Table des matières

1. Analyse de la sérialisation Blockly — Comparaison avec les meilleures pratiques

1.1. État actuel de l'implémentation

Toute la persistance workspace passe par **`UcBlocklyWorkspaceManager`** (`src/core/ucblockly_workspace_manager.ts`), avec l'aide de `sanitizeSerializedWorkspaceState` (`src/core/workspace_state.ts`). Le shell démo (`demo/blockly_application_shell.ts`) et le re-export `blockly_application_type.ts` délèguent à ce manager.

1.1.1. Session storage (''workspaceSaveBlocks'' / ''workspaceLoadBlocks'')

```
public workspaceSaveBlocks = (storageKeyWorkspaceBlocks: string): void => {
  const data = Blockly.serialization.workspaces.save(this.workspace);
  this.sanitizeSerializedWorkspaceState(data);
  window.sessionStorage?.setItem(storageKeyWorkspaceBlocks,
JSON.stringify(data));
};

public workspaceLoadBlocks = (storageKeyWorkspaceBlocks: string): void => {
  const data = window.sessionStorage?.getItem(storageKeyWorkspaceBlocks);
  if (!data) return;
  try {
    const parsed = JSON.parse(data);
    if (typeof parsed !== 'object' || parsed === null) {
      console.warn('Invalid workspace data format, skipping load');
      return;
    }
    this.sanitizeSerializedWorkspaceState(parsed);
    // clear + load + normalizeVariableTypes + refreshBoardPinDropdowns
  } catch (error) {
    console.error('Failed to load workspace blocks:', error);
    window.sessionStorage?.removeItem(storageKeyWorkspaceBlocks);
  }
};
```

1.1.2. Fichiers (''workspaceSaveToFile'' / ''workspaceLoadFromFile'')

- Export JSON téléchargeable via l'API officielle Blockly.
- Import avec validation, gestion d'erreurs utilisateur (`alert`), mode **fusion** (`mergeBlocksFromState`) ou remplacement complet.
- `sanitizeSerializedWorkspaceState` normalise l'état avant chargement.

1.1.3. Points positifs

1. **API JSON officielle** : `Blockly.serialization.workspaces.save()` / `load()`
2. **Sauvegarde automatique** : debounce (`scheduleSave()`) sur les changements du workspace
3. **Sauvegarde à la fermeture** : `onbeforeunload`
4. **Charge au démarrage** : restauration depuis sessionStorage
5. **Robustesse** : `try/catch`, validation objet, suppression des données corrompues
6. **Post-traitement** : `normalizeVariableTypes()`, `refreshBoardPinDropdowns()`
7. **Ancien `serialization.ts`** : fichier supprimé ; plus de code parallèle obsolète

1.2. Comparaison avec les meilleures pratiques Blockly

Selon la documentation officielle Blockly :

Critère	Statut
Format JSON (XML legacy évité)	
API `Blockly.serialization.workspaces`	
État complet (blocs, variables, plugins)	
Timing (changements + déchargement page)	
Gestion d'erreurs au chargement	(session + fichier)

1.3. Améliorations encore possibles (optionnel)

1.3.1. Version explicite du format exporté

Pour faciliter les migrations futures, on pourrait encapsuler :

```
{ version: 1, data: Blockly.serialization.workspaces.save(workspace) }
```

Non implémenté aujourd'hui : le JSON reste le format Blockly natif, compatible avec les outils tiers (dont `convert-rduino`).

1.3.2. Validation typée plus stricte

Une garde `isValidWorkspaceData(data)` (présence de `blocks` ou `variables`) pourrait compléter la validation actuelle `typeof object`.

1.4. Conversion Blockly@rduino

Les projets XML Blockly@rduino ne passent pas par cette sérialisation : utiliser **`npm run convert-rduino`** → voir CONVERT_RDUINO.md.

1.5. Conclusion

L'implémentation actuelle est **alignée sur les recommandations Blockly** et inclut déjà la plupart des améliorations suggérées dans les versions antérieures de ce document (erreurs, validation basique, suppression de `serialization.ts`).

—

1.6. Voir aussi

- ARCHITECTURE.md — flux de démarrage et persistance du workspace ;
- CONVERT_RDUINO.md — import de sauvegardes Blockly@rduino ;
- INTEGRATION_DEPOT.md — intégration du dépôt dans un autre projet.