Types of callbacks - Agent Development Kit

Source URL: https://google.github.io/adk-docs/callbacks/types-of-callbacks/

Types of Callbacks¶

The framework provides different types of callbacks that trigger at various stages of an agent's execution. Understanding when each callback fires and what context it receives is key to using them effectively.

Agent Lifecycle Callbacks¶

These callbacks are available on *any* agent that inherits from `BaseAgent` (including `LlmAgent`, `SequentialAgent`, `ParallelAgent`, `LoopAgent`, etc).

Note

The specific method names or return types may vary slightly by SDK language (e.g., return `None` in Python, return `Optional.empty()` or `Maybe.empty()` in Java). Refer to the language-specific API documentation for details.

Before Agent Callback¶

When: Called *immediately before* the agent's `_run_async_impl` (or `_run_live_impl`) method is executed. It runs after the agent's `InvocationContext` is created but *before* its core logic begins.

Purpose: Ideal for setting up resources or state needed only for this specific agent's run, performing validation checks on the session state (callback_context.state) before execution starts, logging the entry point of the agent's activity, or potentially modifying the invocation context before the core logic uses it.

Code

PythonJava

```
# # --- Setup Instructions ---
# # 1. Install the ADK package:
# !pip install google-adk
# # Make sure to restart kernel if using colab/jupyter notebooks

# # 2. Set up your Gemini API Key:
# #    - Get a key from Google AI Studio: https://aistudio.google.com/
# #    - Set it as an environment variable:
# import os
# os.environ["GOOGLE_API_KEY"] = "YOUR_API_KEY_HERE" # <--- REPLACE wi
# # Or learn about other authentication methods (like Vertex AI):
# # https://google.github.io/adk-docs/agents/models/

# ADK Imports
from google.adk.agents import LlmAgent
from google.adk.agents.callback_context import CallbackContext
from google.adk.runners import InMemoryRunner # Use InMemoryRunner
from google.genai import types # For types.Content
from typing import Optional

# Define the model - Use the specific model name requested
GEMINI_2_FLASH="gemini-2.0-flash"

# --- 1. Define the Callback Function ---
def check_if_agent_should_run(callback_context: CallbackContext) -> Op
    """
    Logs entry and checks 'skip_llm_agent' in session state.
    If True, returns Content to skip the agent's execution.
    If False or not present, returns None to allow execution.
    """
    agent_name = callback_context.agent_name
    invocation_id = callback_context.invocation_id
    current_state = callback_context.state.to_dict()
```

```
    print(f"\n[Callback] Entering agent: {agent_name} (Inv: {invocatio
    print(f"[Callback] Current State: {current_state}")

    # Check the condition in session state dictionary
    if current_state.get("skip_llm_agent", False):
        print(f"[Callback] State condition 'skip_llm_agent=True' met:
        # Return Content to skip the agent's run
        return types.Content(
            parts=[types.Part(text=f"Agent {agent_name} skipped by bef
            role="model" # Assign model role to the overriding respons
        )
    else:
        print(f"[Callback] State condition not met: Proceeding with ag
        # Return None to allow the LlmAgent's normal execution
        return None

# --- 2. Setup Agent with Callback ---
llm_agent_with_before_cb = LlmAgent(
    name="MyControlledAgent",
    model=GEMINI_2_FLASH,
    instruction="You are a concise assistant.",
    description="An LLM agent demonstrating stateful before_agent_call
    before_agent_callback=check_if_agent_should_run # Assign the callb
)

# --- 3. Setup Runner and Sessions using InMemoryRunner ---
async def main():
    app_name = "before_agent_demo"
    user_id = "test_user"
    session_id_run = "session_will_run"
    session_id_skip = "session_will_skip"

    # Use InMemoryRunner - it includes InMemorySessionService
    runner = InMemoryRunner(agent=llm_agent_with_before_cb, app_name=a
    # Get the bundled session service to create sessions
    session_service = runner.session_service
```

```
# Create session 1: Agent will run (default empty state)
session_service.create_session(
    app_name=app_name,
    user_id=user_id,
    session_id=session_id_run
    # No initial state means 'skip_llm_agent' will be False in the
)

# Create session 2: Agent will be skipped (state has skip_llm_agen
session_service.create_session(
    app_name=app_name,
    user_id=user_id,
    session_id=session_id_skip,
    state={"skip_llm_agent": True} # Set the state flag here
)

# --- Scenario 1: Run where callback allows agent execution ---
print("\n" + "="*20 + f" SCENARIO 1: Running Agent on Session '{se
async for event in runner.run_async(
    user_id=user_id,
    session_id=session_id_run,
    new_message=types.Content(role="user", parts=[types.Part(text=
):
    # Print final output (either from LLM or callback override)
    if event.is_final_response() and event.content:
        print(f"Final Output: [{event.author}] {event.content.part
    elif event.is_error():
         print(f"Error Event: {event.error_details}")

# --- Scenario 2: Run where callback intercepts and skips agent --
print("\n" + "="*20 + f" SCENARIO 2: Running Agent on Session '{se
async for event in runner.run_async(
    user_id=user_id,
    session_id=session_id_skip,
    new_message=types.Content(role="user", parts=[types.Part(text=
```

```
    ):
          # Print final output (either from LLM or callback override)
          if event.is_final_response() and event.content:
             print(f"Final Output: [{event.author}] {event.content.part
          elif event.is_error():
              print(f"Error Event: {event.error_details}")

# --- 4. Execute ---
# In a Python script:
# import asyncio
# if __name__ == "__main__":
#     # Make sure GOOGLE_API_KEY environment variable is set if not us
#     # Or ensure Application Default Credentials (ADC) are configured
#     asyncio.run(main())

# In a Jupyter Notebook or similar environment:
await main()
```

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.agents.BaseAgent;
import com.google.adk.agents.CallbackContext;
import com.google.adk.events.Event;
import com.google.adk.runner.InMemoryRunner;
import com.google.adk.sessions.Session;
import com.google.adk.sessions.State;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;
import io.reactivex.rxjava3.core.Maybe;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class BeforeAgentCallbackExample {
```

```
  private static final String APP_NAME = "AgentWithBeforeAgentCallback
  private static final String USER_ID = "test_user_456";
  private static final String SESSION_ID = "session_id_123";
  private static final String MODEL_NAME = "gemini-2.0-flash";

  public static void main(String[] args) {
    BeforeAgentCallbackExample callbackAgent = new BeforeAgentCallback
    callbackAgent.defineAgent("Write a document about a cat");
  }

  // --- 1. Define the Callback Function ---
  /**
   * Logs entry and checks 'skip_llm_agent' in session state. If True,
   * agent's execution. If False or not present, returns None to allow
   */
  public Maybe<Content> checkIfAgentShouldRun(CallbackContext callback
    String agentName = callbackContext.agentName();
    String invocationId = callbackContext.invocationId();
    State currentState = callbackContext.state();

    System.out.printf("%n[Callback] Entering agent: %s (Inv: %s)%n", a
    System.out.printf("[Callback] Current State: %s%n", currentState.e

