Table des matières

1. Intégrer le dépôt µcBlockly dans un autre projet 3
1.1. 1. État actuel de µcBlockly 3
1.2. 2. Oui, c’est possible : trois approches 3
1.3. 3. Synthèse 5
1.4. 4. API noyau ''createUcBlockly'' 5
1.5. 5. Publication npm (P3) 6

1. Intégrer le dépôt µcBlockly dans un autre projet

Ce document décrit **comment intégrer le dépôt µcBlockly** (tel quel, sans tout dupliquer) dans un autre projet qui s'appuierait dessus — par exemple une application qui embarque l'éditeur Blockly Arduino dans une page ou un onglet.

—

1.1. 1. État actuel de µcBlockly

- **Noyau embeddable** : `src/core/` + API `createUcBlockly()` — voir KERNEL.md.
- **Démo autonome** : `src/index.ts` + Webpack → `dist/bundle/` (HTML + Monaco + shell).
- **Embed minimal** : `npm run build:core:bundle` → `dist/core/` (sans Monaco).
- **Exports npm** : "`.`" / "`./core`" → noyau ; "`./host`" → aides produit ; "`./demo`" → entrée démo.
- **"`private`": `true`** : pas encore publié sur npm (P3).

La démo complète reste couplée au DOM (`requiredElements` dans `src/index.ts`). Les intégrations embeddables doivent utiliser **`µcBlockly/core`** uniquement.

—

1.2. 2. Oui, c'est possible : trois approches

1.2.1. 2.1 Intégration par iframe (recommandé si vous voulez peu toucher à µcBlockly)

Idée : votre projet contient µcBlockly (submodule ou copie), le build une fois, et affiche le résultat dans un **iframe**. Votre app est la « coque » (menu, auth, autre contenu) ; µcBlockly tourne dans l'iframe comme une app à part.

Mise en œuvre :

1. **Ajouter µcBlockly en submodule Git** (depuis votre dépôt) :

```
git submodule add https://github.com/A-S-T-U-C-E/ucBlockly.git ucBlockly
```

```
Ou cloner le dépôt dans un sous-dossier de votre projet.
```

1. **Build** : dans le dossier µcBlockly, exécuter `npm ci` puis `npm run build` (ou `build:local` si vous ouvrez en local). Le résultat est dans `ucBlockly/dist/bundle/` (ou `build/` en dev). Pour un embed sans Monaco : `npm run build:core:bundle` → `dist/core/`.

1. **Servir le bundle** : votre projet sert le contenu de `ucBlockly/dist/bundle/` (ou `dist/core/`) sous une URL (ex. `/blockly/` ou un sous-répertoire statique).

1. **Intégration dans votre page** :

```
<iframe
  src="/blockly/"
  title="µcBlockly"
  style="width:100%; height:800px; border:0;"
></iframe>
```

1. **Communication (optionnel)** : si vous devez échanger des données (charger/sauvegarder un XML ou du JSON) entre la page parente et µcBlockly, utiliser `postMessage` entre la fenêtre parente et `iframe.contentWindow`.

Avantages : pas de modification du code µcBlockly, mises à jour du submodule + rebuild.
Inconvénients : pas d'API directe (tout passe par l'iframe et éventuellement postMessage), et vous devez gérer le chemin du build (CI, déploiement).

—

1.2.2. 2.2 Intégration par sous-dossier + build dans le pipeline du projet parent

Idée : comme en 2.1, µcBlockly est un sous-dossier (submodule ou clone). Votre outil de build (Webpack, Vite, etc.) ou votre script de déploiement **lance le build de µcBlockly** puis copie les artefacts où il faut.

