# Présentation

# Présentation Générale

La bibliothèque **M5lib** vise à uniformiser la gestion d’une flotte de M5 Core2 au sein d’un FabLab, facilitant le déploiement, l’évolution et la maintenance. Sa structure repose sur des concepts avancés de programmation orientée objet : **factory design pattern** et **polymorphisme**.

# Factory Design Pattern dans M5lib

## Structure de Base

- La classe **M5lib** est la classe de base commune à tous les types d’appareils M5 (accueil, servante, machine, ordinateur, matériel), définissant l’interface et le comportement standard minimal.
- Plusieurs classes dérivées spécialisent ce comportement pour chaque type de M5 : `accueil`, `servante`, `machine`, `ordinateur`, `materiel`, chacune héritant de M5lib.

## Objectif du Pattern

Le **factory design pattern** permet d’instancier dynamiquement le bon type d’objet dérivé (`accueil`, `servante`, etc.) en fonction du contexte (par exemple, d’un paramètre de configuration ou d’une saisie utilisateur), tout en manipulant leur interface commune à travers des pointeurs ou références de type M5lib. Dans ce cas, cela veut dire que la déclaration du type d'objet dérivé dans le firmware du M5 permet automatiquement et de manière transparente d'adapter toutes les fonctions nécessaires au comportement de ce type de M5 (requêtes d'API, type d'emprunt...).

## Avantages

- Centralisation de la logique d’instanciation : éviter la duplication de code lors de la création de nouveaux modules.
- **Extensibilité et maintenance** : ajouter un nouveau “type” d’objet devient trivial (ajout d’une seule classe dérivée et adaptation de la factory).

# Polymorphisme dans M5lib

## Implémentation

- La classe `M5lib` déclare des méthodes virtuelles pures (`virtual ... = 0;`) comme `uploadlog` et `changestatus`. Ces fonctions ne sont alors pas appelables dans M5lib et n'existent pas en tant que tel tant que le type de M5 n'a pas été déterminé.
- Chaque classe dérivée (*accueil*, *servante*, etc.) implémente ces méthodes selon les besoins spécifiques de son rôle (par exemple, signature différente, formats d’upload différents…). Ainsi, pour un même appel de fonction, le comportement s'adapte parfaitement au cas d'utilisation du type de M5 concerné.

## Usage

- Grâce au polymorphisme, le reste du programme peut manipuler les objets à travers des pointeurs de type M5lib et appeler, par exemple, `uploadlog` sans connaître la classe exacte : c’est la bonne implémentation qui sera automatiquement utilisée à l’exécution.
- Cela permet d’écrire du code générique, réutilisable, et facile à maintenir. De plus, si par la suite un nouveau cas d'utilisation est identifié comme nécessitant la création d'un nouveau type de M5, il ne faudra ajouter dans le code que les définitions de comportement spécifique à ce type, rendant l'évolutivité beaucoup plus simple.

# Limites et inconvénients

Cette flexibilité du code ne vient pas sans compromis :

- Au cours du développement, il est apparu que la bibliothèque devenait trop grosse pour être déployée sur des M5Stack Core, et afin résoudre ces limitations matérielles, la bibliothèque a été optimisée pour fonctionner sur des M5Stack Core2 (qui possèdent plus de RAM et de stockage interne).
- La logique de polymorphisme et de type abstrait implique que chaque appareil embarque toutes les méthodes de toutes les classes, y compris de celles dont il n'a pas besoin. Cela ne pose pour le moment pas de problème, mais si la bibliothèque est amenée à grandir encore plus, une optimisation de ce côté pourrait s'avérer nécessaire. Son implémentation n'est pas compliquer : il suffit de rajouter des macros avant la définition de chaque classe afin de n'inclure que le code nécessaire, mais cela implique de définir le type de M5 à la compilation, rendant le déploiement plus technique et le code moins flexible.

---

Revision #2
Created 17 July 2025 12:27:32 by Eyglier De Rooster Mathieu
Updated 21 July 2025 12:39:02 by Eyglier De Rooster Mathieu