    // Check the condition in session state dictionary
    if (Boolean.TRUE.equals(currentState.get("skip_llm_agent"))) {
      System.out.printf(
          "[Callback] State condition 'skip_llm_agent=True' met: Skipp
      // Return Content to skip the agent's run
      return Maybe.just(
          Content.fromParts(
              Part.fromText(
                  String.format(
                      "Agent %s skipped by before_agent_callback due t
    }

    System.out.printf(
```

```
        "[Callback] State condition 'skip_llm_agent=True' NOT met: Run
    // Return empty response to allow the LlmAgent's normal execution
    return Maybe.empty();
  }

  public void defineAgent(String prompt) {
    // --- 2. Setup Agent with Callback ---
    BaseAgent llmAgentWithBeforeCallback =
        LlmAgent.builder()
            .model(MODEL_NAME)
            .name(APP_NAME)
            .instruction("You are a concise assistant.")
            .description("An LLM agent demonstrating stateful before_a
            // You can also use a sync version of this callback "befor
            .beforeAgentCallback(this::checkIfAgentShouldRun)
            .build();

    // --- 3. Setup Runner and Sessions using InMemoryRunner ---

    // Use InMemoryRunner - it includes InMemorySessionService
    InMemoryRunner runner = new InMemoryRunner(llmAgentWithBeforeCallb
    // Scenario 1: Initial state is null, which means 'skip_llm_agent'
    // check
    runAgent(runner, null, prompt);
    // Scenario 2: Agent will be skipped (state has skip_llm_agent=tru
    runAgent(runner, new ConcurrentHashMap<>(Map.of("skip_llm_agent",
  }

  public void runAgent(InMemoryRunner runner, ConcurrentHashMap<String
    // InMemoryRunner automatically creates a session service. Create
    Session session =
        runner
            .sessionService()
            .createSession(APP_NAME, USER_ID, initialState, SESSION_ID
            .blockingGet();
    Content userMessage = Content.fromParts(Part.fromText(prompt));
```

```
    // Run the agent
    Flowable<Event> eventStream = runner.runAsync(USER_ID, session.id(

    // Print final output (either from LLM or callback override)
    eventStream.blockingForEach(
        event -> {
          if (event.finalResponse()) {
            System.out.println(event.stringifyContent());
          }
        });
  }
}
```

Note on the `before_agent_callback` Example:

- **What it Shows:** This example demonstrates the `before_agent_callback`. This callback runs *right before* the agent's main processing logic starts for a given request.
- **How it Works:** The callback function (`check_if_agent_should_run`) looks at a flag (`skip_llm_agent`) in the session's state.
- If the flag is `True`, the callback returns a `types.Content` object. This tells the ADK framework to **skip** the agent's main execution entirely and use the callback's returned content as the final response.
- If the flag is `False` (or not set), the callback returns `None` or an empty object. This tells the ADK framework to **proceed** with the agent's normal execution (calling the LLM in this case).
- **Expected Outcome:** You'll see two scenarios:
- In the session *with* the `skip_llm_agent: True` state, the agent's LLM call is bypassed, and the output comes directly from the callback ("Agent... skipped...").
- In the session *without* that state flag, the callback allows the agent to run, and you see the actual response from the LLM (e.g., "Hello!").
- **Understanding Callbacks:** This highlights how `before_` callbacks act as **gatekeepers**, allowing you to intercept execution *before* a major step

and potentially prevent it based on checks (like state, input validation, permissions).

After Agent Callback¶

When: Called *immediately after* the agent's `_run_async_impl` (or `_run_live_impl`) method successfully completes. It does *not* run if the agent was skipped due to `before_agent_callback` returning content or if `end_invocation` was set during the agent's run.

Purpose: Useful for cleanup tasks, post-execution validation, logging the completion of an agent's activity, modifying final state, or augmenting/replacing the agent's final output.

Code

PythonJava

```
# # --- Setup Instructions ---
# # 1. Install the ADK package:
# !pip install google-adk
# # Make sure to restart kernel if using colab/jupyter notebooks

# # 2. Set up your Gemini API Key:
# #    - Get a key from Google AI Studio: https://aistudio.google.com/
# #    - Set it as an environment variable:
# import os
# os.environ["GOOGLE_API_KEY"] = "YOUR_API_KEY_HERE" # <--- REPLACE wi
# # Or learn about other authentication methods (like Vertex AI):
# # https://google.github.io/adk-docs/agents/models/

# ADK Imports
from google.adk.agents import LlmAgent
from google.adk.agents.callback_context import CallbackContext
from google.adk.runners import InMemoryRunner # Use InMemoryRunner
from google.genai import types # For types.Content
from typing import Optional
```

```
# Define the model - Use the specific model name requested
GEMINI_2_FLASH="gemini-2.0-flash"

# --- 1. Define the Callback Function ---
def modify_output_after_agent(callback_context: CallbackContext) -> Op
    """
    Logs exit from an agent and checks 'add_concluding_note' in sessio
    If True, returns new Content to *replace* the agent's original out
    If False or not present, returns None, allowing the agent's origin
    """
    agent_name = callback_context.agent_name
    invocation_id = callback_context.invocation_id
    current_state = callback_context.state.to_dict()

    print(f"\n[Callback] Exiting agent: {agent_name} (Inv: {invocation
    print(f"[Callback] Current State: {current_state}")

    # Example: Check state to decide whether to modify the final outpu
    if current_state.get("add_concluding_note", False):
        print(f"[Callback] State condition 'add_concluding_note=True'
        # Return Content to *replace* the agent's own output
        return types.Content(
            parts=[types.Part(text=f"Concluding note added by after_ag
            role="model" # Assign model role to the overriding respons
        )
    else:
        print(f"[Callback] State condition not met: Using agent {agent
        # Return None - the agent's output produced just before this c
        return None

# --- 2. Setup Agent with Callback ---
llm_agent_with_after_cb = LlmAgent(
    name="MySimpleAgentWithAfter",
    model=GEMINI_2_FLASH,
    instruction="You are a simple agent. Just say 'Processing complete
    description="An LLM agent demonstrating after_agent_callback for c
```

```
    after_agent_callback=modify_output_after_agent # Assign the callba
)

# --- 3. Setup Runner and Sessions using InMemoryRunner ---
async def main():
    app_name = "after_agent_demo"
    user_id = "test_user_after"
    session_id_normal = "session_run_normally"
    session_id_modify = "session_modify_output"

    # Use InMemoryRunner - it includes InMemorySessionService
    runner = InMemoryRunner(agent=llm_agent_with_after_cb, app_name=ap
    # Get the bundled session service to create sessions
    session_service = runner.session_service

    # Create session 1: Agent output will be used as is (default empty
    session_service.create_session(
        app_name=app_name,
        user_id=user_id,
        session_id=session_id_normal
        # No initial state means 'add_concluding_note' will be False i
    )
    # print(f"Session '{session_id_normal}' created with default state

    # Create session 2: Agent output will be replaced by the callback
    session_service.create_session(
        app_name=app_name,
        user_id=user_id,
        session_id=session_id_modify,
        state={"add_concluding_note": True} # Set the state flag here
    )
    # print(f"Session '{session_id_modify}' created with state={{'add_

