STM32 Package Manager for VS Code

Technical Report and Implementation Details

Project Documentation

August 11, 2025

Contents

1 Overview

This VS Code extension provides a lightweight workflow to browse STM32Cube firmware packages, select a board, view templates/projects, and import a selected project into a user-chosen

location. On import, the extension automatically bundles the package's `Drivers` folder into the imported project to ensure build readiness.

2 Key Features

- Automatic discovery of boards under `Projects/` in STM32Cube packages.
- Template/project enumeration across categories (e.g., `Examples`, `Examples_LL`, `Templates`, `Applications`, `Demonstrations`).
- Webview UI to pick package, board, template, target location, and project name.
- Import pipeline that copies the selected template and always includes `Drivers/`.
- Robust detection of toolchains (GCC/Keil/IAR/STM32CubeIDE) and typical project roots.

3 Technology Stack

- **VS Code API**: commands, webview, tree provider, dialogs.
- **TypeScript**: extension logic and type safety.
- **Node.js `fs-extra`**: filesystem traversal and copying.
- **`xml2js`**: optional parsing of `package.xml`/`.pdsc`.
- **HTML/CSS/JS**: webview UI (front-end) with message passing to the extension host.

4 Repository Structure (Core Files)

- `package.json`: metadata, commands, views, activation config, deps.
- `src/extension.ts`: activation; registers commands and tree view.
- `src/packageImportPanel.ts`: webview lifecycle and message routing.
- `src/packageManager.ts`: backend logic for discovery and import.
- `media/main.js`, `media/main.css`: webview UI scripts and styles.

5 Contributions and Commands

Commands

- `stm32PackageManager.importPackage`
- `stm32PackageManager.openPackageManager`

Views

- Activity bar container: `stm32-package-manager`
- Tree view: `stm32PackageExplorer`

6 Runtime Architecture

6.1 High-Level Flow

1. User opens the Import UI (webview).
2. Selects STM32Cube package root; backend analyzes package and loads boards.
3. Selects board; backend enumerates templates/projects.
4. Selects template; UI suggests project name.
5. Clicks "Browse" to pick target directory; extension opens native folder dialog.
6. Clicks "Import"; backend copies project and `Drivers` to target.

6.2 Message Protocol (Webview ↔ Extension)

- Webview → Extension:
  - `selectPackage`, `selectBoard`, `selectProject`
  - `browseLocation`
  - `importProject` (with `packagePath`, `boardId`, `projectName`, `projectPath`, `location`, `targetName`)
- Extension → Webview:
  - `packageSelected`, `boardsLoaded`, `projectsLoaded`
  - `locationSelected`
  - `importComplete`

7 Backend: `PackageManager`

7.1 Package Analysis

If `package.xml`/`.pdsc` exists, it's parsed for basic info. Otherwise, metadata falls back to directory naming.

7.2 Board Discovery (Projects Root)

Boards are recognized as first-level directories under `Projects/`. Each directory is analyzed and accepted as a board if it contains at least one valid project category with projects.

Listing 1: Scanning boards under Projects/

```
const projectsPath = path.join(packagePath, 'Projects');
const entries = await fs.readdir(projectsPath, { withFileTypes: true });
console.log(`Entries in boards path (${projectsPath}):`,
           entries.map(e => e.name));
for (const entry of entries) {
  if (!entry.isDirectory()) continue;
  const boardDir = path.join(projectsPath, entry.name);
  const board = await this.analyzeBoardDirectory(boardDir, entry.name);
  if (board) boards.push(board);
}
```

7.3 Category Detection

We support canonical and variant names using a regex:

Listing 2: Accepted project categories

```
/^(Examples(?!\w)|Examples_[A-Za-z0-9_]+|
  Templates(?!\w)|Templates_[A-Za-z0-9_]+|
  Applications|Demonstrations)$/i
```

7.4 Project Discovery

Projects are discovered recursively via $\leq$ 2–3 directory levels using structural and file heuristics:

- Toolchain directories: `stm32cubeide`, `mdk-arm`, `ewarm`, `sw4stm32`
- File indicators: `.ioc`, `Makefile`, `.uvprojx`, `.eww`, `.ewp`, `.cproject`
- Source layout: `Core/`, or `Inc/`+`Src/`

Listing 3: Project root heuristics

```
private async looksLikeProjectRoot(dir: string): Promise<boolean> {
  const entries = await fs.readdir(dir, { withFileTypes: true });
  const names = entries.map(e => e.name.toLowerCase());

  const toolchainDirs = ['stm32cubeide','mdk-arm','ewarm','sw4stm32','iar'];
  if (entries.some(e => e.isDirectory() && toolchainDirs.includes(e.name.toLowerCase()
      ))) return true;

  const fileIndicators = ['uvprojx','eww','ewp','cproject','project','ioc','makefile
      '];
  if (entries.some(e => !e.isDirectory() && fileIndicators.some(ext => e.name.
      toLowerCase().endsWith(ext)))) return true;

  const hasCore = names.includes('core');
  const hasInc = names.includes('inc');
  const hasSrc = names.includes('src');
  return hasCore || (hasInc && hasSrc);
}
```

