

# Digital Preservation at Oxford and Cambridge

A collaborative research project to evaluate and provide sustainable recommendations for our digital preservation programmes

## How I got JHOVE running in a debugger

Posted on **9 August, 2018** by **Dave Gerrard**

*Cambridge's Technical Fellow, Dave, steps through how he got JHOVE running in a debugger, including the various troubleshooting steps. As for what he found when he got under the skin of JHOVE—stay tuned.*

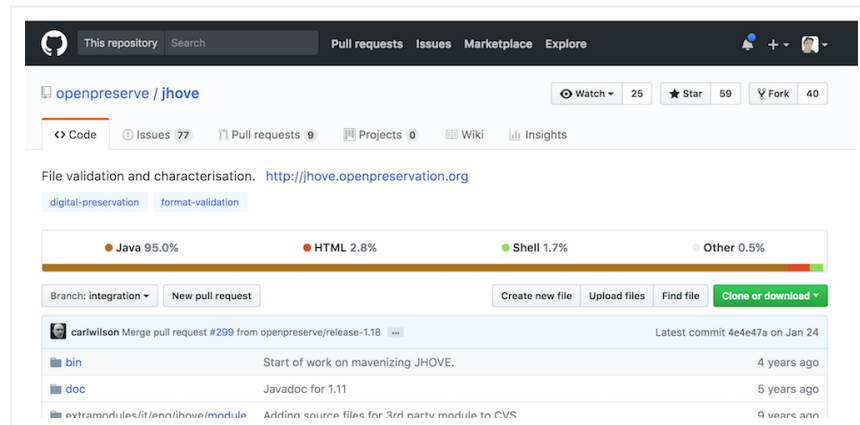
Over the years of developing apps, I have come to rely upon the tools of the trade; so rather than read programming documentation, I prefer getting code running under a debugger and stepping through it, to let it show me what an app does. In my defence, Object Oriented code tends to get quite complicated, with various methods of one class calling unexpected methods of another... To avoid this, you can use [Design Patterns](#) and write [Clean Code](#), but it's also very useful to let the debugger show you the path through the code, too.

This was the approach I took when I took a closer look at [JHOVE](#). I wanted to look under the hood of this application to help James with [validating a major collection of TIFFs](#) for a digitisation project by Bodleian Libraries and The Vatican Library.

### Step 1: Getting the JHOVE code into an IDE

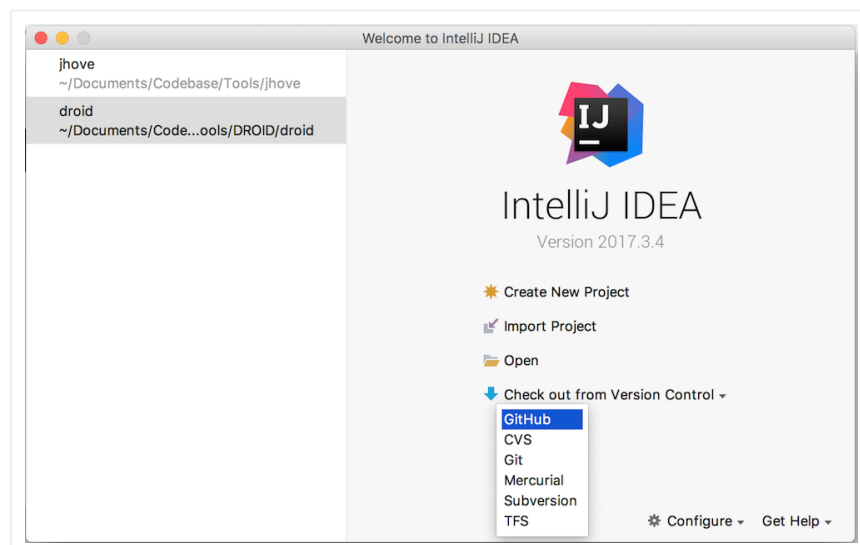
**Jargon alert:** ‘IDE’ – stands for ‘Integrated Development Environment’, which means: “... piece of software for writing, managing, sharing, testing and (in this instance) *debugging* code”.

So I had to pick the correct IDE to use... I already knew that JHOVE was a Java app: the fact it’s compiled as a Java Archive (JAR) was the giveaway, though if I’d needed confirmation, checking the coloured bar on the [homepage of its GitHub repository](#) would have told me, too.