    # --- Scenario 1: Run where callback allows agent's original outpu
    print("\n" + "="*20 + f" SCENARIO 1: Running Agent on Session '{se
    async for event in runner.run_async(
```

```
        user_id=user_id,
        session_id=session_id_normal,
        new_message=types.Content(role="user", parts=[types.Part(text=
    ):
        # Print final output (either from LLM or callback override)
        if event.is_final_response() and event.content:
            print(f"Final Output: [{event.author}] {event.content.part
        elif event.is_error():
             print(f"Error Event: {event.error_details}")

    # --- Scenario 2: Run where callback replaces the agent's output -
    print("\n" + "="*20 + f" SCENARIO 2: Running Agent on Session '{se
    async for event in runner.run_async(
        user_id=user_id,
        session_id=session_id_modify,
        new_message=types.Content(role="user", parts=[types.Part(text=
    ):
         # Print final output (either from LLM or callback override)
         if event.is_final_response() and event.content:
            print(f"Final Output: [{event.author}] {event.content.part
         elif event.is_error():
             print(f"Error Event: {event.error_details}")

# --- 4. Execute ---
# In a Python script:
# import asyncio
# if __name__ == "__main__":
#     # Make sure GOOGLE_API_KEY environment variable is set if not us
#     # Or ensure Application Default Credentials (ADC) are configured
#     asyncio.run(main())

# In a Jupyter Notebook or similar environment:
await main()
```

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.agents.CallbackContext;
import com.google.adk.events.Event;
import com.google.adk.runner.InMemoryRunner;
import com.google.adk.sessions.State;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;
import io.reactivex.rxjava3.core.Maybe;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class AfterAgentCallbackExample {

  // --- Constants ---
  private static final String APP_NAME = "after_agent_demo";
  private static final String USER_ID = "test_user_after";
  private static final String SESSION_ID_NORMAL = "session_run_normall
  private static final String SESSION_ID_MODIFY = "session_modify_outp
  private static final String MODEL_NAME = "gemini-2.0-flash";

  public static void main(String[] args) {
    AfterAgentCallbackExample demo = new AfterAgentCallbackExample();
    demo.defineAgentAndRunScenarios();
  }

  // --- 1. Define the Callback Function ---
  /**
   * Log exit from an agent and checks 'add_concluding_note' in sessio
   * Content to *replace* the agent's original output. If False or not
   * Maybe.empty(), allowing the agent's original output to be used.
   */
  public Maybe<Content> modifyOutputAfterAgent(CallbackContext callbac
```

```
    String agentName = callbackContext.agentName();
    String invocationId = callbackContext.invocationId();
    State currentState = callbackContext.state();

    System.out.printf("%n[Callback] Exiting agent: %s (Inv: %s)%n", ag
    System.out.printf("[Callback] Current State: %s%n", currentState.e

    Object addNoteFlag = currentState.get("add_concluding_note");

    // Example: Check state to decide whether to modify the final outp
    if (Boolean.TRUE.equals(addNoteFlag)) {
      System.out.printf(
          "[Callback] State condition 'add_concluding_note=True' met:
              + " output.%n",
          agentName);

      // Return Content to *replace* the agent's own output
      return Maybe.just(
          Content.builder()
              .parts(
                  List.of(
                      Part.fromText(
                          "Concluding note added by after_agent_callba
              .role("model") // Assign model role to the overriding re
              .build());

    } else {
      System.out.printf(
          "[Callback] State condition not met: Using agent %s's origin
      // Return None - the agent's output produced just before this ca
      return Maybe.empty();
    }
  }

  // --- 2. Setup Agent with Callback ---
  public void defineAgentAndRunScenarios() {
```

```
    LlmAgent llmAgentWithAfterCb =
        LlmAgent.builder()
            .name(APP_NAME)
            .model(MODEL_NAME)
            .description("An LLM agent demonstrating after_agent_callb
            .instruction("You are a simple agent. Just say 'Processing
            .afterAgentCallback(this::modifyOutputAfterAgent) // Assig
            .build();

    // --- 3. Setup Runner and Sessions using InMemoryRunner ---
    // Use InMemoryRunner - it includes InMemorySessionService
    InMemoryRunner runner = new InMemoryRunner(llmAgentWithAfterCb, AP

    // --- Scenario 1: Run where callback allows agent's original outp
    System.out.printf(
        "%n%s SCENARIO 1: Running Agent (Should Use Original Output) %
        "=".repeat(20), "=".repeat(20));
    // No initial state means 'add_concluding_note' will be false in t
    runScenario(
        runner,
        llmAgentWithAfterCb.name(), // Use agent name for runner's app
        SESSION_ID_NORMAL,
        null,
        "Process this please.");

    // --- Scenario 2: Run where callback replaces the agent's output
    System.out.printf(
        "%n%s SCENARIO 2: Running Agent (Should Replace Output) %s%n",
        "=".repeat(20), "=".repeat(20));
    Map<String, Object> modifyState = new HashMap<>();
    modifyState.put("add_concluding_note", true); // Set the state fla
    runScenario(
        runner,
        llmAgentWithAfterCb.name(), // Use agent name for runner's app
        SESSION_ID_MODIFY,
        new ConcurrentHashMap<>(modifyState),
```

```
        "Process this and add note.");
  }

  // --- 3. Method to Run a Single Scenario ---
  public void runScenario(
      InMemoryRunner runner,
      String appName,
      String sessionId,
      ConcurrentHashMap<String, Object> initialState,
      String userQuery) {

    // Create session using the runner's bundled session service
    runner.sessionService().createSession(appName, USER_ID, initialSta

    System.out.printf(
        "Running scenario for session: %s, initial state: %s%n", sessi
    Content userMessage =
        Content.builder().role("user").parts(List.of(Part.fromText(use

    Flowable<Event> eventStream = runner.runAsync(USER_ID, sessionId,

    // Print final output
    eventStream.blockingForEach(
        event -> {
          if (event.finalResponse() && event.content().isPresent()) {
            String author = event.author() != null ? event.author() :
            String text =
                event
                    .content()
                    .flatMap(Content::parts)
                    .filter(parts -> !parts.isEmpty())
                    .map(parts -> parts.get(0).text().orElse("").trim(
                    .orElse("[No text in final response]");
            System.out.printf("Final Output for %s: [%s] %s%n", sessio
          } else if (event.errorCode().isPresent()) {
            System.out.printf(
```

```
                "Error Event for %s: %s%n",
                sessionId, event.errorMessage().orElse("Unknown error"
            }
        });
    }
}
```

Note on the `after_agent_callback` Example:

- **What it Shows:** This example demonstrates the `after_agent_callback`. This callback runs *right after* the agent's main processing logic has finished and produced its result, but *before* that result is finalized and returned.
- **How it Works:** The callback function (`modify_output_after_agent`) checks a flag (`add_concluding_note`) in the session's state.
- If the flag is `True`, the callback returns a *new* `types.Content` object. This tells the ADK framework to **replace** the agent's original output with the content returned by the callback.
- If the flag is `False` (or not set), the callback returns `None` or an empty object. This tells the ADK framework to **use** the original output generated by the agent.
- **Expected Outcome:** You'll see two scenarios:
- In the session *without* the `add_concluding_note: True` state, the callback allows the agent's original output ("Processing complete!") to be used.
- In the session *with* that state flag, the callback intercepts the agent's original output and replaces it with its own message ("Concluding note added...").
- **Understanding Callbacks:** This highlights how `after_` callbacks allow **post-processing** or **modification**. You can inspect the result of a step (the agent's run) and decide whether to let it pass through, change it, or completely replace it based on your logic.

