Updating Applications

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

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

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:

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

Invoke the updater from your app's main process file:

```
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.

Deploying an Update Server

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.

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.

Implementing Updates in Your App

Once you've deployed your update server, continue with importing the required modules in your code. The following code might vary for different server software, but it works like described when using Hazel.

Important: Please ensure that the code below will only be executed in your packaged app, and not in development. You can use electron-is-dev to check for the environment.

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

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

```
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:

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

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

Applying Updates

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:

```
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`:

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

Handling Updates Manually

Because the requests made by Auto Update 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 does support files, which means that with some effort, you can sidestep the server-communication aspect of the process. Here's an example of how this could work.