Creating A New Electronite Release

Summary

- Electronite is a fork of Electron (latest release notes at https://github.com/unfoldingWord/electronite/releases) that is patched for Graphite Font support (About Graphite from SIL).
    - See the release notes for links https://github.com/unfoldingWord/electronite/releases to build notes, etc.
- There is a paired repo https://github.com/unfoldingWord-dev/electronite-cli/releases that is for the Electronite installer that is posted on the repository site (https://www.npmjs.com).
- Also there is the Electronite Packager

Creating The Release

Notes

- ☐ Before starting In the steps below, make copy of this document so you can check off the boxes as you go, and:
    - ☐ Replace the version `25.3.2` with the version of the new Electron version you want to build on (see releases at https://github.com/electron/electron/releases)
    - ☐ Then replace version `23.3.10` with last Electronite version released (e.g. `25.3.2`)

a. you need to build on Intel computer (currently for Windows and Linux, but for MacOS builds you now can build on ARM):
    i. Building for MacOS on Arm64 is now supported. Can build faster than the Intel 6 core Mac Mini if the processor has more than 6 performance cores. Was able to build locally using a Mac VM running on UTM (3.2.4).
b. for each OS (Mac, Windows, Linux), it can take a day or two to build each of the targets (depending on the performance of the build PC).
c. It's best to have dedicated build PCs or dedicated VMs (even better is cloud based VPC which enables setting up multiple machines in parallel). Here are some build requirements:
    i. Using a fast internet connection and fast drive it can take as little as 45 minutes to download and patch source files with cached git files. Otherwise it can take over 4 hours on a clean PC.
    ii. doing the builds can use over 100GB of disk space (particularly for arm64 builds). So it's best to have at least 160GB of primary disk space (80GB free after old builds are deleted) or have a second drive with 100GB of storage.
    iii. It's best to have at least 4 cores and a fast disk.
    iv. Best to have at least 16GB RAM and fast disk or the final link phase can take many hours due to disk swapping.
    v. It looks like **Oracle Cloud Compute** is cheapest and fastest for doing Windows and Linux builds (no Mac). It's only about $10 a build for Windows and Linux architectures.

- The most efficient configuration I found was (each build takes 3 to 4 hrs):
  Shape: VM.Standard.E3.Flex
  OCPU count: 6
  CPU Model: AMD EPYC 7742 64-Core
  CPU Core: 12
  Network bandwidth (Gbps): 6
  Memory (GB): 32
- Seems to give fast builds at minimal cost

vi. **AWS Compute** looks much more expensive, but has MacOS support (unfortunately you are required to rent the Mac for a minimum of 24 hours). The full costs seem to be about 2x the cost of the EC2 compute instance (due higher EC2 rates and extra charges for internet data, storage size, storage bandwidth, etc.).

d. Tried using the same CircleCI automated build system that Electron uses, but it only works for MacOS and Linux, not WIndows. Plus it uses the GOMA build system to speed up the build process so builds don't time out. We can use GOMA setting cache-only, but it doesn't have the same level of performance. We could use higher level performance machines, but CircleCI would require a yearly license.

e. Note that the build process is always changing for Electron, so if it doesn't build you should check the build instructions at `` `docs/development/build-instructions-*.md` `` in the repo of the latest release.

Create a new Electronite beta release

- Creating new Electronite beta branch
  - ☐ If you haven't done this yet, you will need to clone https://github.com/unfoldingWord/electronite
  - ☐ Now cd into the `electronite` folder
  - ☐ Run script file to create new branches and copy files from previous Electronite branch.
    - ☐ First do `` `chmod +x ./docs/development/Electronite/make_new_electronite.sh` ``.
    - ☐ Make sure `` `--no-verify` `` Is applied to pushes in script.
    - ☐ Then do `` `./docs/development/Electronite/make_new_electronite.sh v25.3.2 electronite-v23.3.10-beta` `` (source below) which will:
      - get the tag `` `v25.3.2` `` from the upstream electron repo
      - make both a new electron branch `` `v25.3.2-electron` `` and a new electronite branch `` `electronite-v25.3.2-beta` `` in the electronite repo
      - copy files from old electronite branch `` `electronite-v23.3.10` ``
      - commit changes to new electronite branch `` `electronite-v25.3.2-beta` ``
  - ☐ Update the `` `./electronite/docs/development/Electronite/electronite.patch` `` :

- [ ] Manually go through `electronite.patch` tweaking line numbers to match line numbers in new files.
- [ ] Manually go through the patch and make the changes to the files (except for creating the file `./electronite/patches/chromium/add_graphite.patch`):
    - [ ] edit `./DEPS`
    - [ ] edit `./patches/chromium/.patches`