LLM Interaction Callbacks¶

These callbacks are specific to `LlmAgent` and provide hooks around the interaction with the Large Language Model.

Before Model Callback¶

When: Called just before the `generate_content_async` (or equivalent) request is sent to the LLM within an `LlmAgent`'s flow.

Purpose: Allows inspection and modification of the request going to the LLM. Use cases include adding dynamic instructions, injecting few-shot examples based on state, modifying model config, implementing guardrails (like profanity filters), or implementing request-level caching.

Return Value Effect:
If the callback returns `None` (or a `Maybe.empty()` object in Java), the LLM continues its normal workflow. If the callback returns an `LlmResponse` object, then the call to the LLM is **skipped**. The returned `LlmResponse` is used directly as if it came from the model. This is powerful for implementing guardrails or caching.

Code

PythonJava

```
from google.adk.agents import LlmAgent
from google.adk.agents.callback_context import CallbackContext
from google.adk.models import LlmResponse, LlmRequest
from google.adk.runners import Runner
from typing import Optional
from google.genai import types
from google.adk.sessions import InMemorySessionService

GEMINI_2_FLASH="gemini-2.0-flash"

# --- Define the Callback Function ---
def simple_before_model_modifier(
    callback_context: CallbackContext, llm_request: LlmRequest
```

```
) -> Optional[LlmResponse]:
    """Inspects/modifies the LLM request or skips the call."""
    agent_name = callback_context.agent_name
    print(f"[Callback] Before model call for agent: {agent_name}")

    # Inspect the last user message in the request contents
    last_user_message = ""
    if llm_request.contents and llm_request.contents[-1].role == 'user
        if llm_request.contents[-1].parts:
            last_user_message = llm_request.contents[-1].parts[0].text
    print(f"[Callback] Inspecting last user message: '{last_user_messa

    # --- Modification Example ---
    # Add a prefix to the system instruction
    original_instruction = llm_request.config.system_instruction or ty
    prefix = "[Modified by Callback] "
    # Ensure system_instruction is Content and parts list exists
    if not isinstance(original_instruction, types.Content):
        # Handle case where it might be a string (though config expec
        original_instruction = types.Content(role="system", parts=[ty
    if not original_instruction.parts:
        original_instruction.parts.append(types.Part(text="")) # Add a

    # Modify the text of the first part
    modified_text = prefix + (original_instruction.parts[0].text or ""
    original_instruction.parts[0].text = modified_text
    llm_request.config.system_instruction = original_instruction
    print(f"[Callback] Modified system instruction to: '{modified_text

    # --- Skip Example ---
    # Check if the last user message contains "BLOCK"
    if "BLOCK" in last_user_message.upper():
        print("[Callback] 'BLOCK' keyword found. Skipping LLM call.")
        # Return an LlmResponse to skip the actual LLM call
        return LlmResponse(
            content=types.Content(
```

```
                role="model",
                parts=[types.Part(text="LLM call was blocked by before
            )
        )
    else:
        print("[Callback] Proceeding with LLM call.")
        # Return None to allow the (modified) request to go to the LLM
        return None

# Create LlmAgent and Assign Callback
my_llm_agent = LlmAgent(
        name="ModelCallbackAgent",
        model=GEMINI_2_FLASH,
        instruction="You are a helpful assistant.", # Base instruction
        description="An LLM agent demonstrating before_model_callback"
        before_model_callback=simple_before_model_modifier # Assign th
)

APP_NAME = "guardrail_app"
USER_ID = "user_1"
SESSION_ID = "session_001"

# Session and Runner
session_service = InMemorySessionService()
session = session_service.create_session(app_name=APP_NAME, user_id=US
runner = Runner(agent=my_llm_agent, app_name=APP_NAME, session_service

# Agent Interaction
def call_agent(query):
  content = types.Content(role='user', parts=[types.Part(text=query)])
  events = runner.run(user_id=USER_ID, session_id=SESSION_ID, new_mess

  for event in events:
      if event.is_final_response():
          final_response = event.content.parts[0].text
          print("Agent Response: ", final_response)
```

```
call_agent("callback example")
```

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.agents.CallbackContext;
import com.google.adk.events.Event;
import com.google.adk.models.LlmRequest;
import com.google.adk.models.LlmResponse;
import com.google.adk.runner.InMemoryRunner;
import com.google.adk.sessions.Session;
import com.google.common.collect.ImmutableList;
import com.google.common.collect.Iterables;
import com.google.genai.types.Content;
import com.google.genai.types.GenerateContentConfig;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;
import io.reactivex.rxjava3.core.Maybe;
import java.util.ArrayList;
import java.util.List;

public class BeforeModelCallbackExample {

  // --- Define Constants ---
  private static final String AGENT_NAME = "ModelCallbackAgent";
  private static final String MODEL_NAME = "gemini-2.0-flash";
  private static final String AGENT_INSTRUCTION = "You are a helpful a
  private static final String AGENT_DESCRIPTION =
      "An LLM agent demonstrating before_model_callback";

  // For session and runner
  private static final String APP_NAME = "guardrail_app_java";
  private static final String USER_ID = "user_1_java";

  public static void main(String[] args) {
```

```
    BeforeModelCallbackExample demo = new BeforeModelCallbackExample()
    demo.defineAgentAndRun();
  }

  // --- 1. Define the Callback Function ---
  // Inspects/modifies the LLM request or skips the actual LLM call.
  public Maybe<LlmResponse> simpleBeforeModelModifier(
      CallbackContext callbackContext, LlmRequest llmRequest) {
    String agentName = callbackContext.agentName();
    System.out.printf("%n[Callback] Before model call for agent: %s%n"

    String lastUserMessage = "";
    if (llmRequest.contents() != null && !llmRequest.contents().isEmpt
      Content lastContentItem = Iterables.getLast(llmRequest.contents(
      if ("user".equals(lastContentItem.role().orElse(null))
          && lastContentItem.parts().isPresent()
          && !lastContentItem.parts().get().isEmpty()) {
        lastUserMessage = lastContentItem.parts().get().get(0).text().
      }
    }
    System.out.printf("[Callback] Inspecting last user message: '%s'%n

    // --- Modification Example ---
    // Add a prefix to the system instruction
    Content systemInstructionFromRequest = Content.builder().parts(Imm
    // Ensure system_instruction is Content and parts list exists
    if (llmRequest.config().isPresent()) {
      systemInstructionFromRequest =
          llmRequest
              .config()
              .get()
              .systemInstruction()
              .orElseGet(() -> Content.builder().role("system").parts(
    }
    List<Part> currentSystemParts =
        new ArrayList<>(systemInstructionFromRequest.parts().orElse(Im
```