— Coding language analysis in a GitHub project

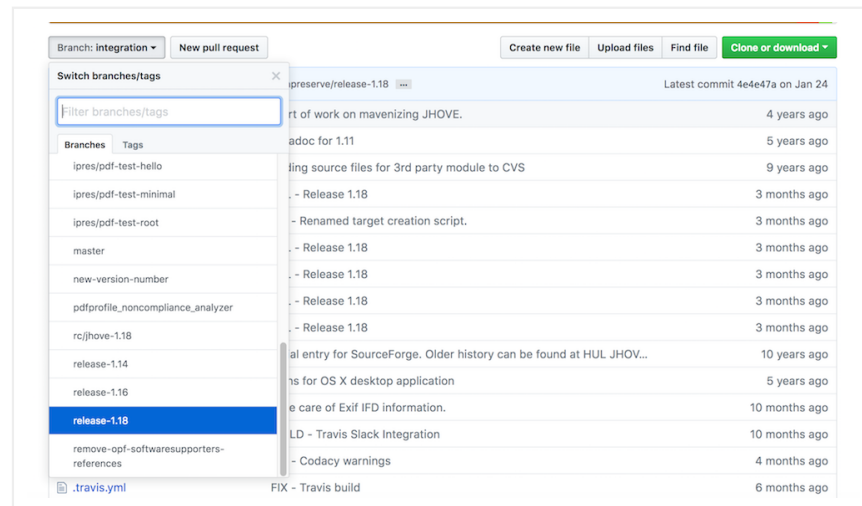
My Java IDE of choice is [JetBrains’s IntelliJ IDEA](#), so the easiest way to get the code was to start a new project by *Checking Out from Version Control*, selecting the GitHub option and adding the URL for the JHOVE project (<https://github.com/openpreserve/JHOVE>). This copied (or ‘cloned’) all the code to my local machine.



— Loading a project into IntelliJ IDEA directly from GitHub

GitHub makes it quite easy to manage code *branches*, i.e.: different versions of the codebase that can be developed in parallel with each other – so you can, say, fix a bug and re-release the app quickly in one branch, while taking longer to add a new feature in another.

The [Open Preservation Foundation](#) (who manage JHOVE's codebase now) have (more or less) followed a convention of 'branching on release' – so you can easily debug the specific version you're running in production by switching to the relevant branch... (...though version 1.7 seems to be missing a branch?) It's usually easy to switch branches within your IDE – doing so simply pulls the code from the different branch down and loads it into your IDE, and your local Git repository in the background.



— Finding the correct code branch in GitHub. Where's 1.7 gone?

## Step 2: Finding the right starting point for the debugger

Like a lot of apps that have been around for a while, JHOVE's codebase is quite large, and it's therefore not immediately obvious where the 'starting point' is. At least, it isn't obvious if you don't **READ** the [README file in the codebase's root](#). Once you finally get around to doing that, there's a clue buried quite near the bottom in the Project Structure section:

*JHOVE-apps: The **JHOVE-apps module contains the command-line and GUI application code** and builds a fat JAR containing the entire Java application.*

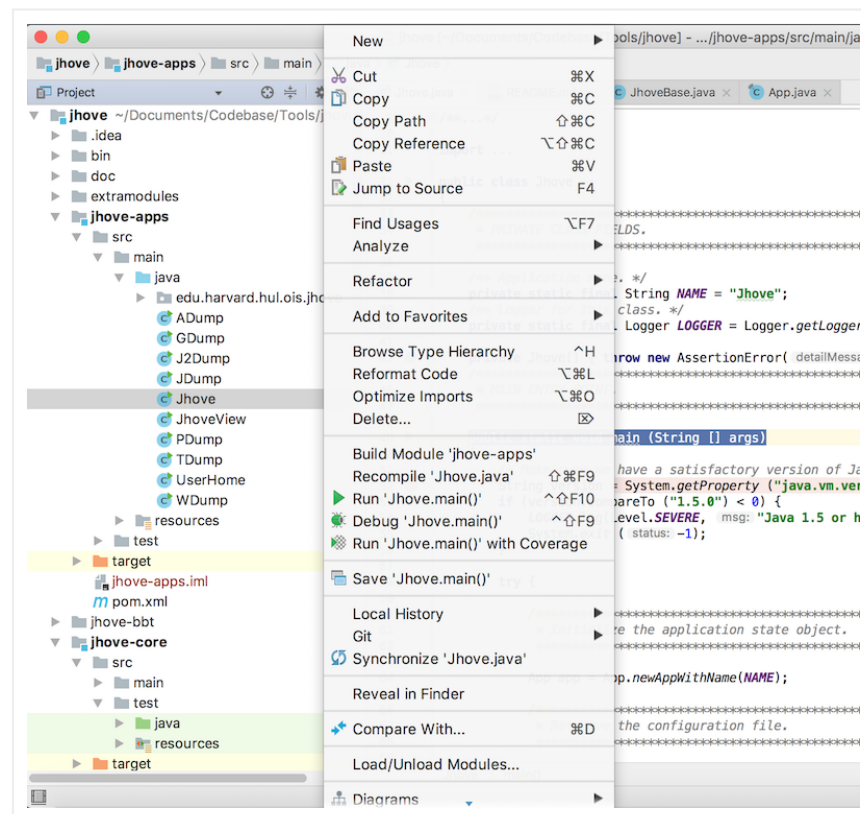
... so the app starts from within the jhove-apps folder somewhere. A little extra sniffing about and I found a class file in the src/main/java