- [ ] search and replace in the source files for instances old version number (such as `23.3.10`) and replace with `25.3.2`

- [ ] Commit changes and push up.
    - [ ] Be careful here - the newer Electron releases are running pre-commit hooks which may fail on your setup. So if you get an error on git commit, try doing `git commit --no-verify` To skip the pre-commit.
    - [ ] Do git push: `git push --no-verify https://<<TOKEN>>@github.com/unfoldingWord/electronite.git`

- [ ] Test by building using a Intel computer on Windows or on Arm for Mac (or Intel):
    - [ ] Delete the cached Electronite repo in git_cache:
        - [ ] Look in `git_cache` and delete the folder and lock file that contains `unfoldingWord/electronite`. Otherwise it will not use the recent changes, branches, commits.
    - [ ] use build instructions in `./docs/development/Electronite`. Use appropriate instructions in the sections: AWS Windows VM Setup , AWS Linux VM Setup , or AWS MacOS VM Setup

    - [ ] Follow steps for setting up the build environment (if not up to date).
    - [ ] And follow the steps to build all.
    - [ ] If fetching sources fails at `add_graphite.patch`:
        - [ ] Update/fix `add_graphite.patch` and push up to branch.
            i. If hand editing doesn't work, then apply the changes to the files and do `git diff HEAD > patch.patch`
            ii. Copy the content of `patch.patch` and paste it into the first part of the `add_graphite.patch` file (except for the files to create).
            iii. Commit the changes and push it up to the branch

        - [ ] Delete source folder (e.g. `~/Develop/Electronite-Build/src`)
        - [ ] Start over again with test
    - [ ] If build fails on graphite due to cpp deprecations, then you will have to update our custom patch `add_graphite_cpp_std_iterator.patch`. If hand editing doesn't work, then apply the changes to the files and do `git diff HEAD > patch.patch` to generate the first part of the file (except for files to create).

- [ ] fix the compile error in the cpp source code in `.\src\third_party\graphite\graphite2`. *Note - We hope that eventually SIL will update their code, and then we can remove this patch from build scripts/batch.*
- [ ] Once it compiles past graphite make an updated patch) by doing:
  - [ ] `cd src/third_party/graphite/graphite2 && git diff HEAD > ~/graphite.patch`
  - [ ] make copy of patch for Electronite: `cp ~/graphite.patch ~/add_graphite_cpp_std_iterator.patch`
  - [ ] in `~/add_graphite_cpp_std_iterator.patch` do search for instances of `a/src` and replace with `a/third_party/graphite/graphite2/src`
  - [ ] Also create issue in `https://github.com/silnrsi/graphite/issues/78` to share the fix with others and give content of `graphite.patch`
  - [ ] update `./electronite/docs/development/Electronite/add_graphite_cpp_std_iterator.patch` in ELectronite branch with contents of `~/add_graphite_cpp_std_iterator.patch` and commit/push
- [ ] Make sure that `.\src\third_party\graphite` has been downloaded. Otherwise fix patches and start over with test

- [ ] Save the latest version of these instructions (with boxes unchecked) to Electronite repo at `docs/development/Electronite/Creating new Electronite Release.pdf`
- [ ] Commit and Push up your fixes to the Electronite branch
- [ ] Do builds for Mac, Windows, and Linux using notes in `docs/development/Electronite` => `MacBuildNotes.md`, `LinuxBuildNotes.md`, and `WindowsBuildNotes.md`.
  - [ ] *Note* - builds take up lots of disk space, so after building each target, copy the `dist.zip` folder from `./src/out/Release-*` and rename it appropriate to version/OS/architecture such as `electronite-v25.3.2-graphite-beta-win32-x64.zip`. Then delete folder `./src/out` before building for the next architecture.
  - [ ] If you run out of space, particularly building for arm64, you can delete the `git_cache` folder which should free up over 20GB of space.
- [ ] At https://github.com/unfoldingWord/electronite/releases, create a draft release on Electronite using the previous release as a template (set it up to create a tag with name `v25.3.2-graphite-beta` when release is made).
  - [ ] Attach all the builds renaming the dist.zip file to match the tag name (`v25.3.2-graphite-beta`) and architecture of the build (again using the previous release as a template)
  - [ ] Attach the patched `electron.d.ts` from Electronite branch
- [ ] Create a pre-release for testing