```
    // Ensure a part exists for modification
    if (currentSystemParts.isEmpty()) {
      currentSystemParts.add(Part.fromText(""));
    }
    // Modify the text of the first part
    String prefix = "[Modified by Callback] ";
    String conceptuallyModifiedText = prefix + currentSystemParts.get(
    llmRequest =
        llmRequest.toBuilder()
            .config(
                GenerateContentConfig.builder()
                    .systemInstruction(
                        Content.builder()
                            .parts(List.of(Part.fromText(conceptuallyM
                            .build())
                    .build())
            .build();
    System.out.printf(
        "Modified System Instruction %s", llmRequest.config().get().sy

    // --- Skip Example ---
    // Check if the last user message contains "BLOCK"
    if (lastUserMessage.toUpperCase().contains("BLOCK")) {
      System.out.println("[Callback] 'BLOCK' keyword found. Skipping L
      // Return an LlmResponse to skip the actual LLM call
      return Maybe.just(
          LlmResponse.builder()
              .content(
                  Content.builder()
                      .role("model")
                      .parts(
                          ImmutableList.of(
                              Part.fromText("LLM call was blocked by b
                      .build())
              .build());
    }
```

```
  // Return Empty response to allow the (modified) request to go to
  System.out.println("[Callback] Proceeding with LLM call (using the
  return Maybe.empty();
}

// --- 2. Define Agent and Run Scenarios ---
public void defineAgentAndRun() {
  // Setup Agent with Callback
  LlmAgent myLlmAgent =
      LlmAgent.builder()
          .name(AGENT_NAME)
          .model(MODEL_NAME)
          .instruction(AGENT_INSTRUCTION)
          .description(AGENT_DESCRIPTION)
          .beforeModelCallback(this::simpleBeforeModelModifier)
          .build();

  // Create an InMemoryRunner
  InMemoryRunner runner = new InMemoryRunner(myLlmAgent, APP_NAME);
  // InMemoryRunner automatically creates a session service. Create
  Session session = runner.sessionService().createSession(APP_NAME,
  Content userMessage =
      Content.fromParts(
          Part.fromText("Tell me about quantum computing. This is a

  // Run the agent
  Flowable<Event> eventStream = runner.runAsync(USER_ID, session.id(

  // Stream event response
  eventStream.blockingForEach(
      event -> {
        if (event.finalResponse()) {
          System.out.println(event.stringifyContent());
        }
      });
```

```
  }
}
```

After Model Callback¶

When: Called just after a response (`LlmResponse`) is received from the LLM, before it's processed further by the invoking agent.

Purpose: Allows inspection or modification of the raw LLM response. Use cases include

- logging model outputs,
- reformatting responses,
- censoring sensitive information generated by the model,
- parsing structured data from the LLM response and storing it in `callback_context.state`
- or handling specific error codes.

Code

PythonJava

```
from google.adk.agents import LlmAgent
from google.adk.agents.callback_context import CallbackContext
from google.adk.runners import Runner
from typing import Optional
from google.genai import types
from google.adk.sessions import InMemorySessionService
from google.adk.models import LlmResponse

GEMINI_2_FLASH="gemini-2.0-flash"

# --- Define the Callback Function ---
def simple_after_model_modifier(
    callback_context: CallbackContext, llm_response: LlmResponse
) -> Optional[LlmResponse]:
    """Inspects/modifies the LLM response after it's received."""
```

```
    agent_name = callback_context.agent_name
    print(f"[Callback] After model call for agent: {agent_name}")

    # --- Inspection ---
    original_text = ""
    if llm_response.content and llm_response.content.parts:
        # Assuming simple text response for this example
        if llm_response.content.parts[0].text:
            original_text = llm_response.content.parts[0].text
            print(f"[Callback] Inspected original response text: '{ori
        elif llm_response.content.parts[0].function_call:
             print(f"[Callback] Inspected response: Contains function
             return None # Don't modify tool calls in this example
        else:
             print("[Callback] Inspected response: No text content fou
             return None
    elif llm_response.error_message:
        print(f"[Callback] Inspected response: Contains error '{llm_re
        return None
    else:
        print("[Callback] Inspected response: Empty LlmResponse.")
        return None # Nothing to modify

    # --- Modification Example ---
    # Replace "joke" with "funny story" (case-insensitive)
    search_term = "joke"
    replace_term = "funny story"
    if search_term in original_text.lower():
        print(f"[Callback] Found '{search_term}'. Modifying response."
        modified_text = original_text.replace(search_term, replace_ter
        modified_text = modified_text.replace(search_term.capitalize()

        # Create a NEW LlmResponse with the modified content
        # Deep copy parts to avoid modifying original if other callbac
        modified_parts = [copy.deepcopy(part) for part in llm_response
        modified_parts[0].text = modified_text # Update the text in th
```

```
        new_response = LlmResponse(
             content=types.Content(role="model", parts=modified_parts)
             # Copy other relevant fields if necessary, e.g., groundin
             grounding_metadata=llm_response.grounding_metadata
             )
        print(f"[Callback] Returning modified response.")
        return new_response # Return the modified response
    else:
        print(f"[Callback] '{search_term}' not found. Passing original
        # Return None to use the original llm_response
        return None

# Create LlmAgent and Assign Callback
my_llm_agent = LlmAgent(
        name="AfterModelCallbackAgent",
        model=GEMINI_2_FLASH,
        instruction="You are a helpful assistant.",
        description="An LLM agent demonstrating after_model_callback",
        after_model_callback=simple_after_model_modifier # Assign the
)

APP_NAME = "guardrail_app"
USER_ID = "user_1"
SESSION_ID = "session_001"

# Session and Runner
session_service = InMemorySessionService()
session = session_service.create_session(app_name=APP_NAME, user_id=US
runner = Runner(agent=my_llm_agent, app_name=APP_NAME, session_service

# Agent Interaction
def call_agent(query):
  content = types.Content(role='user', parts=[types.Part(text=query)])
  events = runner.run(user_id=USER_ID, session_id=SESSION_ID, new_mess
```

```
  for event in events:
      if event.is_final_response():
          final_response = event.content.parts[0].text
          print("Agent Response: ", final_response)

