Building 4D apps with GitHub actions

Milan Adamov
milan@clearviewsys.com | madamov@mac.com

https://github.com/madamov/appBuilder

What is the goal?

- Automate building process releasing builds as publicly available archives allowing us to implement automatic updates of our standalone and client/server applications
- This required converting database to project mode, start using git, learn and still learning most efficient techniques how to work with git

What are GitHub actions?

- CI/CD platform for automation of certain tasks
- Use workflows (defined as YAML files stored in .github/workflows folder in your repository) that are triggered to run by certain action in repository (push, pull request, issue created,…), on schedule or manually
- Workflows will launch runner(s), then jobs will run within each runner

- Runner is freshly created Linux, Windows or macOS virtual machine that exists only until the last job assigned to it ends

- Job is made as series of steps of shell commands or shell scripts for particular platform

- You can use actions written by other developers in order not to write your own when you need something common or repetitive (for example checking out your repository)

- Secrets are place where passwords and other sensitive information needed in workflows can be stored. We use a dozen of them.

- Repository variables are values that workflows can read from and write to. We have two of them: one to store current build number, second to store current version number.

4D in headless mode

- 4D can run in headless mode, with or without data file
- It can skip startup method (we are not doing that), it can run specific method
- We can pass parameters to 4D:

```
/Applications/4D.app/Contents/MacOS/4D/ --headless --dataless --project
myApp.4DProject --user-param “{\”action\":\"BUILD\",\"value\":100}"

%HOMEDRIVE%%HOMEPATH%\Documents\4D\4D.exe --headless --dataless –project
myApp.4DProject --user-param “{\”action\":\"BUILD\",\"value\":100}"
```

Workflow for automating 4D builds

- We will need at least two runners, one macOS and one Windows, in that order (Windows Volume Desktop can be used on macOS to build and include Windows Client to masOS Server)
- In each runner, we will checkout our repository
- Our repository will have folder with scripts and other files needed to automate build process. There is parameters.json file in that folder that we use to decide: what to build, final application name, from where to download 4D and where to upload built files, signing certificate information, …

- We use different parameters.json file depending of the branch workflow is executing in: if we are running in develop branch we can build different files than if we run in release branch, we can upload to different location or even use another 4D binaries (R version instead of LTS) for building
- We have to download 4D and developer licenses in each runner (we are using 4D in headless mode for compiling and building)
- Depending of the build action we defined in parameters.json file we will need to download 4D Volume Desktop and/or 4D Server

- Then we open 4D in headless mode passing parameters.json file to 4D
- In 4D, we check if we are running in headless mode, if we do, we will always compile project
- If compilation fails, we will write status.log file in certain folder so our runner scripts can know that we failed and pass that information to next (Windows) runner so it will not run if compilation was unsuccessful
- If compilation was successful, our 4D code will set build settings XML file according to build actions we defined in parameters.json file

- We will call BUILD APPLICATION command passing the path to the XML file we created as parameter
- If this fails, we will also create status.log file indicating that our runner didn't finish the job properly
- At this point, 4D will quit.
- If we were successful, post build scripts will run and upload build files to SFTP server (we used to make GitHub release also)
- https://github.com/madamov/appBuilder

To do

- Integrate signing and notarizing dmg archives using DMG Canvas
- Adding and removing components and plugins from build settings file
- Support uploads to B2, AWS, Google drive
- Use subfolder with current build number in Documents folder to store everything from that build (artifacts, built files, logs …). This will be kind of history when using self-hosted runners.