Neon AI SDK Installation

Today we're going to look at creating our first Neon AI skill using Adapt. Before you continue here, if you haven't already, make sure you have your Neon SDK installed and have PyCharm installed as well. I'll link the two videos for those below. Let's get started.

Skill Intro
Okay, so we're going to make a new skill for Neon today and we're just going to make a launcher skill. So, a skill where you can say 'launch a program', and it will launch that program.

AVMusic Skill Example
To start we're going to look at AV Music here. So, this is just an example of a skill we have that uses the Adapt intent builder and if we scroll down, there are some aspects of every skill that look very much the same. They're all going to be a subset of Mycroft skill and we're going to have this init function and also I'm going to have this initialize function where we register our intents.

Skill File Structure
So, if we look over in our folder structure, I've already made a launcher skill outline, kind of. We have the folder for the skill, our dialog and vocab sub-directories, both with US English is the only language in there, and then we have our init file.

Writing our Skill Intent If I pull that up, it's so far empty, so we're going to fill that in. Since this is a launcher skill, we need to be able to launch programs; to do that we're going to use subprocess, which is a module that basically lets you run a command as you would run it in a terminal. We're also going to use Adapt as I mentioned previously, so we're going to need to import the intent builder there. This is a skill so we need to import the parent skill; so from mycroft.skills.core import mycroft skill, and then lastly, I always import our log, especially when you're creating something new or making changes. It's good to put log statements around things to see where something failed if it fails. Alright so we're going to do class launcher skill, and that is a Mycroft skill, and then our init. We're not initializing any variables so we are only going to have this one line under our init and this is just following convention of mycroft skills. And then we're going to make our initialize function, and this is where we register our intents. All right, so let's look at take a look at what I just typed up here. So, we're defining this launch chromium intent using the intent builder we imported from Adapt. We're naming that intent launch chromium intent, and we told it it's going to require a launch keyword and it's going to require a chromium keyword, and then we told it to build. So this is a pretty common formula you'll see, we'll get into that with some variations a little bit later. So now what we need to do is we need to actually register that intent to something. So we need to tell it when it matches what it's going to do. And this is all just following convention, this self dot launch chromium intent is a function we're going to

write here in just a moment. It's just convention that this defined name matches this defined name, so it's not like we're calling something recursively. They just match the names for convention's sake.

Writing our Skill Function

And we're going to pass it this message parameter. Message is an object that contains things like what you actually said and what these key words are that it found. So, for debugging, I always like to just log message.data. This can help if you are trying to pull a lot of key words out of something, or you just want to see what keywords you found; some other stuff like that. It's always good to put a self.speak under any intent match. That just means that Neon will say something when it matches. You could leave this out, and just have it call up a program, but it makes a lot more sense to have it say something, so you know why that program launched for example. I'm going to use sub-process here to launch chromium and then you give it a parameter of a website to go to. And then if we look here, we have a warning that we're not implementing all of our abstract methods.

'stop' Abstract Method

I'm just going to pop over to the Mycroft skill, and I know which method this is, so I'm just going to search for it. So, stop is an abstract method that every skill should implement. From this particular skill, we don't have any processes that we would need to explicitly stop, so it's just going to pass. But say you had something like AV Music, if I go back to our example there. It has stop and what it has to do is it has to for example disable the pause intent, because if it's not playing anything, there's nothing to pause and it also has to kill the process– so the– in this case, MPV, which is playing the audio and video.

Finishing Up Our Skill init

For this skill, we just want it to launch the program. We don't want it to track it, we don't want it to have to close it at a later time; it's just going to launch it and then be done. And then the last part we have to add here is our create skill method, and all that's going to do is return our class. And that is our completed skill init, now I'm going to look at our keywords here. We have a launch keyword and we have a chromium keyword, so those need to be added.

Creating .voc Files

If we go over to our vocab directory and then we made our en-us. I'm going to make a new file here, and this is going to be the same string I used here in require, with the file extension .voc. We'll add that and then this file, basically one per line, we're going to add what words mean launch. So for this example I'm going to say launch and I'm going to say open. So, either launch or open will fulfill this requirement and then we're going to do the same thing for chromium. Add that to our git, and for chromium I'll just say chromium, also Chrome, or browser; and I think that makes sense. So as long as we have one word from our launch keyword and one word from our chromium keyword, this function will be called. So now I'm going to commit these changes, and we'll give it a test.

Testing our Skill Okay, so we have pulled our changes on our test machine here, so now we're going to test our new skill just by saying launch Chrome. (launching Chrome) And it works. (welcome to neongecko.com, thanks for being a developer. While you're here, make sure to check out some of our information on our neon SDK or maybe check out our Klat adaptive forums, which also integrate our neon AI.)