call_agent("callback example")
```

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.agents.CallbackContext;
import com.google.adk.events.Event;
import com.google.adk.models.LlmResponse;
import com.google.adk.runner.InMemoryRunner;
import com.google.adk.sessions.Session;
import com.google.common.collect.ImmutableList;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;
import io.reactivex.rxjava3.core.Maybe;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class AfterModelCallbackExample {

  // --- Define Constants ---
  private static final String AGENT_NAME = "AfterModelCallbackAgent";
  private static final String MODEL_NAME = "gemini-2.0-flash";
  private static final String AGENT_INSTRUCTION = "You are a helpful a
  private static final String AGENT_DESCRIPTION = "An LLM agent demons

  // For session and runner
  private static final String APP_NAME = "AfterModelCallbackAgentApp";
```

```
  private static final String USER_ID = "user_1";

  // For text replacement
  private static final String SEARCH_TERM = "joke";
  private static final String REPLACE_TERM = "funny story";
  private static final Pattern SEARCH_PATTERN =
      Pattern.compile("\\b" + Pattern.quote(SEARCH_TERM) + "\\b", Patt

  public static void main(String[] args) {
    AfterModelCallbackExample example = new AfterModelCallbackExample(
    example.defineAgentAndRun();
  }

  // --- Define the Callback Function ---
  // Inspects/modifies the LLM response after it's received.
  public Maybe<LlmResponse> simpleAfterModelModifier(
      CallbackContext callbackContext, LlmResponse llmResponse) {
    String agentName = callbackContext.agentName();
    System.out.printf("%n[Callback] After model call for agent: %s%n",

    // --- Inspection Phase ---
    if (llmResponse.errorMessage().isPresent()) {
      System.out.printf(
          "[Callback] Response has error: '%s'. No modification.%n",
          llmResponse.errorMessage().get());
      return Maybe.empty(); // Pass through errors
    }

    Optional<Part> firstTextPartOpt =
        llmResponse
            .content()
            .flatMap(Content::parts)
            .filter(parts -> !parts.isEmpty() && parts.get(0).text().i
            .map(parts -> parts.get(0));

    if (!firstTextPartOpt.isPresent()) {
```

```
    // Could be a function call, empty content, or no text in the fi
    llmResponse
        .content()
        .flatMap(Content::parts)
        .filter(parts -> !parts.isEmpty() && parts.get(0).functionCa
        .ifPresent(
            parts ->
                System.out.printf(
                    "[Callback] Response is a function call ('%s').
                    parts.get(0).functionCall().get().name().orElse(
    if (!llmResponse.content().isPresent()
        || !llmResponse.content().flatMap(Content::parts).isPresent(
        || llmResponse.content().flatMap(Content::parts).get().isEmp
      System.out.println(
          "[Callback] Response content is empty or has no parts. No
    } else if (!firstTextPartOpt.isPresent()) { // Already checked f
      System.out.println("[Callback] First part has no text content.
    }
    return Maybe.empty(); // Pass through non-text or unsuitable res
  }

  String originalText = firstTextPartOpt.get().text().get();
  System.out.printf("[Callback] Inspected original text: '%.100s..."

  // --- Modification Phase ---
  Matcher matcher = SEARCH_PATTERN.matcher(originalText);
  if (!matcher.find()) {
    System.out.printf(
        "[Callback] '%s' not found. Passing original response throug
    return Maybe.empty();
  }

  System.out.printf("[Callback] Found '%s'. Modifying response.%n",

  // Perform the replacement, respecting original capitalization of
  String foundTerm = matcher.group(0); // The actual term found (e.g
```

```
    String actualReplaceTerm = REPLACE_TERM;
    if (Character.isUpperCase(foundTerm.charAt(0)) && REPLACE_TERM.len
      actualReplaceTerm = Character.toUpperCase(REPLACE_TERM.charAt(0)
    }
    String modifiedText = matcher.replaceFirst(Matcher.quoteReplacemen

    // Create a new LlmResponse with the modified content
    Content originalContent = llmResponse.content().get();
    List<Part> originalParts = originalContent.parts().orElse(Immutabl

    List<Part> modifiedPartsList = new ArrayList<>(originalParts.size(
    if (!originalParts.isEmpty()) {
      modifiedPartsList.add(Part.fromText(modifiedText)); // Replace f
      // Add remaining parts as they were (shallow copy)
      for (int i = 1; i < originalParts.size(); i++) {
        modifiedPartsList.add(originalParts.get(i));
      }
    } else { // Should not happen if firstTextPartOpt was present
      modifiedPartsList.add(Part.fromText(modifiedText));
    }

    LlmResponse.Builder newResponseBuilder =
        LlmResponse.builder()
            .content(
                originalContent.toBuilder().parts(ImmutableList.copyOf
            .groundingMetadata(llmResponse.groundingMetadata());

    System.out.println("[Callback] Returning modified response.");
    return Maybe.just(newResponseBuilder.build());
  }

  // --- 2. Define Agent and Run Scenarios ---
  public void defineAgentAndRun() {
    // Setup Agent with Callback
    LlmAgent myLlmAgent =
        LlmAgent.builder()
```

```
            .name(AGENT_NAME)
            .model(MODEL_NAME)
            .instruction(AGENT_INSTRUCTION)
            .description(AGENT_DESCRIPTION)
            .afterModelCallback(this::simpleAfterModelModifier)
            .build();

    // Create an InMemoryRunner
    InMemoryRunner runner = new InMemoryRunner(myLlmAgent, APP_NAME);
    // InMemoryRunner automatically creates a session service. Create
    Session session = runner.sessionService().createSession(APP_NAME,
    Content userMessage =
        Content.fromParts(
            Part.fromText(
                "Tell me a joke about quantum computing. Include the w

    // Run the agent
    Flowable<Event> eventStream = runner.runAsync(USER_ID, session.id(

    // Stream event response
    eventStream.blockingForEach(
        event -> {
          if (event.finalResponse()) {
            System.out.println(event.stringifyContent());
          }
        });
  }
}
```

Tool Execution Callbacks¶

These callbacks are also specific to `LlmAgent` and trigger around the execution of tools (including `FunctionTool`, `AgentTool`, etc.) that the LLM might request.

Before Tool Callback¶

When: Called just before a specific tool's `run_async` method is invoked, after the LLM has generated a function call for it.

Purpose: Allows inspection and modification of tool arguments, performing authorization checks before execution, logging tool usage attempts, or implementing tool-level caching.

Return Value Effect:

1. If the callback returns `None` (or a `Maybe.empty()` object in Java), the tool's `run_async` method is executed with the (potentially modified) `args`.
2. If a dictionary (or `Map` in Java) is returned, the tool's `run_async` method is **skipped**. The returned dictionary is used directly as the result of the tool call. This is useful for caching or overriding tool behavior.