Create a new electronite-cli beta release

- [ ] *If you haven't already done this*, then clone the repo**:** https://github.com/unfoldingWord-dev/electronite-cli
- [ ] Checkout the latest release and create a branch `electronite-v25.3.2-beta`
    - [ ] Update version in package.json to be `25.3.2-graphite-beta`
    - [ ] Replace `` `electron.d.ts` `` with the patched `` `electron.d.ts` `` from above
    - [ ] Commit and push the changes
- [ ] Test package by **deleting the dist folder**, and then run `` `npm i --legacy-peer-deps && npm pack` ``, this should create a file `electronite-25.3.2-graphite-beta.tgz`
- [ ] Create a test branch in translationCore (such as `` `feature-electronite-v25.3.2` ``)
    - [ ] Import the electronite tgz file from step 2: `` `npm i --legacy-peer-deps --save-dev <<path-to-tgz>>` `` **<=** replace `` `<<path-to-tgz>>` `` with actual path to file
    - [ ] Make sure translationCore will start up `` `npm i --legacy-peer-deps && npm start` ``
    - [ ] make sure translationCore version is correctly displayed in app.
    - [ ] Check the log file to make sure it is running the right version of electron
    - [ ] Make sure graphite is working:
        - [ ] Look at this issue for how to test: https://github.com/unfoldingWord/translationCore/issues/6788
        - [ ] Example `ar` project: https://git.door43.org/ElsyLambert/ar_new_phm_book
- [ ] Now delete the dist folder, and npm publish the electronite-cli beta branch: `` `npm i --legacy-peer-deps && npm publish --tag beta` ``
- [ ] in translationCore remove electronite tgz file dependency: `` `npm uninstall electronite` ``
- [ ] update translationCore to use the published package: `` `npm i --legacy-peer-deps --save-dev electronite@v25.3.2-graphite-beta` ``
- [ ] in package.json remove the ^ before Electronite version.
- [ ] Create builds and test:
    - [ ] commit and push up changes
    - [ ] create a translationCore PR for this so github actions will try to create all the versions (ensures that all Electronite dist zip files are named correctly)
    - [ ] test the builds - particularly macos x64 and windows x64:
        - [ ] install and start up translationCore builds.
        - [ ] Check the log file to make sure it is running the right version of electron
        - [ ] Make sure graphite is working:
            a. Look at this issue for how to test: https://github.com/unfoldingWord/translationCore/issues/6788
            b. Example `ar` project: https://git.door43.org/ElsyLambert/ar_new_phm_book
- [ ] At https://github.com/unfoldingWord-dev/electronite-cli/releases, create a prelim release:
    - [ ] Commit and push up the changes to the new branch.

- [ ] Draft a new release based on `electronite-v25.3.2-beta`
- [ ] Set up to create a new tag `v25.3.2-graphite-beta` on publish.
- [ ] Publish as pre-Release

Creating the final releases

- [ ] Make another Electronite draft release (https://github.com/unfoldingWord/electronite/releases) using the same branch as above `electronite-v25.3.2-beta` and select to create tag `v25.3.2-graphite` on release
  - [ ] Attach the same Electronite builds as the prerelease beta, but rename them by removing `beta-` from name.
  - [ ] Change description to take beta out of links
  - [ ] Attach the latest `electron.d.ts` patched for electronite
  - [ ] Save the latest version of these instructions (with boxes unchecked) to Electronite repo at `docs/development/Electronite/Creating new Electronite Release.pdf`
  - [ ] Publish this draft as a pre-release for the moment

- [ ] Make another electronite-cli branch: https://github.com/unfoldingWord-dev/electronite-cli
  - [ ] Checkout the latest branch `electronite-v25.3.2-beta` and create a new branch `electronite-v25.3.2`:
    - [ ] Update version in package.json to be `25.3.2-graphite`
    - [ ] Commit and push the changes
  - [ ] Test package by deleting the dist folder, and then run `npm i --legacy-peer-deps && npm pack`, this should create a file such as `electronite-25.3.2-graphite.tgz`
  - [ ] If that succeeds, commit changes and push up new branch.
  - [ ] Make an electronite-cli release selecting to create tag `v25.3.2-graphite` when done
  - [ ] change release description to take beta out.
  - [ ] Publish branch: `npm i --legacy-peer-deps && npm publish`

- [ ] Test builds in translationCore:
  - [ ] Remove electronite tgz file dependency: `npm uninstall electronite`
  - [ ] update translationCore to use the published package: `npm i --legacy-peer-deps --save-dev electronite@v25.3.2-graphite`
  - [ ] in package.json remove the ^ before Electronite version.
  - [ ] Do `npm i --legacy-peer-deps && npm start` to make sure electronite downloads and runs
  - [ ] Commit and push the tCore changes.
  - [ ] Wait for github actions to auto-build the updated branch. If it succeeds then the electronite dist zip files are named correctly

- [ ] Make the Electronite (https://github.com/unfoldingWord/electronite/releases) pre-release for `v25.3.2-graphite` the latest release.

