle vise `studentUid`.

- Annuel : `evaluation.logbookEval[actUid][yearId]`.
- Global : `evaluation.logbookEval[actUid]`.
- La valeur `note = -1` supprime la note concernée.

`getLogbookScore()`, dans `utils_eval.js`, calcule en annuel la moyenne arrondie des notes présentes jusqu'à `currentYear`. Sans note, il retourne `0` si l'année est terminée, sinon `-1`. En global, il retourne la note stockée.

La sauvegarde des notes émet `logbook_eval_changed_stub` et, pour les notes annuelles, `logbook_eval_by_year_changed`. `get_logbook_evaluation.js` charge les notes annuelles d'un étudiant. Sauvegarder le texte ne sauvegarde pas sa note.

Réouverture annuelle

`utils_reopen_year.js` appelle `deleteYearLogbook(actData, teamUid)` après avoir ajusté l'état et, selon le cas, l'année courante. Le helper cible précisément `yearId = actData.currentYear + 1`.

Il supprime les notes de cette année pour les pondérations annuelles, puis les entrées courantes et leurs historiques pour les rôles concernés. Il sauvegarde les documents et émet les messages de changement du logbook et d'évaluation. Les notes globales ne sont pas supprimées par ce traitement annuel.

Fichiers de référence

Dans `master/game/Assets/Src` : `Game/Data/Logbook.cs`, `Game/Controllers/TeamBaseController.cs`, `Game/BackEnd/BackEndBrainCloud.cs`, `UI/SimByYear/LogbookRoleControl.cs`.

Dans `master/node.js/backend/braincloud/server_scripts` : `_command/get_logbook.js`, `save_logbook.js`, `restore_logbook_history.js`, `save_logbook_eval.js`, `get_logbook_evaluation.js`, `_utils/utils_eval.js`, `utils_by_year.js` et `_utils/command/utils_reopen_year.js`.

Vérifié le 30 septembre 2026 dans le code Game et BrainCloud de la branche `refactor_notifications`. Les fichiers de logbook et d'évaluation décrits ici sont identiques à ceux de `master` pour les chemins inspectés. Les parcours décrits ont été contrôlés par lecture du code; aucune sauvegarde ou réouverture de simulation réelle n'a été exécutée pour cette vérification.

Revision #5

Created 2026-09-30 14:53:23 UTC by Mathis

Updated 2026-09-30 15:23:44 UTC by Mathis