StatConverter: Auto-Update and Release Instructions

Adrian Duşa

2025-05-23

1 Setup Overview

This document explains how the auto-update functionality works in the StatConverter Electron app and how to publish updates properly across macOS, Windows, and Linux.

2 Build Configuration

In `package.json`, the `build` section is used by `electron-builder` to configure how your app is packaged and published:

```
"build": {
  "appId": "StatConverter.tool",
  "productName": "StatConverter",
  "directories": {
    "output": "build/output"
  },
  "win": {
    "icon": "build/StatConverter.ico",
    "legalTrademarks": "MIT Licence",
    "target": {
      "target": "nsis",
      "arch": [
        "x64"
      ]
    }
  },
  "mac": {
    "category": "public.app-category.productivity",
    "target": {
      "target": "dmg",
      "arch": [
        "x64",
        "arm64"
      ]
```

```
    },
    "icon": "build/StatConverter.icns"
  },
  "publish": {
    "provider": "github",
    "owner": "RODA",
    "repo": "StatConverter"
  }
}
```

The `"publish"` field tells `electron-builder` where to upload release binaries. This is essential for enabling `electron-updater` to check GitHub for updates and download new versions automatically.

The Linux part is missing from the `build` section. When on this platform, the following command will be used to build the app:

```
npx electron-builder --linux --x64 --arm64
```

This command will create an `.AppImage` file for that specific architecture in the `build/output` directory.

3 Auto-Updater

Implemented in `main.ts`:

```
import { autoUpdater } from "electron-updater";

app.whenReady().then(() => {
  createWindow();
  initWebR();
  if (production) {
    autoUpdater.checkForUpdatesAndNotify();
  }
});

autoUpdater.on("update-available", () => {
  dialog.showMessageBox(mainWindow, {
    type: "info",
    title: "Update Available",
    message:
      "A new version is available. It will be downloaded in the background.",
  });
});

autoUpdater.on("update-downloaded", () => {
  dialog
```

```
    .showMessageBox(mainWindow, {
      type: "question",
      buttons: ["Restart", "Later"],
      defaultId: 0,
      cancelId: 1,
      title: "Update Ready",
      message: "Update downloaded. Restart now to apply it?",
    })
    .then((result) => {
      if (result.response === 0) autoUpdater.quitAndInstall();
    });
});
```

4 Publishing Instructions

4.1 Local Build (No Upload)

```
npm run dist
```

Use when testing locally. This builds but does **not upload** the app.

4.2 Publish to GitHub (Per Platform)

Run the build on each platform (macOS, Windows, Linux) **independently**, using:

```
GH_TOKEN=your_token npm run dist
```

This builds the installer for the current platform and prepares files for manual upload.

4.3 Renaming Binaries (Before Upload)

After building, rename the output files using platform-specific scripts:

- **macOS/Linux**: run the Bash script to rename `.dmg` or `.AppImage` based on version and architecture.

  ```
  bash scripts/rename-unix-binaries.sh
  ```

- **Windows**: run the PowerShell script to rename the `.exe` binary.

  ```
  .\scripts\rename-binaries-windows.ps1
  ```

This ensures consistent filenames like:

- `StatConverter_1.2.0_intel.dmg`
- `StatConverter_1.2.0_intel.AppImage`
- `StatConverter_setup_1.2.0.exe`

Note: For Windows, the rename is version-based only; architecture is no longer included in filenames since only 64-bit builds are produced.

After renaming, upload all binaries manually to the **same GitHub release** corresponding to the current version in `package.json`.

4.4 Automating with `package.json` Scripts

You can simplify your build and rename process by adding scripts to `package.json`. For example:

```
"scripts": {
  "dist": "npm run build && electron-builder",
  "rename:unix": "bash scripts/rename-unix-binaries.sh",
  "rename:win": "powershell -ExecutionPolicy Bypass -File scripts/rename-binaries-windows.ps1"
  "dist:unix": "npm run dist && npm run rename:unix",
  "dist:win": "npm run dist && npm run rename:win"
}
```

Usage:

- On macOS or Linux:

  ```
  npm run dist:unix
  ```

- On Windows:

  ```
  npm run dist:win
  ```

These commands will build and rename the binary in one step.

5 Renaming Script Details

5.1 `scripts/rename-unix-binaries.sh`

This Bash script is used on macOS and Linux to rename the output binary (`.dmg` or `.AppImage`) based on the version and architecture.

- It automatically detects:

  - Platform (`Darwin` or `Linux`)
  - Architecture (`x86_64`, `arm64`, etc.)

- It renames files like:
  - `StatConverter-1.2.0.dmg` → `StatConverter_1.2.0_intel.dmg`
  - `StatConverter-1.2.0.AppImage` → `StatConverter_1.2.0_intel.AppImage`

5.1.1 Script contents:

```
#!/bin/bash

VERSION=$(node -p "require('./package.json').version")
ARCH=$(uname -m)
PLATFORM=$(uname -s)

if [ "$PLATFORM" = "Darwin" ]; then
    # macOS section
    if [ "$ARCH" = "arm64" ]; then
        NEW_NAME="StatConverter_${VERSION}_silicon.dmg"
    elif [ "$ARCH" = "x86_64" ]; then
        NEW_NAME="StatConverter_${VERSION}_intel.dmg"
    else
        echo "Unknown macOS architecture: $ARCH"
        exit 1
    fi
    ORIGINAL_FILE="build/output/StatConverter-${VERSION}.dmg"

elif [ "$PLATFORM" = "Linux" ]; then
    # Linux section
    if [ "$ARCH" = "x86_64" ]; then
        NEW_NAME="StatConverter_${VERSION}_intel.AppImage"
    elif [ "$ARCH" = "aarch64" ]; then
        NEW_NAME="StatConverter_${VERSION}_arm.AppImage"
    else
        echo "Unknown Linux architecture: $ARCH"
        exit 1
    fi
    ORIGINAL_FILE="build/output/StatConverter-${VERSION}.AppImage"

else
    echo "Unsupported platform: $PLATFORM"
    exit 1
fi

# Rename if file exists
if [ -f "$ORIGINAL_FILE" ]; then
```

```
    mv "$ORIGINAL_FILE" "build/output/$NEW_NAME"
    echo "Renamed to $NEW_NAME"
else
    echo "Original file not found: $ORIGINAL_FILE"
    exit 1
fi
```

5.2 scripts/rename-binaries-windows.ps1

This PowerShell script is used on Windows to rename the installer binary based on the version in `package.json`.

- Detects architecture (`x86`, `x64`) but currently standardizes output to 64-bit naming only.
- Renames:
  - `StatConverter Setup 1.2.0.exe` → `StatConverter_setup_1.2.0.exe`

These scripts ensure that all output files follow a consistent naming convention for GitHub Releases.

5.2.1 Script contents:

```
# Get version from package.json
$packageJson = Get-Content "./package.json" -Raw | ConvertFrom-Json
$version = $packageJson.version

# Define original and new file names
$originalFile = "build/output/StatConverter Setup $version.exe"
$newName = "StatConverter_setup_${version}.exe"
$newPath = "build/output/$newName"

# Rename if file exists
if (Test-Path $originalFile) {
    Rename-Item -Path $originalFile -NewName $newName
    Write-Host "Renamed to $newName"
} else {
    Write-Error "Original file not found: $originalFile"
    exit 1
}
```

6 Tips

- Always increment `"version"` in `package.json` before releasing.
- Ensure the `GH_TOKEN` has `repo` access rights.
- All artifacts will attach to a single GitHub release for auto-update compatibility.