Mise en œuvre :

1. Submodule ou copie du dépôt, par ex. dans `vendor/ucBlockly` ou `apps/blockly-editor`.
2. Dans le script de build ou la CI du **projet parent** :

```
cd vendor/ucBlockly && npm ci && npm run build && cd ../..
```

1. Copier `vendor/ucBlockly/dist/bundle/*` vers le répertoire statique de votre app (ex. `public/blockly/`).
2. Dans votre app, ouvrir cette URL dans une iframe (comme en 2.1) ou rediriger une route vers `index.html` du bundle.

Avantages : un seul dépôt / une seule CI, intégration claire « µcBlockly = sous-projet buildé ».
Inconvénients : même contraintes qu'en 2.1 (interface = iframe ou page entière).

—

1.2.3. 2.3 Intégration dans la page via le noyau (''createUcBlockly'') — recommandé sans iframe

Idée : submodule ou dépendance locale, build du noyau, puis injection dans un conteneur DOM de votre app.

```
import { createUcBlockly, basic_toolbox } from "µcBlockly/core";
// depuis les sources : "./ucBlockly/build/src/core/index.js"
```

```
const editor = createUcBlockly(document.getElementById("editor")!, {
  toolbox: basic_toolbox,
  language: "fr",
  media: "./media/", // servir aussi dist/core/media/ ou les media Blockly
  enforcers: { strictTypes: true, programStructure: true },
  plugins: { minimap: true, workspaceSearch: true },
  onCodeChange: (code) => {
    document.getElementById("code")!.textContent = code;
  },
});

// editor.getCode(), editor.saveWorkspace(), editor.loadWorkspace(state),
// editor.resize(), editor.dispose()
```

Prérequis : `npm run build:core` (et éventuellement `build:core:bundle` pour un artefact navigateur sous `dist/core/`). Voir KERNEL.md.

Pour les aides **produit** (thèmes coque, a11y, Monaco) : `µcBlockly/host` — pas le noyau.

Variante fragile (démo complète) : importer `src/index.ts` / `µcBlockly/demo` en reproduisant le DOM de `public/index.html` (`requiredElements`). Réservé aux cas où vous voulez la coque démo entière, pas seulement l'éditeur.

Avantages : un seul document, API propre, sans Monaco dans le noyau. **Inconvénients** : le bundler parent doit résoudre Blockly et les plugins `@blockly/*` (ou consommer `dist/core/`).

—

1.3. 3. Synthèse

Besoin	Approche recommandée
Intégrer µcBlockly « tel quel » avec le moins de changements	**2.1 Iframe** (submodule + `build` / `build:core:bundle`)
Automatiser le build dans un gros projet	**2.2** (submodule + CI parent, puis iframe ou route)
Intégrer dans la page avec une API propre	**2.3** — `createUcBlockly()` via `µcBlockly/core`
Reprendre la coque démo complète (Monaco, topbar)	Variante démo de **2.3** (DOM `public/index.html`)

—

1.4. 4. API noyau ''createUcBlockly''

Point d'entrée : `src/core/create_ucblockly.ts` → `build/src/core/` / exports "`.`" et "`./core`". Détail des options : KERNEL.md.

- Pas de `getElementById` imposé pour le workspace (seul un conteneur est requis).
- La démo (`src/index.ts` + shell) reste l'exemple produit complet.

1.5. 5. Publication npm (P3)

Déjà en place :

1. Point d'entrée lib : `createUcBlockly()` (`src/core/`).
2. Build noyau : `npm run build:core` + `build:core:bundle` → `dist/core/` (sans Monaco).

Reste à faire pour `npm install` public : retirer "`private": true` et publier sur npm.

```
import { createUcBlockly, basic_toolbox } from "µcBlockly";
// ou "µcBlockly/core"
```

—

En résumé : **oui**, intégration possible. La plus simple reste l'**iframe** ; pour une intégration dans la page, utilisez **`createUcBlockly`** (KERNEL.md).

From:
https://wiki.libreduc.cc/ - **LibrEduc**

Permanent link:
https://wiki.libreduc.cc/fr:arduino:ucblockly:integration

Last update: **2026/08/11 00:18**