- [ ] *Not sure this is needed anymore now that we directly check out the build branch in the build scripts:* Merge released branch into graphite branch and push - lots of changes typically
    - [ ] checkout and update the released branch,
    - [ ] merge in origin/graphite
    - [ ] diff with released branch - download all changes from branch
    - [ ] commit and push
    - [ ] make PR for merge to graphite
    - [ ] merge PR

AWS VM Setups

AWS MacOS VM Setup

Summary

Now can build on Arm (much faster) as well as Intel

On AWS - can run on Dedicated Mac Intel or Arm Hardware:

- mac1.metal - intel - which must be reserved for at least 24hr (about $26)

    Mac mini (2018)
    3.2 GHz 6-Core Intel Core i7
    32 GB 2667 MHz DDR4

- create an instance with Monterey OS and mac1.metal. Allocate storage as gb3, 300GB now 300MB/s, 6000iops (tried setting to max 1000MB/s, 16000iops, but didn't see speed-up). When the instance starts the partition will only have 100GB allocated. Using the Disk Utility add a new partition and apply it. It only seems to create the new partition with 6GB and will fail any resize attempt using the Disk Utility. So use the section below `Using the Mac Mini's Built-in SSD`

- Could build using the Mac Mini's SSD - but didn't see significant speedup. It is only 128GB (which might explain the slowness) . But after getting the sources, Electronite does not require a super fast storage - only large amounts.

Build Steps

Automated Build of All Architectures

Change to build folder and do (where <version> is like v22.0.0):

Get Newest Build scripts for version, for example download:

Without **Goma** acceleration (faster on slow internet, less RAM, slow storage, and fewer cores), get these files:

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_all_mac.sh

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_target_mac.sh

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-3.sh

Goma acceleration is not working anymore => ~~To use **Goma** acceleration (faster with fast internet, much RAM, fast storage, and more cores), get these files:~~

~~https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_all_goma_mac.sh~~

~~https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_target_goma_mac.sh~~

~~https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-goma-3.sh~~

Initial Clear:

If you want to do a clean build where all sources are fetched again do:

```
rm -rf ./git_cache
```

Otherwise just clear the Electronite cache:

```
rm -rf ./git_cache/github.com-unfoldingword-electronite
rm ./git_cache/github.com-unfoldingword-electronite.locked
```

Next do setup:

```
# setup:
export PATH=$(pwd)/depot_tools:$PATH
# clear old sources
mv src src.old
rm -rf src.old &
```

Start the build:

Without Goma:

```
# and start the build
./build_all_mac.sh electronite-<version>-beta
results/mac/<version>
```

(e.g. `./build_all_mac.sh electronite-v25.3.2-beta results/mac/v25.3.2`)

With Goma:

```
export GOMA=cache-only
./build_all_goma_mac.sh electronite-<version>-beta
results/mac/<version>
```

Troubleshooting:

- If you need a different version of xcode, best to download the specific version directly from Apple - while need developer account
- If you then get asked again to download xcode command-line tools, do: `` `sudo xcode-select -switch /Library/Developer/CommandLineTools` ``

Manual Build of All Architectures

Get Newest Build scripts for version, for example download:

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-3.sh

Next do:

```
export PATH=$(pwd)/depot_tools:$PATH
./electronite-tools-3.sh get electronite-v22.0.0-beta

./electronite-tools-3.sh build x64
./electronite-tools-3.sh release x64

./electronite-tools-3.sh build arm64
./electronite-tools-3.sh release arm64
```

Connect with VNC over SSH

```
# open ssh with tunnel
ssh -i "~/.Keys/BruceMcL.pem" -C -L 25900:localhost:5900
ec2-user@ec2-18-236-233-104.us-west-2.compute.amazonaws.com

# do vnc over tunnel
open vnc://localhost:25900
```

Using the Mac Mini's Built-in SSD

Initializing It

```
diskutil list
diskutil info /dev/disk0

# initialize and format disk as BuildDisk
```

```
diskutil eraseDisk APFS BuildDisk /dev/disk0
```

Using it for builds

- To use Mac Mini's internal (128GB) SSD for builds (see writes peaking at 520MB/s):

```
mkdir -p /Volumes/BuildDisk/Build-Electronite
cd /Volumes/BuildDisk/Build-Electronite

cp /Volumes/DataDisk/Build-Electronite/*.sh
/Volumes/BuildDisk/Build-Electronite

ln -s /Volumes/DataDisk/Build-Electronite/git_cache
ln -s /Volumes/DataDisk/Build-Electronite/logs
ln -s /Volumes/DataDisk/Build-Electronite/results
```

Moving Goma cache to DataDisk

- If Goma is already running, move cache:

```
~/.electron_build_tools/third_party/goma/goma_ctl.py ensure_stop
mv ~/.gome_output_cache /Volumes/DataDisk/.gome_output_cache
ln -s /Volumes/DataDisk/.gome_output_cache ~/.gome_output_cache
~/.electron_build_tools/third_party/goma/goma_ctl.py ensure_start
```

