# Callback patterns - Agent Development Kit

**Source URL:** https://google.github.io/adk-docs/callbacks/design-patterns-and-best-practices/

---

# Design Patterns and Best Practices for Callbacks¶

Callbacks offer powerful hooks into the agent lifecycle. Here are common design patterns illustrating how to leverage them effectively in ADK, followed by best practices for implementation.

## Design Patterns¶

These patterns demonstrate typical ways to enhance or control agent behavior using callbacks:

### 1. Guardrails & Policy Enforcement¶

- **Pattern:** Intercept requests before they reach the LLM or tools to enforce rules.
- **How:** Use `before_model_callback` to inspect the `LlmRequest` prompt or `before_tool_callback` to inspect tool arguments. If a policy violation is detected (e.g., forbidden topics, profanity), return a predefined response (`LlmResponse` or `dict` / `Map`) to block the operation and optionally update `context.state` to log the violation.
- **Example:** A `before_model_callback` checks `llm_request.contents` for sensitive keywords and returns a standard "Cannot process this request" `LlmResponse` if found, preventing the LLM call.

### 2. Dynamic State Management¶

- **Pattern:** Read from and write to session state within callbacks to make agent behavior context-aware and pass data between steps.

- **How:** Access `callback_context.state` or `tool_context.state`. Modifications (`state['key'] = value`) are automatically tracked in the subsequent `Event.actions.state_delta` for persistence by the `SessionService`.
- **Example:** An `after_tool_callback` saves a `transaction_id` from the tool's result to `tool_context.state['last_transaction_id']`. A later `before_agent_callback` might read `state['user_tier']` to customize the agent's greeting.

### 3. Logging and Monitoring¶

- **Pattern:** Add detailed logging at specific lifecycle points for observability and debugging.
- **How:** Implement callbacks (e.g., `before_agent_callback`, `after_tool_callback`, `after_model_callback`) to print or send structured logs containing information like agent name, tool name, invocation ID, and relevant data from the context or arguments.
- **Example:** Log messages like `INFO: [Invocation: e-123] Before Tool: search_api - Args: {'query': 'ADK'}`.

### 4. Caching¶

- **Pattern:** Avoid redundant LLM calls or tool executions by caching results.
- **How:** In `before_model_callback` or `before_tool_callback`, generate a cache key based on the request/arguments. Check `context.state` (or an external cache) for this key. If found, return the cached `LlmResponse` or result directly, skipping the actual operation. If not found, allow the operation to proceed and use the corresponding `after_` callback (`after_model_callback`, `after_tool_callback`) to store the new result in the cache using the key.
- **Example:** `before_tool_callback` for `get_stock_price(symbol)` checks `state[f"cache:stock:`

`{symbol}"]`. If present, returns the cached price; otherwise, allows the API call and `after_tool_callback` saves the result to the state key.

## 5. Request/Response Modification¶

- **Pattern:** Alter data just before it's sent to the LLM/tool or just after it's received.
- **How:**
- `before_model_callback`: Modify `llm_request` (e.g., add system instructions based on `state`).
- `after_model_callback`: Modify the returned `LlmResponse` (e.g., format text, filter content).
- `before_tool_callback`: Modify the tool `args` dictionary (or Map in Java).
- `after_tool_callback`: Modify the `tool_response` dictionary (or Map in Java).
- **Example:** `before_model_callback` appends "User language preference: Spanish" to `llm_request.config.system_instruction` if `context.state['lang'] == 'es'`.

## 6. Conditional Skipping of Steps¶

- **Pattern:** Prevent standard operations (agent run, LLM call, tool execution) based on certain conditions.
- **How:** Return a value from a `before_` callback (`Content` from `before_agent_callback`, `LlmResponse` from `before_model_callback`, `dict` from `before_tool_callback`). The framework interprets this returned value as the result for that step, skipping the normal execution.
- **Example:** `before_tool_callback` checks `tool_context.state['api_quota_exceeded']`. If `True`, it returns `{'error': 'API quota exceeded'}`, preventing the actual tool function from running.

## 7. Tool-Specific Actions (Authentication & Summarization Control)¶

- **Pattern:** Handle actions specific to the tool lifecycle, primarily authentication and controlling LLM summarization of tool results.
- **How:** Use `ToolContext` within tool callbacks (`before_tool_callback`, `after_tool_callback`).
- **Authentication:** Call `tool_context.request_credential(auth_config)` in `before_tool_callback` if credentials are required but not found (e.g., via `tool_context.get_auth_response` or state check). This initiates the auth flow.
- **Summarization:** Set `tool_context.actions.skip_summarization = True` if the raw dictionary output of the tool should be passed back to the LLM or potentially displayed directly, bypassing the default LLM summarization step.
- **Example:** A `before_tool_callback` for a secure API checks for an auth token in state; if missing, it calls `request_credential`. An `after_tool_callback` for a tool returning structured JSON might set `skip_summarization = True`.

## 8. Artifact Handling¶

- **Pattern:** Save or load session-related files or large data blobs during the agent lifecycle.
- **How:** Use `callback_context.save_artifact` / `await tool_context.save_artifact` to store data (e.g., generated reports, logs, intermediate data). Use `load_artifact` to retrieve previously stored artifacts. Changes are tracked via `Event.actions.artifact_delta`.
- **Example:** An `after_tool_callback` for a "generate_report" tool saves the output file using `await tool_context.save_artifact("report.pdf", report_part)`. A `before_agent_callback` might load a configuration artifact using `callback_context.load_artifact("agent_config.json")`.

## Best Practices for Callbacks¶

- **Keep Focused:** Design each callback for a single, well-defined purpose (e.g., just logging, just validation). Avoid monolithic callbacks.
- **Mind Performance:** Callbacks execute synchronously within the agent's processing loop. Avoid long-running or blocking operations (network calls, heavy computation). Offload if necessary, but be aware this adds complexity.
- **Handle Errors Gracefully:** Use `try...except/ catch` blocks within your callback functions. Log errors appropriately and decide if the agent invocation should halt or attempt recovery. Don't let callback errors crash the entire process.
- **Manage State Carefully:**
- Be deliberate about reading from and writing to `context.state`. Changes are immediately visible within the *current* invocation and persisted at the end of the event processing.
- Use specific state keys rather than modifying broad structures to avoid unintended side effects.
- Consider using state prefixes (`State.APP_PREFIX`, `State.USER_PREFIX`, `State.TEMP_PREFIX`) for clarity, especially with persistent `SessionService` implementations.
- **Consider Idempotency:** If a callback performs actions with external side effects (e.g., incrementing an external counter), design it to be idempotent (safe to run multiple times with the same input) if possible, to handle potential retries in the framework or your application.
- **Test Thoroughly:** Unit test your callback functions using mock context objects. Perform integration tests to ensure callbacks function correctly within the full agent flow.
- **Ensure Clarity:** Use descriptive names for your callback functions. Add clear docstrings explaining their purpose, when they run, and any side effects (especially state modifications).
- **Use Correct Context Type:** Always use the specific context type provided (`CallbackContext` for agent/model, `ToolContext` for tools) to ensure access to the appropriate methods and properties.

By applying these patterns and best practices, you can effectively use callbacks to create more robust, observable, and customized agent behaviors in ADK.