Updating Applications

There are several ways to provide automatic updates to your Electron application. The easiest and officially supported one is taking advantage of the built-in Squirrel framework and Electron's autoUpdater module.

Using cloud object storage (serverless)

For a simple serverless update flow, Electron's autoUpdater module can check if updates are available by pointing to a static storage URL containing latest release metadata.

When a new release is available, this metadata needs to be published to cloud storage alongside the release itself. The metadata format is different for macOS and Windows.

Publishing release metadata

With Electron Forge, you can set up static file storage updates by publishing metadata artifacts from the ZIP Maker (macOS) with `macUpdateManifestBaseUrl` and the Squirrel.Windows Maker (Windows) with `remoteReleases`.

See Forge's Auto updating from S3 guide for an end-to-end example.

> **Manual publishing**

Reading release metadata

The easiest way to consume metadata is by installing update-electron-app, a drop-in Node.js module that sets up autoUpdater and prompts the user with a native dialog.

For static storage updates, point the `updateSource.baseUrl` parameter to the directory containing your release metadata files.

main.js

```
const { updateElectronApp, UpdateSourceType } = require('update-electron-app')

updateElectronApp({
  updateSource: {
    type: UpdateSourceType.StaticStorage,
    baseUrl: `https://my-bucket.s3.amazonaws.com/my-app-updates/${process.platform}/${proc
  }
})
```

Using update.electronjs.org

The Electron team maintains update.electronjs.org, a free and open-source webservice that Electron apps can use to self-update. The service is designed for Electron apps that meet the following criteria:

- App runs on macOS or Windows
- App has a public GitHub repository
- Builds are published to GitHub Releases
- Builds are code-signed (**macOS only**)

The easiest way to use this service is by installing update-electron-app, a Node.js module preconfigured for use with update.electronjs.org.

Install the module using your Node.js package manager of choice:

npm | Yarn

```
npm install update-electron-app
```

Then, invoke the updater from your app's main process file:

main.js

```
require('update-electron-app')()
```

By default, this module will check for updates at app startup, then every ten minutes. When an update is found, it will automatically be downloaded in the background. When the download completes, a dialog is displayed allowing the user to restart the app.

If you need to customize your configuration, you can pass options to update-electron-app or use the update service directly.

Using other update services

If you're developing a private Electron application, or if you're not publishing releases to GitHub Releases, it may be necessary to run your own update server.

Step 1: Deploying an update server

Depending on your needs, you can choose from one of these:

- Hazel – Update server for private or open-source apps which can be deployed for free on Vercel. It pulls from GitHub Releases and leverages the power of GitHub's CDN.
- Nuts – Also uses GitHub Releases, but caches app updates on disk and supports private repositories.
- electron-release-server – Provides a dashboard for handling releases and does not require releases to originate on GitHub.
- Nucleus – A complete update server for Electron apps maintained by Atlassian. Supports multiple applications and channels; uses a static file store to minify server cost.

Once you've deployed your update server, you can instrument your app code to receive and apply the updates with Electron's autoUpdater module.

Step 2: Receiving updates in your app

First, import the required modules in your main process code. The following code might vary for different server software, but it works like described when using Hazel.

> ⚠ CHECK YOUR EXECUTION ENVIRONMENT!
>
> Please ensure that the code below will only be executed in your packaged app, and not in development. You can use the app.isPackaged API to check the environment.

main.js

```
const { app, autoUpdater, dialog } = require('electron')
```

Next, construct the URL of the update server feed and tell autoUpdater about it:

main.js

```
const server = 'https://your-deployment-url.com'
const url = `${server}/update/${process.platform}/${app.getVersion()}`

autoUpdater.setFeedURL({ url })
```

As the final step, check for updates. The example below will check every minute:

main.js

```
setInterval(() => {
  autoUpdater.checkForUpdates()
}, 60000)
```

Once your application is packaged, it will receive an update for each new GitHub Release that you publish.

Step 3: Notifying users when updates are available

Now that you've configured the basic update mechanism for your application, you need to ensure that the user will get notified when there's an update. This can be achieved using the autoUpdater API events:

main.js

```
autoUpdater.on('update-downloaded', (event, releaseNotes, releaseName) => {
  const dialogOpts = {
    type: 'info',
    buttons: ['Restart', 'Later'],
    title: 'Application Update',
    message: process.platform === 'win32' ? releaseNotes : releaseName,
    detail:
      'A new version has been downloaded. Restart the application to apply the updates.'
  }

  dialog.showMessageBox(dialogOpts).then((returnValue) => {
    if (returnValue.response === 0) autoUpdater.quitAndInstall()
  })
})
```

Also make sure that errors are being handled. Here's an example for logging them to `stderr`:

main.js

```
autoUpdater.on('error', (message) => {
  console.error('There was a problem updating the application')
  console.error(message)
})
```

> ⓘ HANDLING UPDATES MANUALLY
>
> Because the requests made by autoUpdate aren't under your direct control, you may find situations that are difficult to handle (such as if the update server is behind authentication). The `url` field supports the `file://` protocol, which means that with some effort, you can sidestep the server-communication aspect of the process by loading your update from a local directory. Here's an example of how this could work.

Update server specification

For advanced deployment needs, you can also roll out your own Squirrel-compatible update server. For example, you may want to have percentage-based rollouts, distribute your app through separate release channels, or put your update server behind an authentication check.

Squirrel.Windows and Squirrel.Mac clients require different response formats, but you can use a single server for both platforms by sending requests to different endpoints depending on the value of `process.platform`.

main.js

```
const { app, autoUpdater } = require('electron')

const server = 'https://your-deployment-url.com'
// e.g. for Windows and app version 1.2.3
// https://your-deployment-url.com/update/win32/1.2.3
const url = `${server}/update/${process.platform}/${app.getVersion()}`

autoUpdater.setFeedURL({ url })
```

Windows

A Squirrel.Windows client expects the update server to return the `RELEASES` artifact of the latest available build at the `/RELEASES` subpath of your endpoint.

For example, if your feed URL is `https://your-deployment-url.com/update/win32/1.2.3`, then the `https://your-deployment-url.com/update/win32/1.2.3/RELEASES` endpoint should return the contents of the `RELEASES` artifact of the version you want to serve.

https://your-deployment-url.com/update/win32/1.2.3/RELEASES

```
B0892F3C7AC91D72A6271FF36905FEFBFE993520 https://your-static.storage/your-app-1.2.3-full.
```

Squirrel.Windows does the comparison check to see if the current app should update to the version returned in `RELEASES`, so you should return a response even when no update is available.

macOS

When an update is available, the Squirrel.Mac client expects a JSON response at the feed URL's endpoint. This object has a mandatory `url` property that maps to a ZIP archive of the app update. All other properties in the object are optional.

https://your-deployment-url.com/update/darwin/0.31.0

```
{
    "url": "https://your-static.storage/your-app-1.2.3-darwin.zip",
    "name": "1.2.3",
    "notes": "Theses are some release notes innit",
    "pub_date": "2024-09-18T12:29:53+01:00"
}
```

If no update is available, the server should return a `204 No Content` HTTP response.

Edit this page

Previous « Code Signing

Next Mac App Store Submission Guide »