Code

PythonJava

```
 from google.adk.agents import LlmAgent
from google.adk.runners import Runner
from typing import Optional
from google.genai import types
from google.adk.sessions import InMemorySessionService
from google.adk.tools import FunctionTool
from google.adk.tools.tool_context import ToolContext
from google.adk.tools.base_tool import BaseTool
from typing import Dict, Any

GEMINI_2_FLASH="gemini-2.0-flash"

def get_capital_city(country: str) -> str:
    """Retrieves the capital city of a given country."""
    print(f"--- Tool 'get_capital_city' executing with country: {count
    country_capitals = {
        "united states": "Washington, D.C.",
```

```
        "canada": "Ottawa",
        "france": "Paris",
        "germany": "Berlin",
    }
    return country_capitals.get(country.lower(), f"Capital not found f

capital_tool = FunctionTool(func=get_capital_city)

def simple_before_tool_modifier(
    tool: BaseTool, args: Dict[str, Any], tool_context: ToolContext
) -> Optional[Dict]:
    """Inspects/modifies tool args or skips the tool call."""
    agent_name = tool_context.agent_name
    tool_name = tool.name
    print(f"[Callback] Before tool call for tool '{tool_name}' in ager
    print(f"[Callback] Original args: {args}")

    if tool_name == 'get_capital_city' and args.get('country', '').low
        print("[Callback] Detected 'Canada'. Modifying args to 'France
        args['country'] = 'France'
        print(f"[Callback] Modified args: {args}")
        return None

    # If the tool is 'get_capital_city' and country is 'BLOCK'
    if tool_name == 'get_capital_city' and args.get('country', '').upp
        print("[Callback] Detected 'BLOCK'. Skipping tool execution.")
        return {"result": "Tool execution was blocked by before_tool_c

    print("[Callback] Proceeding with original or previously modified
    return None

my_llm_agent = LlmAgent(
        name="ToolCallbackAgent",
        model=GEMINI_2_FLASH,
        instruction="You are an agent that can find capital cities. Us
        description="An LLM agent demonstrating before_tool_callback",
```

```
        tools=[capital_tool],
        before_tool_callback=simple_before_tool_modifier
)

APP_NAME = "guardrail_app"
USER_ID = "user_1"
SESSION_ID = "session_001"

# Session and Runner
session_service = InMemorySessionService()
session = session_service.create_session(app_name=APP_NAME, user_id=US
runner = Runner(agent=my_llm_agent, app_name=APP_NAME, session_service

# Agent Interaction
def call_agent(query):
  content = types.Content(role='user', parts=[types.Part(text=query)])
  events = runner.run(user_id=USER_ID, session_id=SESSION_ID, new_mess

  for event in events:
      if event.is_final_response():
          final_response = event.content.parts[0].text
          print("Agent Response: ", final_response)

call_agent("callback example")
```

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.agents.InvocationContext;
import com.google.adk.events.Event;
import com.google.adk.runner.InMemoryRunner;
import com.google.adk.sessions.Session;
import com.google.adk.tools.Annotations.Schema;
import com.google.adk.tools.BaseTool;
import com.google.adk.tools.FunctionTool;
import com.google.adk.tools.ToolContext;
```

```
import com.google.common.collect.ImmutableMap;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;
import io.reactivex.rxjava3.core.Maybe;
import java.util.HashMap;
import java.util.Map;

public class BeforeToolCallbackExample {

  private static final String APP_NAME = "ToolCallbackAgentApp";
  private static final String USER_ID = "user_1";
  private static final String SESSION_ID = "session_001";
  private static final String MODEL_NAME = "gemini-2.0-flash";

  public static void main(String[] args) {
    BeforeToolCallbackExample example = new BeforeToolCallbackExample(
    example.runAgent("capital of canada");
  }

  // --- Define a Simple Tool Function ---
  // The Schema is important for the callback "args" to correctly iden
  public static Map<String, Object> getCapitalCity(
      @Schema(name = "country", description = "The country to find the
          String country) {
    System.out.printf("--- Tool 'getCapitalCity' executing with countr
    Map<String, String> countryCapitals = new HashMap<>();
    countryCapitals.put("united states", "Washington, D.C.");
    countryCapitals.put("canada", "Ottawa");
    countryCapitals.put("france", "Paris");
    countryCapitals.put("germany", "Berlin");

    String capital =
        countryCapitals.getOrDefault(country.toLowerCase(), "Capital n
    // FunctionTool expects a Map<String, Object> as the return type f
    return ImmutableMap.of("capital", capital);
```

```
}

// Define the Callback function
// The Tool callback provides all these parameters by default.
public Maybe<Map<String, Object>> simpleBeforeToolModifier(
    InvocationContext invocationContext,
    BaseTool tool,
    Map<String, Object> args,
    ToolContext toolContext) {

  String agentName = invocationContext.agent().name();
  String toolName = tool.name();
  System.out.printf(
      "[Callback] Before tool call for tool '%s' in agent '%s'%n", t
  System.out.printf("[Callback] Original args: %s%n", args);

  if ("getCapitalCity".equals(toolName)) {
    String countryArg = (String) args.get("country");
    if (countryArg != null) {
      if ("canada".equalsIgnoreCase(countryArg)) {
        System.out.println("[Callback] Detected 'Canada'. Modifying
        args.put("country", "France");
        System.out.printf("[Callback] Modified args: %s%n", args);
        // Proceed with modified args
        return Maybe.empty();
      } else if ("BLOCK".equalsIgnoreCase(countryArg)) {
        System.out.println("[Callback] Detected 'BLOCK'. Skipping to
        // Return a map to skip the tool call and use this as the re
        return Maybe.just(
            ImmutableMap.of("result", "Tool execution was blocked by
      }
    }
  }

  System.out.println("[Callback] Proceeding with original or previou
  return Maybe.empty();
```

```
  }

  public void runAgent(String query) {
    // --- Wrap the function into a Tool ---
    FunctionTool capitalTool = FunctionTool.create(this.getClass(), "g

    // Create LlmAgent and Assign Callback
    LlmAgent myLlmAgent =
        LlmAgent.builder()
            .name(APP_NAME)
            .model(MODEL_NAME)
            .instruction(
                "You are an agent that can find capital cities. Use th
            .description("An LLM agent demonstrating before_tool_callb
            .tools(capitalTool)
            .beforeToolCallback(this::simpleBeforeToolModifier)
            .build();

    // Session and Runner
    InMemoryRunner runner = new InMemoryRunner(myLlmAgent);
    Session session =
        runner.sessionService().createSession(APP_NAME, USER_ID, null,

    Content userMessage = Content.fromParts(Part.fromText(query));

    System.out.printf("%n--- Calling agent with query: \"%s\" ---%n",
    Flowable<Event> eventStream = runner.runAsync(USER_ID, session.id(
    // Stream event response
    eventStream.blockingForEach(
        event -> {
          if (event.finalResponse()) {
            System.out.println(event.stringifyContent());
          }
        });
  }
```

```
}
```

After Tool Callback¶

When: Called just after the tool's `run_async` method completes successfully.

Purpose: Allows inspection and modification of the tool's result before it's sent back to the LLM (potentially after summarization). Useful for logging tool results, post-processing or formatting results, or saving specific parts of the result to the session state.

Return Value Effect:

1. If the callback returns `None` (or a `Maybe.empty()` object in Java), the original `tool_response` is used.
2. If a new dictionary is returned, it **replaces** the original `tool_response`. This allows modifying or filtering the result seen by the LLM.

Code

PythonJava

```
 # Copyright 2025 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or impl
# See the License for the specific language governing permissions and
# limitations under the License.

from google.adk.agents import LlmAgent
```

```
from google.adk.runners import Runner
from typing import Optional
from google.genai import types
from google.adk.sessions import InMemorySessionService
from google.adk.tools import FunctionTool
from google.adk.tools.tool_context import ToolContext
from google.adk.tools.base_tool import BaseTool
from typing import Dict, Any
from copy import deepcopy

GEMINI_2_FLASH="gemini-2.0-flash"

# --- Define a Simple Tool Function (Same as before) ---
def get_capital_city(country: str) -> str:
    """Retrieves the capital city of a given country."""
    print(f"--- Tool 'get_capital_city' executing with country: {count
    country_capitals = {
        "united states": "Washington, D.C.",
        "canada": "Ottawa",
        "france": "Paris",
        "germany": "Berlin",
    }
    return {"result": country_capitals.get(country.lower(), f"Capital