7.5 Toolchain Detection

Listing 4: Toolchain detection

```
private async detectToolchains(projectDir: string): Promise<string[]> {
  const entries = await fs.readdir(projectDir, { withFileTypes: true });
  const names = entries.map(e => e.name.toLowerCase());
  const t: string[] = [];
  const addIf = (c: boolean, n: string) => { if (c) t.push(n); };

  addIf(names.includes('stm32cubeide') || await this.hasMatchingFile(projectDir, '*.
      cproject'), 'STM32CubeIDE');
  addIf(names.includes('mdk-arm') || await this.hasMatchingFile(projectDir, '*.uvprojx
      '), 'Keil');
  addIf(names.includes('ewarm') || await this.hasMatchingFile(projectDir, '*.eww') ||
      await this.hasMatchingFile(projectDir, '*.ewp'), 'IAR');
  addIf(await this.hasMatchingFile(projectDir, 'Makefile'), 'GCC');
```

```
  return Array.from(new Set(t));
}
```

7.6 Import Pipeline

The import method supports both legacy and extended signatures. It uses the user-chosen location, derives a filesystem-safe target name, copies the project, and then always copies `Drivers/` from the package root into the target.

Listing 5: Import with Drivers inclusion

```
async importProject(pkg: string, boardId: string, projName: string,
                    location?: string, targetName?: string, projectPath?: string):
                        Promise<string> {
  const targetBase = await this.resolveTargetBase(location);
  const finalName = this.deriveSafeName(projName, targetName);
  const target = path.join(targetBase, finalName);

  // Ensure only parent exists; avoid double nesting.
  await fs.ensureDir(targetBase);

  const src = projectPath ?? await this.findProjectSource(pkg, boardId, projName);
  if (!src) throw new Error(`Project source not found for "${projName}"`);

  await fs.copy(src, target, { overwrite: true, errorOnExist: false });

  // Always copy Drivers
  const driversSrc = path.join(pkg, 'Drivers');
  const driversDst = path.join(target, 'Drivers');
  if (await fs.pathExists(driversSrc)) {
    await fs.copy(driversSrc, driversDst, { overwrite: true, errorOnExist: false });
  }

  await this.updateProjectConfiguration(target, finalName);
  return target;
}
```

8 Frontend: Webview UI

8.1 Template Selection and Name Suggestion

When a template is selected, the UI proposes a default project name based on the last segment of the template's filesystem path.

Listing 6: Name auto-suggestion

```
templateSelect.addEventListener('change', (e) => {
  const chosen = currentProjects.find(p => p.name === e.target.value);
  if (!chosen) return;
  const suggested =
    (chosen.path ? chosen.path.split(/[\\/]/).pop() : (chosen.name||'').split('/').pop
        ());
  if (!projectName.value || projectName.value === chosen.name) {
    projectName.value = suggested || projectName.value;
  }
  updateImportButton();
});
```

8.2 Location Picker (Native Folder Dialog)

The webview asks the extension host to open a native folder picker; the chosen path is returned and used to enable the Import action.

Listing 7: Webview message + folder selection

```
browseLocationBtn.addEventListener('click', () => {
  vscode.postMessage({ command: 'browseLocation' });
});

// In panel (extension host)
private async _browseLocation() {
  const folderUri = await vscode.window.showOpenDialog({
    canSelectMany: false, canSelectFiles: false, canSelectFolders: true,
    openLabel: 'Select Import Location'
  });
  const p = folderUri && folderUri[0] ? folderUri[0].fsPath : '';
  this._panel.webview.postMessage({ command: 'locationSelected', path: p });
}
```

8.3 Import Invocation

The webview sends `projectPath`, `location`, and `targetName` so the backend performs a deterministic copy.

Listing 8: Import message payload

```
vscode.postMessage({
  command: 'importProject',
  packagePath: currentPackageInfo.path,
  boardId: selectedBoard.id,
  projectName: selectedProject.name,
  projectPath: selectedProject.path,
  targetName: projectNameInput.value.trim(),
  location: locationInput.value,
  openReadme: openReadmeCheckbox.checked
});
```

9 Important Implementation Notes ("Astuce")

- **Projects root is `Projects/`** (not `Applications/`) for STM32Cube U5 packages.
- **Category regex** supports `Examples_LL`, `Templates_LL`, etc., not just canonical names.
- **Recursive discovery** avoids hardcoding depth assumptions; instead uses robust project-root heuristics.
- **Drivers inclusion**: always copied into the imported project to ensure headers/sources are available.
- **Avoid double nesting**: ensure only the parent directory is created; copy the project directly into `<Location>/<Name>`.
- **Type safety fixes**: installed `@types/fs-extra`, `@types/xml2js`; added explicit typings to callbacks.
- **Logging**: extensive `console.log` in discovery/import paths to aid troubleshooting.

10 Build & Run

1. `npm install`
2. `npm run compile`
3. Press `F5` in VS Code to launch the Extension Development Host.
4. Open Command Palette and run "STM32: Import STM32 Package".

11 Testing & Debugging Tips

- Use **Help → Toggle Developer Tools** for webview console logs.
- Use **Output → Log (Extension Host)** for backend logs.
- Confirm logs for:
  - "Checking Projects directory" and directory entries
  - Category detection per board
  - "Discovered X projects in <Category>"
  - Import: source path, target path, Drivers copy confirmation

12 Possible Enhancements

- Optional inclusion of `Middlewares/`.
- Board image rendering via local file URIs.
- CMake or VS Code tasks generation per toolchain.
- Persist recently-used packages and locations.