- ~~Or if link is already created:~~

```
~~mkdir -p /Volumes/DataDisk/.gome_output_cache~~
```

VM Setup Steps for Electronite Build

- Need MacOS Ventura

- Arm64-specific prerequisites
  - Rosetta 2
    - We recommend installing Rosetta if using dependencies that need to cross-compile on x64 and arm64 machines. Rosetta can be installed by using the softwareupdate command line tool.

```
softwareupdate --install-rosetta
```

- enable VNC over SSH access:

```
sudo defaults write
/var/db/launchd.db/com.apple.launchd/overrides.plist
com.apple.screensharing -dict Disabled -bool false
sudo launchctl load -w
/System/Library/LaunchDaemons/com.apple.screensharing.plist

# the first time set user password
#   sudo /usr/bin/dscl . -passwd /Users/ec2-user
```

- Install Build tools needed by Electronite in MacOS

Note - need to add step of creating new partition named DataDisk

```
# Fix Disk - in AWS does not use all the volume space
diskutil list
diskutil repairdisk /dev/disk1
# grow partition, to fill disk use 0
diskutil apfs resizeContainer disk1s3 0

# xcode 13.3.1 direct download URL - will prompt to login - download to
/Volume/MacData/Mac_install
#
https://developer.apple.com/services-account/download?path=/Developer_Tools/Xcode_13.3.1/Xcode_13.3.1.xip

# Install brew (not needed for AWS)
# /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# accept xcode license
sudo xcodebuild -license accept

# uncompress xcode into current folder
cd ~
xip -x /Volume/MacData/Mac_install/Xcode_14.3.0.xip
mv ~/Xcode.app /Applications/Xcode.app
# select new xCode
sudo xcode-select --switch /Applications/Xcode.app

# manually launch Xcode, and accept prompt to install additional debug tools

# WARNING
# don't use the python 3 installed by brew, it doesn't have certificates

# download python 3.10 mac installer from (3.11 has problems with v21)
https://www.python.org/downloads/release/python-3109/
# run the installer
```

 # next go into `/Library/Frameworks/Python.framework/Versions/3.10/bin/python3` and run the `Install Certificates.command`

download python 2 mac installer from https://www.python.org/downloads/release/python-2718/
run the installer

```
# set default python to be python 2
sudo cp /bin/python2 /bin/python

# Install Node Version Manager
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.37.2/install.sh |
bash

# initialize Node Version Manager
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This
loads nvm bash_completion

# install node version 16
nvm install v16

# create build location, this is on a separate partition
mkdir -p /Volumes/DataDisk/Build-Electronite/
cd /Volumes/DataDisk/Build-Electronite/

# install chromium build tools
git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git
```

Notes for Future:

- Still too slow. Areas to explore:
    - Goma builds (cache-only) with fast SSD - should speed things up since fast internet.
        - Script worked on local machine, but Goma was too slow - need to test on AWS.
        - On the Mac it kept running out of disk space on boot drive. Goma folder kept growing (got up to 25GB before the build crashed). Trying to move cache file `~/.gome_output_cache` to different drive

    - Discovered that Electronite now builds on Arm64 on the Mac, so we could use the `mac2.metal` instance which is only 0.65/1.1*100=59% of the cost, but project that total build times without Goma would go from 8.6 to 10.5hrs on the Mac Mini! Quicker and cheaper to build on my 16" MBP in only 6 hrs total!

AWS Linux VM Setup

Summary

Started with Ubuntu since Amazon Linux uses yum and setup would have to be transposed.

Runs on EC2 Instances:

- Allocate storage volume as gb3, 300GB now 300MB/s, 6000iops.

- EC2 instance config:
  - **c6a.4xlarge -** AMD processor**,** 16 cpus, 32GB RAM - and will take about 10hrs
  - Using models with less RAM will take much longer at the link phase and use serious disk swap
  - Could use larger models, for example the **c6a.8xlarge -** costs about twice as much per hour as the 4x but should take ½ the time, 32 cpus, 64GB

  - Comparing compute types: https://aws.amazon.com/ec2/instance-types/
  - Comparing prices: https://us-west-2.console.aws.amazon.com/ec2/home?region=us-west-2#InstanceTypes:
  - Within a family, prices seem to be linear with performance for Electronite Builds (e.g. going from 4xLarge to 8xLarge doubles price but takes half as long)
  - Don't see a huge difference between c5 and c6 for Electronite Builds
  - currently c6 (or c6a) seems to be best price and value

