Streaming Tools - Agent Development Kit

Source URL: https://google.github.io/adk-docs/streaming/streaming-tools/

Streaming Tools¶

Info

This is only supported in streaming(live) agents/api.

Streaming tools allows tools(functions) to stream intermediate results back to agents and agents can respond to those intermediate results. For example, we can use streaming tools to monitor the changes of the stock price and have the agent react to it. Another example is we can have the agent monitor the video stream, and when there is changes in video stream, the agent can report the changes.

To define a streaming tool, you must adhere to the following:

1. **Asynchronous Function:** The tool must be an `async` Python function.
2. **AsyncGenerator Return Type:** The function must be typed to return an `AsyncGenerator`. The first type parameter to `AsyncGenerator` is the type of the data you `yield` (e.g., `str` for text messages, or a custom object for structured data). The second type parameter is typically `None` if the generator doesn't receive values via `send()`.

We support two types of streaming tools: - Simple type. This is a one type of streaming tools that only take non video/audio streams(the streams that you feed to adk web or adk runner) as input. - Video streaming tools. This only works in video streaming and the video stream(the streams that you feed to adk web or adk runner) will be passed into this function.

Now let's define an agent that can monitor stock price changes and monitor the video stream changes.

```
import asyncio
from typing import AsyncGenerator
```

```
from google.adk.agents import LiveRequestQueue
from google.adk.agents.llm_agent import Agent
from google.adk.tools.function_tool import FunctionTool
from google.genai import Client
from google.genai import types as genai_types

async def monitor_stock_price(stock_symbol: str) -> AsyncGenerator[str
  """This function will monitor the price for the given stock_symbol i
  print(f"Start monitor stock price for {stock_symbol}!")

  # Let's mock stock price change.
  await asyncio.sleep(4)
  price_alert1 = f"the price for {stock_symbol} is 300"
  yield price_alert1
  print(price_alert1)

  await asyncio.sleep(4)
  price_alert1 = f"the price for {stock_symbol} is 400"
  yield price_alert1
  print(price_alert1)

  await asyncio.sleep(20)
  price_alert1 = f"the price for {stock_symbol} is 900"
  yield price_alert1
  print(price_alert1)

  await asyncio.sleep(20)
  price_alert1 = f"the price for {stock_symbol} is 500"
  yield price_alert1
  print(price_alert1)

# for video streaming, `input_stream: LiveRequestQueue` is required ar
async def monitor_video_stream(
    input_stream: LiveRequestQueue,
) -> AsyncGenerator[str, None]:
```

```
    """Monitor how many people are in the video streams."""
    print("start monitor_video_stream!")
    client = Client(vertexai=False)
    prompt_text = (
        "Count the number of people in this image. Just respond with a n
        " number."
    )
    last_count = None
    while True:
      last_valid_req = None
      print("Start monitoring loop")

      # use this loop to pull the latest images and discard the old ones
      while input_stream._queue.qsize() != 0:
        live_req = await input_stream.get()

        if live_req.blob is not None and live_req.blob.mime_type == "ima
          last_valid_req = live_req

      # If we found a valid image, process it
      if last_valid_req is not None:
        print("Processing the most recent frame from the queue")

        # Create an image part using the blob's data and mime type
        image_part = genai_types.Part.from_bytes(
            data=last_valid_req.blob.data, mime_type=last_valid_req.blob
        )

        contents = genai_types.Content(
            role="user",
            parts=[image_part, genai_types.Part.from_text(prompt_text)],
        )

        # Call the model to generate content based on the provided image
        response = client.models.generate_content(
            model="gemini-2.0-flash-exp",
```

```
            contents=contents,
            config=genai_types.GenerateContentConfig(
                system_instruction=(
                    "You are a helpful video analysis assistant. You can
                    " the number of people in this image or video. Just
                    " with a numeric number."
                )
            ),
        )
        if not last_count:
          last_count = response.candidates[0].content.parts[0].text
        elif last_count != response.candidates[0].content.parts[0].text:
          last_count = response.candidates[0].content.parts[0].text
          yield response
          print("response:", response)

    # Wait before checking for new images
    await asyncio.sleep(0.5)

# Use this exact function to help ADK stop your streaming tools when r
# for example, if we want to stop `monitor_stock_price`, then the agen
# invoke this function with stop_streaming(function_name=monitor_stock
def stop_streaming(function_name: str):
  """Stop the streaming

  Args:
    function_name: The name of the streaming function to stop.
  """
  pass

root_agent = Agent(
    model="gemini-2.0-flash-exp",
    name="video_streaming_agent",
    instruction="""
      You are a monitoring agent. You can do video monitoring and stoc
      using the provided tools/functions.
```

```
      When users want to monitor a video stream,
      You can use monitor_video_stream function to do that. When monit
      returns the alert, you should tell the users.
      When users want to monitor a stock price, you can use monitor_st
      Don't ask too many questions. Don't be too talkative.
    """,
    tools=[
        monitor_video_stream,
        monitor_stock_price,
        FunctionTool(stop_streaming),
    ]
)
```

Here are some sample queries to test: - Help me monitor the stock price for $XYZ stock. - Help me monitor how many people are there in the video stream.