Casey Blair

Team Member Net ID's: cdblar2

CS 410

Final Project

Fix Broken Metapy Installation in Python 3.8 Environments

My final project is to update the Metapy GitHub repo to allow for installation with Python 3.8. Currently, when trying to install Metapy using Pip on Windows 10, Mac OSX, and Linux if using a Python 3.8 or higher environment, the install setup Python wheel fails to install the Metapy package due to errors. However, if using a Python 3.7 - 2.7 environment the Metapy library will install correctly. Since It is vital to keep this library functioning with the latest version of Python to allow this library to be used in production code bases.

I have been researching Pybind11 and how Python can be used with C and C++ code to understand how the Metapy Git repo works. Metapy is a Python wrapper for the Meta Github repo (https://github.com/meta-toolkit/meta). The Meta code base is written in C++, and when the Metapy package is installed with Pip, the Meta C++ code is downloaded and installed then a Python interface is constructed so the code can be used from a Python interpreter.

When installing Metapy in a Python 3.7 - 2.7 environment, the Pip installer downloads a tar.gz file from the Github repo's Releases page with the Metapy code already built for that specific Python version's environment, which is why it can be installed correctly with a Python 3.7 - 2.7 environments. The maintainers of the repository stopped making new releases after they published the Python 3.7 version in August 2018. If there is not a pre-built Tar file for the specific Python version, then Pip will use Python's installation wheel system to manually build the version on the client's computer to do the installation of Metapy.

The Metapy repository is a python wrapper around the Meta library (https://github.com/meta-toolkit/meta). This Meta project's code is written in C++. The PyBind11 library allows for a Python interface to be constructed between C++ code and Python to allow for C++ code to be used in a Python interpreter.

I next researched the PyBind11 Python package and learned how C++ code could be used from a Python interpreter as I have never written a Python program that used C++ code before so I needed to understand this system to understand how Metapy worked. At this point I was unsure if the C++ code was failing because the C++ code needed to be updated to be compatible with a new version of the C++ compiler, or maybe a dependency in the C++ code had changed unexpectedly which was causing the issue. I spent 3-4 hours reading the PyBind documentation so I understood how this library was linking the C++ code from the Meta submodule to the Metapy python code.

I also didn't know if Metapy simply needed to be updated to be Python 3.8 compatible. When I first used this library, my Mac had 0Python 3.8 installed as well as Python 2.7 and since a release compatible with Python 3.8 has not been released it caused the setup.py script to be used instead of the just downloading the prebuilt Tar file. Because the C++ files in the Meta file needed to be compiled using the CMake files I also brushed up on C make files as it has been years since I've written one.

I then downloaded the Metapy code and ran the manual build steps outlined here: (https://github.com/meta-toolkit/metapy#getting-started-the-hard-way). I ran in to many issues trying to get my C++ compiler to compile the Meta. I also had to brush up on C++ make files as the Meta repo's C++ build steps were failing, so I needed to figure out how the system was being built and it required me to refresh my knowledge of C++ make files as I have not written one in over 5 years.

After this initial research, which took 7-8 hours, I started stepping though the C++ compiler errors one at a time. The problem with debugging C++ compiler errors is that when the compiler encounters the first error, it exits the compilation. So the developer doesn't know how many more errors there may be until after the fix the previous error: there can only be one error encountered at a time since the compiler exits the compilation after the first encountered error. So I knew I needed to fix all the errors the C++ compiler encountered here to allow the system to build correctly for Python 3.8.

I attempted to install the Metapy library in my Python 3.8 environment using the following command (which failed):

```
sudo python3.8 -m pip install metapy
```

The first error I ran in to was due to a missing C header file on my operating system. This was to be expected as I just created this virtual machine with a clean install of Ubuntu and it did not have python 3.8 installed on it by default, so I had installed Python 3.8 but forgot to install some development tools associated with python. I needed to install the Debian package python3.8-dev with the following command:

```
Sudo apt-get install python3.8-dev
```

To install the missing Python.h header file that the C++ compiler was throwing an error because this file was misssing from my operating system. I had to research the error being thrown to figure out that the python3.8-dev package was needed to fix the missing Python.h file error being thrown by the Metapy install wheel.