Build Steps

Automated Build of All Architectures

Change to build folder and do (where <version> is like v22.0.0):

Get Newest Build scripts for version, for example download:

Without Goma acceleration (faster on slow internet, with less RAM, with slower SSD, and fewer cores)

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_all_linux.sh

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_target_linux.sh

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-3.sh

Goma acceleration is not working anymore => ~~With **Goma** acceleration (faster with fast internet, much RAM, fast storage, and more cores):~~

~~https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_all_goma_linux.sh~~

~~https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_target_goma_linux.sh~~

~~https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-goma-3.sh~~

Initial Clear:

If you want to do a clean build where all sources are fetched again do:

```
rm -rf ./git_cache
```

Otherwise just clear Electronite repo from the cache:

```
rm -rf ./git_cache/github.com-unfoldingword-electronite
rm ./git_cache/github.com-unfoldingword-electronite.locked
```

Next do setup:

```
# setup:
export PATH=$(pwd)/depot_tools:$PATH
# clear old sources
mv src src.old
rm -rf src.old &
```

Start the build:

Without Goma:

```
# and start the build
./build_all_linux.sh electronite-<version>-beta
results/linux/<version>
```

With Goma:

```
export GOMA=cache-only
./build_all_goma_linux.sh electronite-<version>-beta
results/linux/<version>
```

Manual Build of All Architectures

Get Newest Build scripts for version, for example download:

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-3.sh

Next do:

```
export PATH=$(pwd)/depot_tools:$PATH
./electronite-tools-3.sh get electronite-v22.0.0-beta

./electronite-tools-3.sh build x64
./electronite-tools-3.sh release x64

cd ./src
build/linux/sysroot_scripts/install-sysroot.py --arch=arm64
cd ..

./electronite-tools-3.sh build arm64
./electronite-tools-3.sh release arm64
```

VM Setup Steps for Electronite Build

```
#!/bin/bash
# Install Build tools needed by Electronite in Linux

sudo apt-get update
sudo apt-get upgrade -y

sudo apt-get install build-essential clang libdbus-1-dev libgtk-3-dev \
                     libnotify-dev libasound2-dev libcap-dev \
                     libcups2-dev libxtst-dev \
                     libxss1 libnss3-dev gcc-multilib g++-multilib curl \
                     gperf bison python3-dbusmock openjdk-8-jre -y

sudo apt-get install libc6-dev-arm64-cross linux-libc-dev-arm64-cross \
                     g++-aarch64-linux-gnu -y

sudo apt-get install binutils-aarch64-linux-gnu

sudo apt-get install python3 python2 nano -y

# set default python to be python 2
sudo cp /bin/python2 /bin/python

# Install Node Version Manager
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.37.2/install.sh | bash

# initialize Node Version Manager
```

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This loads nvm
bash_completion

# install node version 16
nvm install v16

# create build location
mkdir -p ~/Develop/Build-Electronite
cd ~/Develop/Build-Electronite

# install chromium build tools
git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git
```

In settings-> WIndows Defender -> Exclusions, and an exclusion for the build folder

Set system environment variables

```
set vs2019_install=c:\Program Files (x86)\Microsoft Visual Studio\2019\Community
set WINDOWSSDKDIR=c:\Program Files (x86)\Windows Kits\10
set DEPOT_TOOLS_WIN_TOOLCHAIN=0
```

AWS Windows VM Setup

Summary

Troubleshooting

~~Not yet working -~~ if build error at final stage saying `error: expected int, +, ~, or (, got graphite` . Make this fix:

Looks to be parsing of tags that it doesn't like the `-graphite` at end of tag. This doesn't seem to affect the other OS's:

```
cd src\electron
git describe --tags

# if last tag has -graphite, then local delete tags and
recreate
git tag -d <version>-graphite
git tag -d <version>
git tag <version>

# make sure just version tag shows up
git describe --tags
```

~~Now only arm64 build is failing~~ - needed to go back to older Python 3.10 version

Runs on EC2 Instances:

- Allocate storage volume as gb3, 300GB now 300MB/s, 6000iops.

- EC2 instance config:
  - **c6a.4xlarge -** AMD processor**,** 16 cpus, 32GB RAM - and will take about 10hrs
  - Using models with less RAM will take much longer at the link phase and use serious disk swap
  - Could use larger models, for example the **c6a.8xlarge -** costs about twice as much per hour as the 4x but should take ½ the time, 32 cpus, 64GB

  - Comparing compute types: https://aws.amazon.com/ec2/instance-types/
  - Comparing prices: https://us-west-2.console.aws.amazon.com/ec2/home?region=us-west-2#InstanceTypes:
  - Within a family, prices seem to be linear with performance for Electronite Builds (e.g. going from 4xLarge to 8xLarge doubles price but takes half as long)
  - Don't see a huge difference between c5 and c6 for Electronite Builds
  - currently c6 (or c6a) seems to be best price and value

Build Steps

Automated Build of All Architectures

Change to build folder and do (where <version> is like v22.0.0):

Initial Clear:

Get Newest Build scripts for version, for example download:

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/build_target_win.bat

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/meta_builds/set_build_env.bat

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-3.bat

If you want to do a clean build where all sources are fetched again do:

```
rmdir /s /q .\git_cache
```

Otherwise just clear the Electronite cache:

```
rmdir /s /q
.\git_cache\github.com-unfoldingword-electronite
del
.\git_cache\github.com-unfoldingword-electronite.locked
```

Next cd to the build folder and do:

```
# remove old source files so desire branch is fetched
rename src src.old
rmdir /s /q src.old

# and start the build
build_all_win.bat electronite-<version> results\<version>
```

Example: `build_all_win.bat electronite-v25.3.2-beta results\v25.3.2`

Troubleshooting - if it can't find gn do this:

```
set Path=%cd%\depot_tools;%Path%
```

Manual Build of All Architectures

Get Newest Build scripts for version, for example download:

https://github.com/unfoldingWord/electronite/blob/electronite-v22.0.0/docs/development/Electronite/electronite-tools-3.bat

Next cd to build folder and do:

```
.\electronite-tools-3.bat get electronite-<version>

.\electronite-tools-3.bat build x64;
.\electronite-tools-3.bat release x64;

.\electronite-tools-3.bat build x86;
.\electronite-tools-3.bat release x86;

.\electronite-tools-3.bat build arm64;
.\electronite-tools-3.bat release arm64;
```

VM Setup

VM Setup Steps for Electronite Build

```
rem Run in powershell with admin privileges
echo off
set working_dir=%cd%

echo "Install Electronite Build tools"

echo "%date% - %time%" > start_time_configure.txt

rem at some point in the future they will be moving to 2022
echo "Installing VS 2019 Community"
```

Import the Visual Studio Configuration In later section

```
vs_community_2019.exe ^
    --wait --quiet --norestart ^
    --installWhileDownloading ^
    --add Microsoft.VisualStudio.Workload.NativeDesktop ^
    --add Microsoft.VisualStudio.Component.VC.ATLMFC ^
    --add Microsoft.VisualStudio.Component.VC.Tools.ARM64 ^
    --add Microsoft.VisualStudio.Component.VC.MFC.ARM64 ^
    --add Microsoft.Component.MSBuild ^
    --add Microsoft.VisualStudio.Component.VC.CLI.Support ^
    --add Microsoft.VisualStudio.Component.VC.Llvm.Clang ^
    --add Microsoft.VisualStudio.Component.VC.CMake.Project ^
    --add Microsoft.VisualStudio.Component.VC.Tools.x86.x64 ^
    --add Microsoft.VisualStudio.Component.VC.CoreIde ^
    --add Microsoft.VisualStudio.Component.VC.Llvm.ClangToolset ^
    --add Microsoft.VisualStudio.Component.VC.14.20.ATL ^
```

```
      add Microsoft.VisualStudio.Component.VC.14.20.ATL.ARM64 ^
      add Microsoft.VisualStudio.Component.VC.14.20.CLI.Support ^
      add Microsoft.VisualStudio.Component.VC.14.20.MFC ^
      add Microsoft.VisualStudio.Component.VC.14.20.MFC.ARM64 ^
      includeRecommended

rem not sure if this is needed:
echo "Installing VS 2019 Build Tools"
vs_BuildTools_2019.exe

rem It looks like this is the SDK being used for builds (download from
https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/)
echo "Installing WinSDK 20348"
winsdksetup_10.0.20348.0.exe /features + /q /norestart
echo "Installing WinSDK 22621"
winsdksetup_10.0.22621.0.exe /features + /q /norestart

rem Not needed anymore?
echo "Installing WinSDK 19041 - hopefully everything gets selected"
winsdksetup_10.0.19041.0.exe /features + /q /norestart

echo "Setting up Python 2.7.18"
msiexec /log python27_install.log /i python-2.7.18.amd64.msi /qn ALL=1 TARGETDIR=c:\python27
ALLUSERS=1
mklink c:\python27\python2.exe c:\python27\python.exe

echo "Setting up Python 3.9.13"
python-3.9.13-amd64.exe /quiet TargetDir=c:\python311 InstallAllUsers=1 PrependPath=1
Include_test=0
rename c:\python311\python.exe c:\python311\python3.exe

Git-2.39.0-64-bit.exe /VERYSILENT /NORESTART

git config --system core.longpaths true
git config --global user.email "you@example.com"
git config --global user.name "Your Name"

echo "Install node"
msiexec /i node-v16.19.0-x64.msi /log node_install.log /qn
```

First, ensure that you are using an administrative shell - you can also install as a non-admin, check out Non-Administrative Installation.