folder called Jhove.java, which contained the magic Java method:

```
public static void main (String [] args) {}
```

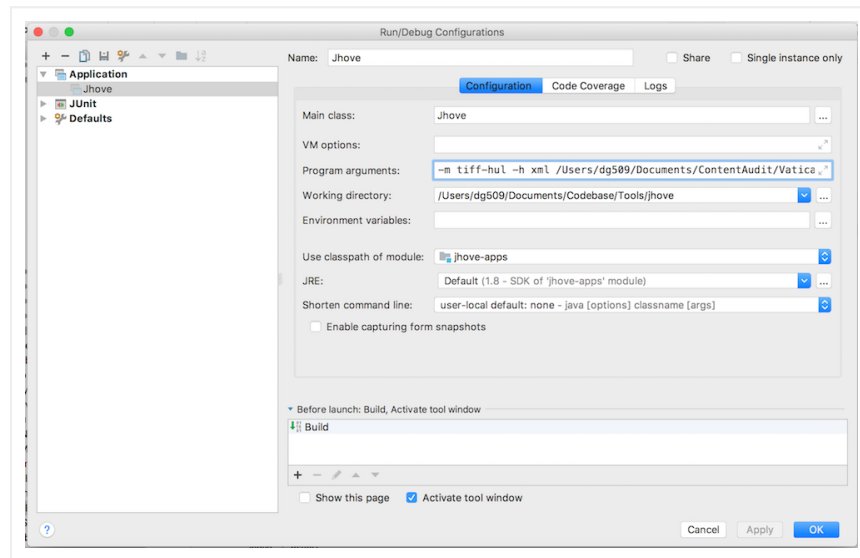
...which is the standard start point for any Java app (and several other languages too).

However, getting the debugger running successfully wasn't just a case of finding the right entry point and clicking 'run' – I also had to setup the debugger configuration to pass the correct command-line arguments to the application, or it fell at the first hurdle. This is achieved in IntelliJ IDEA by editing the Run / Debug configuration. I set this up initially by right-clicking on the Jhove.java file and selecting *Run JHOVE.main()*.



— Running the Jhove class to start the application

The run failed (because I hadn't added the command line arguments) but at least IntelliJ was clever enough to setup a new Run / Debug configuration (called Jhove after the class I'd run) that I could then add the Program Arguments to – in this case, the same [command line arguments you'd run JHOVE with normally](#) (e.g. the module you want to run, the handler you'd want to output the result with, the file you want to characterise etc etc).



### — Editing the Run configuration in IntelliJ

I could then add a breakpoint to the code in the `Jhove.main()` method and off I went... Or did I?

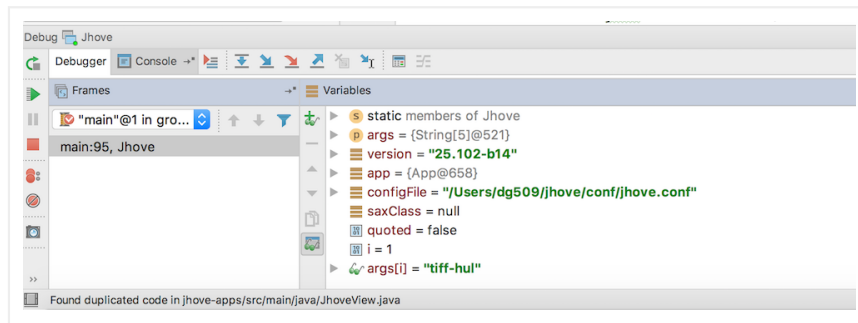
### Step 3: setting up a config file

So this gave me what I needed to start stepping through the code. Unfortunately, my first attempt didn't get any further than the initial `Jhove.main()` method... It got all the way through, but then the following error occurred:

```
Cannot instantiate module: com.mcgath.jhove.modul
```