After fixing the Python.h missing file issue, rerunning the Metapy's installation command would fail with the following error repeating five times before exiting the installation:

```
-- Downloading...
  dst='/Users/caseyblair/Documents/Classes/CS_410/metapy/deps/meta/deps/icu-61.1/icu4c-61_1-src.tgz'
  timeout='none'
-- Using src='http://download.icu-project.org/files/icu4c/61.1/icu4c-61_1-src.tgz'
-- [download 100% complete]
* Closing connection 2
* Closing connection 0
* Closing connection 1
-- verifying file...
     file='/Users/caseyblair/Documents/Classes/CS_410/metapy/deps/meta/deps/icu-61.1/icu4c-61_1-src.tgz'
-- MD5 hash of
```

```
  /Users/caseyblair/Documents/Classes/CS_410/metapy/deps/meta/deps/icu-61.1/icu4c-61_1-src.tgz
does not match expected value
  expected: 'fac212b32b7ec7ab007a12dff1f3aea1'
    actual: 'ac2ff460bdda9c70e6ed803f5e4d03fb'
-- Hash mismatch, removing...
-- Retrying...
```

It was very strange that this ICU Tar file download was producing a different MD5 hash after each attempt to download the Tar file. I decided to investigate this further as if the Tar file had been updated, but the expected MD5 hash had not been, then at least the same incorrect MD5 hash should be displaying each time. However, the actual MD5 hash was a different hash each time, which made me suspect a different error was occurring.

I decided to attempt to visit the URL the C++ code was attempting to download the dependency from (http://download.icu-project.org/files/icu4c/61.1/icu4c-61_1-src.tgz). After pasting this URL in to my web browser, instead of a TGZ file download, my browser took me to an HTML page listing all the releases of this ICU dependency. Puzzled why this URL was not a download link anymore, I navigated around this website until I found an answer.

After going to the "Source Repository" page, at the top of this page I found my answer (http://site.icu-project.org/repository http://site.icu-project.org/repository). The code repository that used to host this ICU dependency was migrated from Subversion to GitHub. Therefore, the previous download link would not work anymore, and all pre-built TGZ files were now on GitHub.

I then returned to the ICU version release page and found my version 61 listed and clicked on that link. I was redirected to a page listing the change log for this new version and found the link to the Git repo where the code now resides: https://github.com/unicode-org/icu/releases/tag/release-61-1. I navigated to the GitHub repo to find many pre-built TGZ files.

I had to step though the setup.py file's build steps and figure out what exactly the build steps were ad where this ICU package was being downloaded at in the code. The setup.py file from the Metapy repo is actually running a C++ make file that downloads the Meta git repo (https://github.com/meta-toolkit/meta), then runs a Makefile to compile the C++ code for the host machine. The icu4c-61_1 -src.tgz file download is performed in the Meta C++ Makefile system, so I need to figure out how to update this file download in this second Git repo. It took me a while to search though all the C++ code in the Meta repo to find the correct location where the ICU package was being downloaded.

I finally found the line in the Meta repo where the ICU package was downloaded. I tried each of the TGZ files until once successfully downloaded and the C++ compiler progressed past that point of the installation. Each attempt took 20 minutes as the C++ compiler is quite slow, so this was a manual process to debug.

At this point, I was not sure how many other issues I would encounter. I didn't know if I would need to fix many more of these external dependency url's that were broken, or if this was the only one. Fortunately, this was the only broken URL and the rest of the compilation succeeded. I successfully fixed the Python 3.8 build system as the C++ compiler was able to compile the Meta repo successfully, then install the Metapy library using Pip.

I did have to make forks of the Metapy and Meta repos in GitHub as my GitHub account does not have write access to the main Metapy or Meta repos. I tried to make a new branch on the Metapy and Meta repos to update the code, however, I was met with 403 unauthorized errors each time. So I needed to fork the repos and make my own version, then make merge requests from my forked repo back to the parent repo.

I currently have three open GitHub pull requests to fix this issue in the original parent repos: https://github.com/meta-toolkit/meta/pull/222, https://github.com/meta-toolkit/meta/pull/220, and https://github.com/meta-toolkit/metapy/pull/10. All three pull requests are currently waiting review by the code repo's members so I have no idea when or if they will approve these changes as none of them have made an update to the repo since August 2018. However, you can currently use my forked version of Metapy to correctly install Metapy for Python 3.8 with the following command:

```
sudo python3.8 -m pip install git+https://github.com/ccasey645/metapy.git
```

This command, on Ubuntu 18.04, will build and install the Metapy library correctly. This command will instead download my forked version of Metapy from my GitHub repo and the installation succeeds.

I did not have time to attempt the build on OSX and Windows 10 too as spent well over 20 hours figuring out the code changes needed to fix the C++ compiler errors. However, when I was attempting to run this on OSX and Windows 10 my C++ compiler installations on these OS's were in a very broken state: Windows 10 needed some new version of Visual Studio C++ library installed that would break other projects I was working on, and my OSX's clang compiler was throwing errors saying that it was not allowed to compile C files using a C++ compiler so I decided to focus just on the code changes needed to fix the errors and leave the OSX and Windows 10 TGZ building to the Metapy code repo owners. When the Metapy code repo owners accept my GitHub pull requests, there is a make-release.sh script in the Metapy repo that will build the appropriate Tar files for all three platforms, so each client computer will not have to have a 100% configured C++ compiler just to install the Metapy library.