```
echo "Install chocolatey"

# Install with powershell.exe
# from https://chocolatey.org/install#individual do

Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

npm i -g yarn
```

```
npm i -g @electron/build-tools
c:
cd %HOMEPATH%\.electron_build_tools
npm i

# create build location
md %HOMEDRIVE%%HOMEPATH%\Develop\Build-Electronite
cd %HOMEDRIVE%%HOMEPATH%\Develop\Build-Electronite

# install chromium build tools
git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git

echo "Restore initial directory
cd %working_dir%

echo "%date% - %time%" > end_time_configure.txt
```

Goma additional setup

Install **sed** from https://www.gnu.org/software/ and add path to bin `C:\Program Files (x86)\GnuWin32\bin` to path environment variable.

Setup System Environment Variables

```
set vs2019_install=c:\Program Files (x86)\Microsoft Visual
Studio\2019\Community
set WINDOWSSDKDIR=c:\Program Files (x86)\Windows Kits\10
set DEPOT_TOOLS_WIN_TOOLCHAIN=0

set
Path=C:\Users\Administrator\Develop\Build-Electronite\depot_tools;C:\pytho
n27\Scripts;C:\python27;C:\Program Files\Python39\Scripts\;C:\Program
Files\Python39\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:
\Windows\System32\WindowsPowerShell\v1.0\;C:\Program
Files\Amazon\cfn-bootstrap\;C:\Program Files (x86)\Windows Kits\10\Windows
Performance Toolkit\;C:\Program
Files\nodejs\;C:\ProgramData\chocolatey\bin;C:\Program
Files\Git\bin;C:\Users\Administrator\AppData\Local\Microsoft\WindowsApps;C
:\Users\Administrator\AppData\Roaming\npm
```

Visual Studio 2019 Configuration

```
{
  "version": "1.0",
  "components": [
```

```
        "Microsoft.VisualStudio.Component.CoreEditor",
        "Microsoft.VisualStudio.Workload.CoreEditor",
        "Microsoft.VisualStudio.Component.NuGet",
        "Microsoft.VisualStudio.Component.Roslyn.Compiler",
        "Microsoft.VisualStudio.Component.Roslyn.LanguageServices",
        "Microsoft.VisualStudio.ComponentGroup.WebToolsExtensions",
        "Microsoft.VisualStudio.Component.TypeScript.4.3",
        "Microsoft.VisualStudio.Component.JavaScript.TypeScript",
        "Microsoft.Component.MSBuild",
        "Microsoft.VisualStudio.Component.TextTemplating",
        "Microsoft.VisualStudio.Component.Debugger.JustInTime",
        "Component.Microsoft.VisualStudio.LiveShare",
        "Microsoft.VisualStudio.Component.IntelliCode",
        "Microsoft.VisualStudio.Component.VC.CoreIde",
        "Microsoft.VisualStudio.Component.VC.Tools.x86.x64",
        "Microsoft.VisualStudio.Component.Graphics.Tools",
        "Microsoft.VisualStudio.Component.VC.DiagnosticTools",
        "Microsoft.VisualStudio.Component.Windows10SDK.20348",
        "Microsoft.VisualStudio.Component.VC.Redist.14.Latest",
        "Microsoft.VisualStudio.ComponentGroup.NativeDesktop.Core",
        "Microsoft.VisualStudio.Component.VC.Tools.ARM64",
        "Microsoft.VisualStudio.ComponentGroup.WebToolsExtensions.CMake",
        "Microsoft.VisualStudio.Component.VC.CMake.Project",
        "Microsoft.VisualStudio.Component.VC.ATL",
        "Microsoft.VisualStudio.Component.VC.TestAdapterForBoostTest",
        "Microsoft.VisualStudio.Component.VC.TestAdapterForGoogleTest",
        "Microsoft.VisualStudio.Component.VC.ATLMFC",
        "Microsoft.VisualStudio.Component.VC.ASAN",
        "Microsoft.VisualStudio.Workload.NativeDesktop",
        "Microsoft.VisualStudio.Component.VC.ATL.ARM64",
        "Microsoft.VisualStudio.Component.VC.MFC.ARM64"
    ]
}
```

Notes for Future:

- Explore moving to Visual Studio 2022 Community.
    - Electron currently uses VS 2019, but Chromium build tools also supports VS 2022
    - Initial testing shows that VS 2022 would work, but may impact Goma if transition is made earlier than Electron does.
- Explore speed up using Goma builds (cache-only) with fast SSD - should speed things up since fast internet.