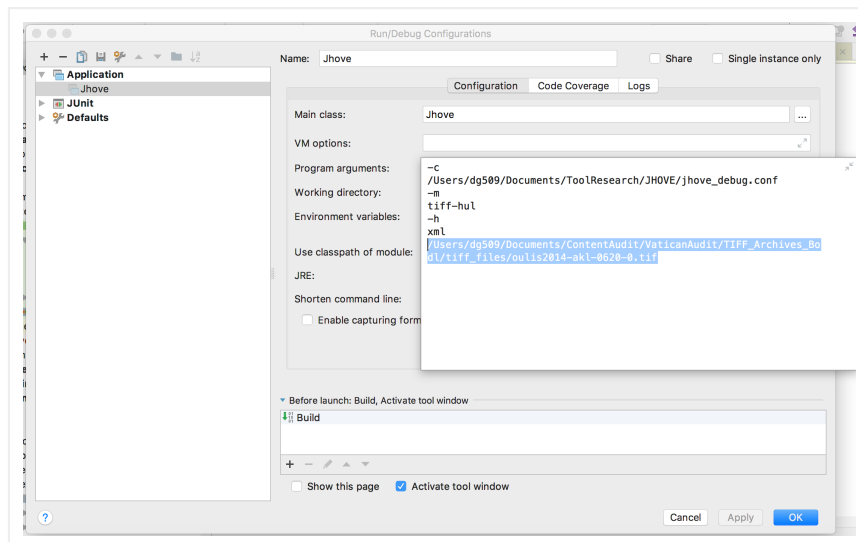
The clue for how to fix this was actually provided by the debugger as it ran, however, and provides a good example of the kind of insight you get from running code in debug mode in your IDE. Because the initial set of command-line parameters I was passing in from the Run / Debug configuration didn't contain a "-c" parameter to set a config file, JHOVE was automatically picking up its configuration from a default location: i.e. the JHOVE/config folder in my user directory – which existed, with a config file, because I'd also installed JHOVE on my machine the easy way beforehand...)



— Debugger points towards the config file mix-up

A quick look at this config showed that JHOVE was expecting all sorts of modules to be available to load, one of which was the ‘external’ module for PNG characterisation mentioned in the error message. This is included in the JHOVE codebase, but in a separate folder (jhove-ext-modules): the build script that pulls JHOVE together for production deployment clearly copes with copying the PNG module from this location to the correct place, but the IDE couldn’t find it when debugging.

So the solution? Put a custom config file in place, and remove the parts that referenced the PNG module. This worked a treat, and allowed me to track the code execution all the way through for a test TIFF file.



— Adding an extra -c config file parameter and a custom config file.

## Conclusion

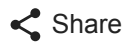
Really, all the above, while making it possible to get under the skin of JHOVE, is just the start. Another blog post may follow regarding what I actually found when I ran through its processes and started

to get an idea of how it worked (though as a bit of a spoiler, it wasn't exactly *pretty*)...

But, given that JHOVE is more or less ubiquitous in digital preservation (i.e. all the major vended solutions wrap it up in their ingest processes in one way or another), hopefully more people will be encouraged to dive into it and learn how it works in more detail. (I guess you could just 'read the manual' – but if you're a developer, doing it this way is more insightful, and more fun, too).

---

#### SHARE THIS:



This entry was posted in [patterns](#), [technology](#), [tools](#) and tagged [debugging](#), [jhove](#), [tool](#) by [Dave Gerrard](#). Bookmark the [permalink \[http://www.dpoc.ac.uk/2018/08/09/jhove-running-in-a-debugger/\]](http://www.dpoc.ac.uk/2018/08/09/jhove-running-in-a-debugger/).

ONE THOUGHT ON “HOW I GOT JHOVE RUNNING IN A DEBUGGER”

markus schnalke

on [18 September, 2018 at 16:00](#) said:

Thanks for this blog post. It is a kind of text that I'd like to see more often! Such “how to take a look under the skin of the stuff that you deal with all the time” are really worthwhile.



Keep going!

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)