

Description

EasyDB

Extension Arma 3 (Go/cgo, `.so` Linux) faisant le pont entre SQF et MySQL 8 pour **FastRP**. Une seule commande générique de requête paramétrée, en synchrone ou en asynchrone — pas de protocole multi-fichiers façon extDB3.

EasyDB est livré comme un **addon Arma complet, server-side only** : `@EasyDB/EasyDB_x64.so` (l'extension) + `@EasyDB/addons/easydb.pbo` (les fonctions `EasyDB_fnc_*`, auto-compilées via `CfgFunctions`, aucun script à coller à la main dans une mission).

Prérequis

- Go 1.21+ avec `CGO_ENABLED=1` (nécessite gcc)
- [armake2](https://github.com/KoffeinFlummi/armake2) pour empaqueter le PBO (`cargo install --git https://github.com/KoffeinFlummi/armake2 armake2`) — outil communautaire Linux, les outils BI officiels (Addon Builder) sont Windows-only
- MySQL 8.x
- Linux x86-64 (aucun support Windows prévu)

Build & packaging

```
make package # -> dist/@EasyDB/EasyDB_x64.so + dist/@EasyDB/addons/easydb.pbo
```

(`make build` seul ne produit que le `.so`, dans `build/`, si vous voulez juste tester l'extension sans passer par le PBO.)

Déploiement : copier `dist/@EasyDB/` entier sur le serveur, à côté des autres mods, et charger `-mod=@EasyDB` **uniquement sur le serveur dédié** (l'addon ne fait rien côté client — `EasyDB_fnc_init` se coupe tout seul si `!isServer`). Résultat final, un mod autonome :

```
@EasyDB/  
├─ EasyDB_x64.so  
├─ easydb.json      (voir easydb.json.example – pas commité, contient un mot de passe)  
└─ addons/  
    └─ easydb.pbo
```

`easydb.json` se pose directement à la racine du mod, à côté du `.so` : l'extension retrouve son propre dossier au chargement (via `dlopen` sur son propre symbole) et y cherche `easydb.json` en priorité — pas besoin de variable d'environnement ni de dépendre du répertoire de travail du serveur. `EASYDB_CONFIG` reste disponible pour pointer ailleurs si besoin, et `./easydb.json` (répertoire de travail) sert de dernier recours.

Important : `EasyDB_x64.so` doit rester à la racine du mod (`@EasyDB/`), jamais dans le PBO — Arma ne charge jamais une extension depuis l'intérieur d'un PBO. Le nom du fichier est sensible à la casse sur Linux et doit correspondre exactement à `"EasyDB" callExtension ...` utilisé dans le code.

API SQF

Une fois `-mod=@EasyDB` chargé, `EasyDB_fnc_init` s'auto-exécute (postInit) et enregistre le handler `ExtensionCallback` — rien à faire côté mission. Deux fonctions, `EasyDB_fnc_query` (synchrone) et `EasyDB_fnc_queryAsync` (asynchrone, avec callback).

Guide complet avec exemples FastRP (argent, grade, inventaire JSON, gestion d'erreurs, bonnes pratiques) : [docs/USAGE.md](#).

Tests

```
make test    # tests unitaires (sqfarray, jsonconv, chunk, db via sqlmock, dispatch, async)  
make vet
```

Test de bout en bout sans Arma (dlopen le `.so` compilé et appelle directement `RVExtension/RVExtensionArgs/RVExtensionRegisterCallback`) :

```
cp easydb.json.example easydb.json    # renseigner un vrai mot de passe  
make smoke    # package + dlopen dist/@EasyDB/EasyDB_x64.so + exercice complet de l'API
```

`docker/init.sql` crée le schéma `easydb` (utilisateur dédié + table d'exemple `example_players`) à jouer contre un serveur MySQL 8 existant — utile pour `make smoke` et pour des tests d'intégration futurs.

Schéma FastRP

Pas encore figé. `example_players` dans `docker/init.sql` n'est qu'une table de démonstration pour valider get/set/increment + colonne JSON ; le schéma réel de FastRP (joueurs/factions/jobs/économie) viendra dans un second temps et n'a besoin d'aucun changement côté extension : c'est justement le but d'une commande de requête générique.

Revision #2

Created 2026-09-29 07:07:00 UTC by ProtoMehka

Updated 2026-09-29 07:18:45 UTC by ProtoMehka