Adding Functionality (More Programs)
So, now that we know our skill works, we're going to make some additions to make it a little more useful. Instead of just launching Chrome we're going to add some options for other programs. So if I want to add our file explorer as an option, I'm going to need to make a new keyword. So I'll add NautilusKeyword.voc. Nautilus is Ubuntu's file explorer. I'll go ahead and add nautilus, files, file explorer. And now for our intent, we're going to modify this. We don't want to require the chromium keyword now, since we're adding another option, so we're going to change this to optionally and then we're going to add also optionally Nautilus.

Modifying our Launcher Function
So, we have those changes, now we need to change our intent, and this isn't really chromium anymore, so I'm going to refactor this and just name it launch program intent. And same with our function. Perfect. So, that looks better and then we'll add this if message.data.get ChromiumKeyword, then we'll launch Chrome. I'm just going to copy this and just change this to Nautilus, and now we have the same thing for our file browser. I'm going to add one more for our terminal in the same way. We'll add another optionally here and we'll do another copy and paste. So that's going to be an else if.

Using Dialog Files for Speech
So now what we're going to do, is we're going to change the way we handle the speaking. Since these are all very similar we're basically just going to say launching, and then the name of the program we're launching. We can use a dialog file to make that a little simpler. So to do that, instead of speaking for each one, what I'm going to do is I'm going to make this variable program. I'm going to start that out as None and then, depending on the program we match, I'm going to assign a value. And then at the very end of this, I'm going to say if program self.speak dialog launch program and feed it this dictionary that has key program, with value whatever this variable evaluates to. And then I'm going to put in this else statement let's get self.peak I don't know that program.

Analyzing Our Skill Logic
So let's look at how this will work. Our intent builder is going to require launch that's the only requirement here, so if it hears the word launch or open, the two that we put in our vocab file, then its going to call this intent. Once it gets here, it's going to look for chromium or chromium key word in the message. If it doesn't find that it'll look for a nautilus one. If it doesn't find that in to look for terminal. If it doesn't find that, it's still going to continue here. But since we started program out as none, unless it matches one of these three, program

is going to have a value of none, so we'll get this message of "I don't know that program." If it does match one of these, then it's going to speak this dialogue, and we need to tell it what this dialogue is.

Adding Another Dialog File

So, if we go back over to our dialogue directory, and under language. And we name it with this string we used here .dialog. text, and to our git. And then I'm just going to type in launching {{program}}. So, any of these values in these double curly braces, it's going to look for in the dictionary we pass it.

Running More Tests

So I'm going to minimize this, and I have Neon running here I just have the mic muted here, so we're going to unmute the mic and give it a test. Launch Chrome. (launching Chrome) And we get exactly what we expect. So if we want to do some further testing, since we have skip Wake words enabled here ,it should be transcribing everything I'm saying. So if I were to without pausing say launch Chrome it should still launch Chrome. And it did. And another important part of testing here is to test things that we wouldn't expect to match to make sure we get an expected result. So, an example would be if I were to just say launch. And it were to transcribe correctly as launch. (I don't know that program). We get the response we expected, because it went through that else where program was none and it just said it doesn't know that program. And then the other case that we talked about, if I were to say launch chromium and File Explorer (launching Chrome) it's only going to match the first one in that statement which means the order I said it didn't really matter. I could just as easily say launch file explorer and chromium, and it could transcribe correctly. Launch file explorer and chromium. (launching Chrome) and it still launches chrome because that's the first one it looks for.

Handling Imperfect Transcriptions

But this brings up another point of Google transcription isn't perfect, and launch and lunch seem to be fairly similar words it's going to match. So what I can do to improve our success rate with the skill is to go back to our vocab file and I can just add lunch as one of our words there, and this may not make a lot of sense looking at it, but when you look at some of the quirks of the transcriptions, this is a useful way to make it a little more reliable.

Testing with Wake Words Enabled

So with that testing done, I'm just going to quickly turn wake words back on and make sure we get expected behavior in that mode. We didn't do anything in the skill that was explicitly dependent on wake words, so this shouldn't make a difference but it's still good to test.

Hey Neon, launch file explorer (launching file explorer)

Hey Neon, launch terminal (launching terminal)

Hey Neon, launch. (I don't know that program)

And we get the same behavior that we would expect, matching what happened when we were skipping wake words.

Thanks for watching this Neon AI tutorial. Be sure to head over to neongecko.com for more information, including a written transcript of this tutorial and any code snippets we may have used. See you next time.