**Trapping out Alexa skill timeout**

****

**Problem:**

Your skill has only 8 seconds to do its job. It then times out and closes the skill. What can you do if your skill sometimes takes longer than 8 seconds?

Here we’ll discuss how to use Python asyncio to trap the timeout and return to your skill.

Currently ASK uses Python version 3.9, so we’ll use **asyncio.wait\_for()**

<https://docs.python.org/3.9/library/asyncio-task.html#timeouts>

This is their example:

async def eternity():

# Sleep for one hour

await asyncio.sleep(3600)

print('yay!')

async def main():

# Wait for at most 1 second

try:

await asyncio.wait\_for(eternity(), timeout=1.0)

except asyncio.TimeoutError:

print('timeout!')

asyncio.run(main())

# Expected output:

#

# timeout!

In Python 3.11 you will be able to use asyncio.timeout(). Example:

async def main():

try:

async with asyncio.timeout(10):

await long\_running\_task()

except TimeoutError:

print("The long operation timed out, but we've handled it.")

print("This statement will run regardless.")

Note asyncio functions can only be called using asyncio,run(). You can’t call them directly.

**Example:**

Start a new Alexa Hosted skill (I’ve called it ‘long timeout’)

Check your Skill invocation name (from “change me”) if necessary.

Save and build the model.

We’ll modify the HelloWorld intent to call the tasks

At the top of the code add the imports:

import time

import asyncio

import random

Start by adding a random long task before class LaunchRequestHandler. This will wait 0,1,2 or 3 seconds

async def randomTime():

timeout = True

randomSecs = random.randint(0, 3) # 0,1,2 or 3 seconds

await asyncio.sleep(randomSecs)

timeout = False # executed if sleep finishes

return timeout

We’ll call this with a 2 second time out. Sometimes we will complete the (sleep) task and other we will timeout. We pass timeout back – True if we time out, False if the task completes.

We’ll call this with the **asyncio.wait\_for(),** passing the call and timeout (2 seconds)

async def waitForIt():

# Wait for at most 2 seconds

try:

timeout = True # True - we had a timeout. False - task (sleep) completed

timeout = await asyncio.wait\_for(randomTime(), timeout=2.0)

except asyncio.TimeoutError:

logger.info('timeout error')

return timeout

This has to be called (using asyncio.run()) from the HelloWorld intent. We’ll pass back timeout to see if we timed out (True) or False if we executed the task (sleep). We’ll use

timeout = asyncio.run(waitForIt())

Modify the HelloWorld intent as follows. I’ve also added **.ask** so that the skill doesn’t end and so we can continue to say hello:

class HelloWorldIntentHandler(AbstractRequestHandler):

"""Handler for Hello World Intent."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_intent\_name("HelloWorldIntent")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Hello World!"

timeout = False # True - we had a timeout, False - task (sleep) complete

# ADD LONG CALL

start = time.time()

timeout = asyncio.run(waitForIt())

end= time.time()

if timeout:

speak\_output = "We timed out. It took " + str(int(end-start)) + " seconds"

else:

speak\_output = "Task completed, we didn't timeout. It took " + str(int(end-start)) + " seconds"

return (

handler\_input.response\_builder

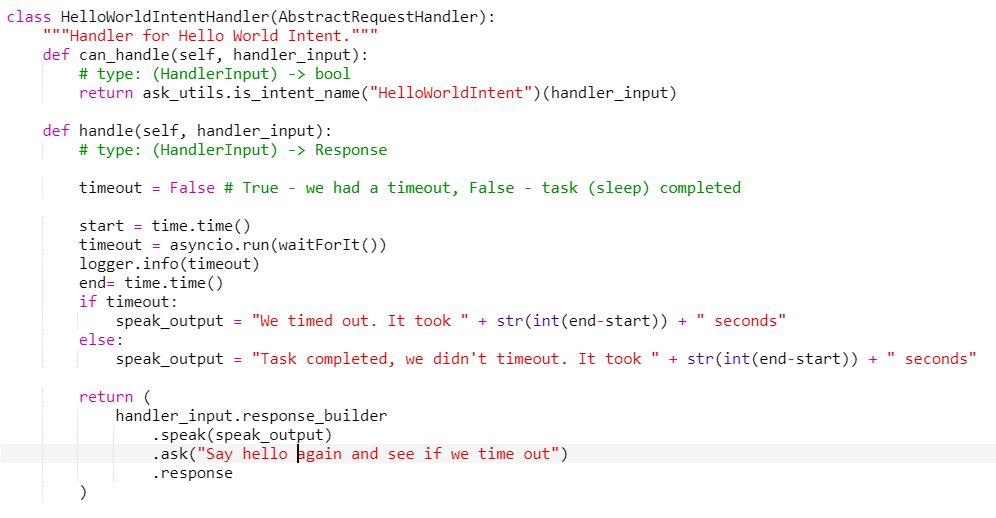
.speak(speak\_output)

.ask("Say hello again and see if we time out")

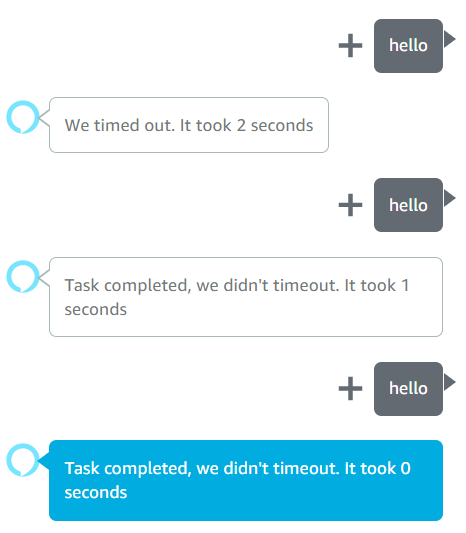
.response

)

I’ve also recorded how long the calls took.



