New Skill 3: Regular Expressions

Today, we're going to continue looking at our launcher skill. Specifically, we're going to look at moving from using the Adapt intent engine to using Padatious. Let's get started.

Adapt vs Padatious

Before we start modifying our skill to use Padatiouis, let's look at how our two different intent parsers differ. So right now, for Adapt, we register this intent with the required launch keyword, required program, and optionally Neon. So, that's a pretty simple formula, but it's pretty strict about– we need one of these keywords and we need something in this position of a very specifically structured sentence.

I Like Brands Skill Example

So I'm going to take a look at our "I like brands" skill because this one uses Padatious and we'll look at how the initialize function is very different. So instead of having a brand intent, we're registering these entity files, and those are things like: type of brands, like keyword, and brand name. And then we register an intent file. So, to show an example, our brand name entity file is just a list of brand names, so any of these lines will match a brand name. And then if we look at one of our intent files, it just has the structure "I", one of our like entities, and one of our brand name entities. So that's pretty flexible and that's the other benefit of using Padatious, is it works on a confidence level. So it's not looking for explicitly a match of "I like brand", it's looking for something that is similar enough to that that it's confident it's a match. That confidence level can be adjusted, but conceptually, it's just more flexible than a sentence with the specific syntax we need to match.

Modifying Our Skill Intents

So our initialize function, we're not going to register an Adapt intent using the intent builder; we're actually not going to use that at all. So I'll remove that import, and I'm going to comment all of this out, and instead what we're going to do is register an entity file and an intent file. So, we're going to make a program.entity and like the brands.entity we looked at, this one's just going to have a list of the programs that we want to be able to launch. And then same with the intent file, and that's going to call our launch program intent in the same way we did before. If we go over here to our vocab, we're actually not going to need either of the launch keywords or the Neon one anymore. Just delete those and we'll make a new one launch.intent and then a new program.entity.

Creating Entity and Intent Files

So program, we're gonna have everything that we had in the other .voc files; all of our keywords for Chromium, Nautilus, Terminal, and Text Editor. These are the same options that are in these lists in our init. I'm going to leave these here just so we can continue to use multiple words for each program. You could simplify this and just get rid of these and just say: if Chrome, if Nautilus if

terminal, if text editor, etc but we're gooing to leave that as it is. And then we're going to go to our intent file and using the same options we used before, I'm going to say launch, or lunch, or open and then one of the items out of our program entity file. So that's that.

Modifying Our Skill Function

We're not going to be able to use the same check for signal or look for Neon; I'm just gonna remove that. Because we're looking for a specific item out of our list, we shouldn't get as many of those false positives anyways. And then that also means we should never get this not found, I'm just going to remove that, which means I can actually remove all of this if match logic because with the way the intent file is, it's not going to match if it didn't find a match in the first place, so we'll never get to here. Clean up our indentations.

Comparing Regex to Padatious Intents

So, before I continue on to testing this skill, I do want to point out the differences here between our regex file we were using previously, and the intent file we're using now. So these look very similar, the main difference here is with our intent, program is an entity, not just some variable. So we're specifically looking for these words that are in our entity file in order for that to match.

Testing Our Modified Skill

So with that said, let's do some quick testing. I have in this terminal here, you can see I'm just following our logs and looking for "launcher", it's the name of our skill, so anything that matches any of our log statements in the skill will get picked up here. Also anytime I say that word, it'll notice. So I'm just going to type in something here. So far this is working exactly as it did before, but let me try something different. So, I just asked it to launch Libre office, and we never told it what that was. If you remember before when we had this, it still matched the skill and it said it didn't know what LibreOffice is. But if you look at our logs it never called the skill, and that's because it LibreOffice isn't in the program entity file. So this kind of helps; you're not going to get calls to skills that don't belong as much this way, because it is looking for– before it even gets to the skill intent– it's looking for all of the keywords it needs to execute.

Comments on Adapt vs. Padatious

I just have a couple of extra comments to make here about using Padatious. Generally, Adapt intents are going to be tried before Padatious, and if we look here in our intent service file. This is in mycroft/skills and then inside of handle_utterance. This is where it handles the intent, and we'll see next I have a comment here that says exactly what I'm trying to say. It's going to send the message to adapt unless Padatious is really sure that it should be handling it, and this confidence level is set at 0.8. If you have issues with skills that rren't matching to Padatious, you can try decreasing this. Or if skills are matching that shouldn't, you can increase it, but it's recommended to leave this the same and modify your intent files if it becomes an issue. 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.