# --- Wrap the function into a Tool ---
capital_tool = FunctionTool(func=get_capital_city)

# --- Define the Callback Function ---
def simple_after_tool_modifier(
    tool: BaseTool, args: Dict[str, Any], tool_context: ToolContext, t
) -> Optional[Dict]:
    """Inspects/modifies the tool result after execution."""
    agent_name = tool_context.agent_name
    tool_name = tool.name
    print(f"[Callback] After tool call for tool '{tool_name}' in agent
    print(f"[Callback] Args used: {args}")
```

```
    print(f"[Callback] Original tool_response: {tool_response}")

    # Default structure for function tool results is {"result": <retur
    original_result_value = tool_response.get("result", "")
    # original_result_value = tool_response

    # --- Modification Example ---
    # If the tool was 'get_capital_city' and result is 'Washington, D.
    if tool_name == 'get_capital_city' and original_result_value == "W
        print("[Callback] Detected 'Washington, D.C.'. Modifying tool

        # IMPORTANT: Create a new dictionary or modify a copy
        modified_response = deepcopy(tool_response)
        modified_response["result"] = f"{original_result_value} (Note:
        modified_response["note_added_by_callback"] = True # Add extra

        print(f"[Callback] Modified tool_response: {modified_response}
        return modified_response # Return the modified dictionary

    print("[Callback] Passing original tool response through.")
    # Return None to use the original tool_response
    return None

# Create LlmAgent and Assign Callback
my_llm_agent = LlmAgent(
        name="AfterToolCallbackAgent",
        model=GEMINI_2_FLASH,
        instruction="You are an agent that finds capital cities using
        description="An LLM agent demonstrating after_tool_callback",
        tools=[capital_tool], # Add the tool
        after_tool_callback=simple_after_tool_modifier # Assign the ca
    )

APP_NAME = "guardrail_app"
USER_ID = "user_1"
SESSION_ID = "session_001"
```

```
# Session and Runner
session_service = InMemorySessionService()
session = await session_service.create_session(app_name=APP_NAME, user
runner = Runner(agent=my_llm_agent, app_name=APP_NAME, session_service

# Agent Interaction
async def call_agent(query):
  content = types.Content(role='user', parts=[types.Part(text=query)])
  events = runner.run_async(user_id=USER_ID, session_id=SESSION_ID, ne

  async for event in events:
      if event.is_final_response():
          final_response = event.content.parts[0].text
          print("Agent Response: ", final_response)

await call_agent("united states")
```

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.agents.InvocationContext;
import com.google.adk.events.Event;
import com.google.adk.runner.InMemoryRunner;
import com.google.adk.sessions.Session;
import com.google.adk.tools.Annotations.Schema;
import com.google.adk.tools.BaseTool;
import com.google.adk.tools.FunctionTool;
import com.google.adk.tools.ToolContext;
import com.google.common.collect.ImmutableMap;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;
import io.reactivex.rxjava3.core.Maybe;
import java.util.HashMap;
import java.util.Map;
```

```
public class AfterToolCallbackExample {

  private static final String APP_NAME = "AfterToolCallbackAgentApp";
  private static final String USER_ID = "user_1";
  private static final String SESSION_ID = "session_001";
  private static final String MODEL_NAME = "gemini-2.0-flash";

  public static void main(String[] args) {
    AfterToolCallbackExample example = new AfterToolCallbackExample();
    example.runAgent("What is the capital of the United States?");
  }

  // --- Define a Simple Tool Function (Same as before) ---
  @Schema(description = "Retrieves the capital city of a given country
  public static Map<String, Object> getCapitalCity(
      @Schema(description = "The country to find the capital of.") Str
    System.out.printf("--- Tool 'getCapitalCity' executing with countr
    Map<String, String> countryCapitals = new HashMap<>();
    countryCapitals.put("united states", "Washington, D.C.");
    countryCapitals.put("canada", "Ottawa");
    countryCapitals.put("france", "Paris");
    countryCapitals.put("germany", "Berlin");

    String capital =
        countryCapitals.getOrDefault(country.toLowerCase(), "Capital n
    return ImmutableMap.of("result", capital);
  }

  // Define the Callback function.
  public Maybe<Map<String, Object>> simpleAfterToolModifier(
      InvocationContext invocationContext,
      BaseTool tool,
      Map<String, Object> args,
      ToolContext toolContext,
      Object toolResponse) {
```

```
// Inspects/modifies the tool result after execution.
String agentName = invocationContext.agent().name();
String toolName = tool.name();
System.out.printf(
    "[Callback] After tool call for tool '%s' in agent '%s'%n", to
System.out.printf("[Callback] Args used: %s%n", args);
System.out.printf("[Callback] Original tool_response: %s%n", toolR

if (!(toolResponse instanceof Map)) {
  System.out.println("[Callback] toolResponse is not a Map, cannot
  // Pass through if not a map
  return Maybe.empty();
}

// Default structure for function tool results is {"result": <retu
@SuppressWarnings("unchecked")
Map<String, Object> responseMap = (Map<String, Object>) toolRespon
Object originalResultValue = responseMap.get("result");

// --- Modification Example ---
// If the tool was 'get_capital_city' and result is 'Washington, D
if ("getCapitalCity".equals(toolName) && "Washington, D.C.".equals
  System.out.println("[Callback] Detected 'Washington, D.C.'. Modi

  // IMPORTANT: Create a new mutable map or modify a copy
  Map<String, Object> modifiedResponse = new HashMap<>(responseMap
  modifiedResponse.put(
      "result", originalResultValue + " (Note: This is the capital
  modifiedResponse.put("note_added_by_callback", true); // Add ext

  System.out.printf("[Callback] Modified tool_response: %s%n", mod
  return Maybe.just(modifiedResponse);
}

System.out.println("[Callback] Passing original tool response thro
```

```
    // Return Maybe.empty() to use the original tool_response
    return Maybe.empty();
  }

  public void runAgent(String query) {
    // --- Wrap the function into a Tool ---
    FunctionTool capitalTool = FunctionTool.create(this.getClass(), "g

    // Create LlmAgent and Assign Callback
    LlmAgent myLlmAgent =
        LlmAgent.builder()
            .name(APP_NAME)
            .model(MODEL_NAME)
            .instruction(
                "You are an agent that finds capital cities using the
                    + " the result clearly.")
            .description("An LLM agent demonstrating after_tool_callba
            .tools(capitalTool) // Add the tool
            .afterToolCallback(this::simpleAfterToolModifier) // Assig
            .build();

    InMemoryRunner runner = new InMemoryRunner(myLlmAgent);

    // Session and Runner
    Session session =
        runner.sessionService().createSession(APP_NAME, USER_ID, null,

    Content userMessage = Content.fromParts(Part.fromText(query));

    System.out.printf("%n--- Calling agent with query: \"%s\" ---%n",
    Flowable<Event> eventStream = runner.runAsync(USER_ID, session.id(
    // Stream event response
    eventStream.blockingForEach(
        event -> {
          if (event.finalResponse()) {
            System.out.println(event.stringifyContent());
```

```
            }
        });
    }
}
```