Save, deploy and test your code (change to Development). Here’s my result:



You can see that sometimes the task takes less than a second and completes, and other times, times out.

You should now be able to trap out any long running tasks and return to your skill.

See: https://github.com/jallwork/AlexaTimeouts

Here’s the complete lambda\_function.py:

# -\*- coding: utf-8 -\*-

# This sample demonstrates handling intents from an Alexa skill using the Alexa Skills Kit SDK for Python.

# Please visit https://alexa.design/cookbook for additional examples on implementing slots, dialog management,

# session persistence, api calls, and more.

# This sample is built using the handler classes approach in skill builder.

# launch open long time out

import logging

import ask\_sdk\_core.utils as ask\_utils

import time

import asyncio

import random

from ask\_sdk\_core.skill\_builder import SkillBuilder

from ask\_sdk\_core.dispatch\_components import AbstractRequestHandler

from ask\_sdk\_core.dispatch\_components import AbstractExceptionHandler

from ask\_sdk\_core.handler\_input import HandlerInput

from ask\_sdk\_model import Response

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.INFO)

async def randomTime():

timeout = True

randomSecs = random.randint(0, 3) # 0,1,2 or 3 seconds

await asyncio.sleep(randomSecs)

timeout = False # excuted if sleep finishes

return timeout

async def waitForIt():

# Wait for at most 2 seconds

try:

timeout = True # True - we had a timeout. False - task (sleep) completed

timeout = await asyncio.wait\_for(randomTime(), timeout=2.0)

except asyncio.TimeoutError:

logger.info('timeout error')

return timeout

class LaunchRequestHandler(AbstractRequestHandler):

"""Handler for Skill Launch."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_request\_type("LaunchRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Welcome, you can say Hello or Help. Which would you like to try?"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class HelloWorldIntentHandler(AbstractRequestHandler):

"""Handler for Hello World Intent."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_intent\_name("HelloWorldIntent")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Hello World!"

timeout = False # True - we had a timeout, False - task (sleep) completed

# ADD LONG CALL

start = time.time()

timeout = asyncio.run(waitForIt())

logger.info(timeout)

end= time.time()

if timeout:

speak\_output = "We timed out. It took " + str(int(end-start)) + " seconds"

else:

speak\_output = "Task completed, we didn't timeout. It took " + str(int(end-start)) + " seconds"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask("Say hello again and see if we time out")

.response

)

class HelpIntentHandler(AbstractRequestHandler):

"""Handler for Help Intent."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_intent\_name("AMAZON.HelpIntent")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "You can say hello to me! How can I help?"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class CancelOrStopIntentHandler(AbstractRequestHandler):

"""Single handler for Cancel and Stop Intent."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return (ask\_utils.is\_intent\_name("AMAZON.CancelIntent")(handler\_input) or

ask\_utils.is\_intent\_name("AMAZON.StopIntent")(handler\_input))

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Goodbye!"

return (

handler\_input.response\_builder

.speak(speak\_output)

.response

)

class FallbackIntentHandler(AbstractRequestHandler):

"""Single handler for Fallback Intent."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_intent\_name("AMAZON.FallbackIntent")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In FallbackIntentHandler")

speech = "Hmm, I'm not sure. You can say Hello or Help. What would you like to do?"

reprompt = "I didn't catch that. What can I help you with?"

return handler\_input.response\_builder.speak(speech).ask(reprompt).response

class SessionEndedRequestHandler(AbstractRequestHandler):

"""Handler for Session End."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_request\_type("SessionEndedRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

# Any cleanup logic goes here.

return handler\_input.response\_builder.response

class IntentReflectorHandler(AbstractRequestHandler):

"""The intent reflector is used for interaction model testing and debugging.

It will simply repeat the intent the user said. You can create custom handlers

for your intents by defining them above, then also adding them to the request

handler chain below.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_request\_type("IntentRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

intent\_name = ask\_utils.get\_intent\_name(handler\_input)

speak\_output = "You just triggered " + intent\_name + "."

return (

handler\_input.response\_builder

.speak(speak\_output)

# .ask("add a reprompt if you want to keep the session open for the user to respond")

.response

)

class CatchAllExceptionHandler(AbstractExceptionHandler):

"""Generic error handling to capture any syntax or routing errors. If you receive an error

stating the request handler chain is not found, you have not implemented a handler for

the intent being invoked or included it in the skill builder below.

"""

def can\_handle(self, handler\_input, exception):

# type: (HandlerInput, Exception) -> bool

return True

def handle(self, handler\_input, exception):

# type: (HandlerInput, Exception) -> Response

logger.error(exception, exc\_info=True)

speak\_output = "Sorry, I had trouble doing what you asked. Please try again."

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

# The SkillBuilder object acts as the entry point for your skill, routing all request and response

# payloads to the handlers above. Make sure any new handlers or interceptors you've

# defined are included below. The order matters - they're processed top to bottom.

sb = SkillBuilder()

sb.add\_request\_handler(LaunchRequestHandler())

sb.add\_request\_handler(HelloWorldIntentHandler())

sb.add\_request\_handler(HelpIntentHandler())

sb.add\_request\_handler(CancelOrStopIntentHandler())

sb.add\_request\_handler(FallbackIntentHandler())

sb.add\_request\_handler(SessionEndedRequestHandler())

sb.add\_request\_handler(IntentReflectorHandler()) # make sure IntentReflectorHandler is last so it doesn't override your custom intent handlers

sb.add\_exception\_handler(CatchAllExceptionHandler())

lambda\_handler = sb.lambda\_handler()

N.b in the future, (Python 3.11) you should be able to use

asyncio.timeout(delay)

I’ll let you research that