Built-in tools - Agent Development Kit

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

Built-in tools¶

These built-in tools provide ready-to-use functionality such as Google Search or code executors that provide agents with common capabilities. For instance, an agent that needs to retrieve information from the web can directly use the **google_search** tool without any additional setup.

How to Use¶

1. **Import:** Import the desired tool from the tools module. This is `agents.tools` in Python or `com.google.adk.tools` in Java.
2. **Configure:** Initialize the tool, providing required parameters if any.
3. **Register:** Add the initialized tool to the **tools** list of your Agent.

Once added to an agent, the agent can decide to use the tool based on the **user prompt** and its **instructions**. The framework handles the execution of the tool when the agent calls it. Important: check the ***Limitations*** section of this page.

Available Built-in tools¶

Note: Java only supports Google Search and Code Execution tools currently.

Google Search¶

The `google_search` tool allows the agent to perform web searches using Google Search. The `google_search` tool is only compatible with Gemini 2 models.

Additional requirements when using the `google_search` tool

When you use grounding with Google Search, and you receive Search suggestions in your response, you must display the Search suggestions in

production and in your applications. For more information on grounding with Google Search, see Grounding with Google Search documentation for Google AI Studio or Vertex AI. The UI code (HTML) is returned in the Gemini response as `renderedContent`, and you will need to show the HTML in your app, in accordance with the policy.

PythonJava

```
 from google.adk.agents import Agent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
from google.adk.tools import google_search
from google.genai import types

APP_NAME="google_search_agent"
USER_ID="user1234"
SESSION_ID="1234"

root_agent = Agent(
    name="basic_search_agent",
    model="gemini-2.0-flash",
    description="Agent to answer questions using Google Search.",
    instruction="I can answer your questions by searching the internet
    # google_search is a pre-built tool which allows the agent to perf
    tools=[google_search]
)

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

# Agent Interaction
def call_agent(query):
    """
    Helper function to call the agent with a query.
```

```
    """
    content = types.Content(role='user', parts=[types.Part(text=query)
    events = runner.run(user_id=USER_ID, session_id=SESSION_ID, new_me

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

call_agent("what's the latest ai news?")
```

```
import com.google.adk.agents.BaseAgent;
import com.google.adk.agents.LlmAgent;
import com.google.adk.runner.Runner;
import com.google.adk.sessions.InMemorySessionService;
import com.google.adk.sessions.Session;
import com.google.adk.tools.GoogleSearchTool;
import com.google.common.collect.ImmutableList;
import com.google.genai.types.Content;
import com.google.genai.types.Part;

public class GoogleSearchAgentApp {

  private static final String APP_NAME = "Google Search_agent";
  private static final String USER_ID = "user1234";
  private static final String SESSION_ID = "1234";

  /**
   * Calls the agent with the given query and prints the final respons
   *
   * @param runner The runner to use.
   * @param query The query to send to the agent.
   */
  public static void callAgent(Runner runner, String query) {
```

```
    Content content =
        Content.fromParts(Part.fromText(query));

    InMemorySessionService sessionService = (InMemorySessionService) r
    Session session =
        sessionService
            .createSession(APP_NAME, USER_ID, /* state= */ null, SESSI
            .blockingGet();

    runner
        .runAsync(session.userId(), session.id(), content)
        .forEach(
            event -> {
              if (event.finalResponse()
                  && event.content().isPresent()
                  && event.content().get().parts().isPresent()
                  && !event.content().get().parts().get().isEmpty()
                  && event.content().get().parts().get().get(0).text()
                String finalResponse = event.content().get().parts().g
                System.out.println("Agent Response: " + finalResponse)
              }
            });
  }

  public static void main(String[] args) {
    // Google Search is a pre-built tool which allows the agent to per
    GoogleSearchTool googleSearchTool = new GoogleSearchTool();

    BaseAgent rootAgent =
        LlmAgent.builder()
            .name("basic_search_agent")
            .model("gemini-2.0-flash") // Ensure to use a Gemini 2.0 m
            .description("Agent to answer questions using Google Searc
            .instruction(
                "I can answer your questions by searching the internet
            .tools(ImmutableList.of(googleSearchTool))
```

```
            .build();

    // Session and Runner
    InMemorySessionService sessionService = new InMemorySessionService
    Runner runner = new Runner(rootAgent, APP_NAME, null, sessionServi

    // Agent Interaction
    callAgent(runner, "what's the latest ai news?");
  }
}
```

Code Execution¶

The `built_in_code_execution` tool enables the agent to execute code, specifically when using Gemini 2 models. This allows the model to perform tasks like calculations, data manipulation, or running small scripts.

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.

import asyncio
from google.adk.agents import LlmAgent
from google.adk.runners import Runner
```

```
from google.adk.sessions import InMemorySessionService
from google.adk.code_executors import BuiltInCodeExecutor
from google.genai import types

AGENT_NAME = "calculator_agent"
APP_NAME = "calculator"
USER_ID = "user1234"
SESSION_ID = "session_code_exec_async"
GEMINI_MODEL = "gemini-2.0-flash"

# Agent Definition
code_agent = LlmAgent(
    name=AGENT_NAME,
    model=GEMINI_MODEL,
    executor=[BuiltInCodeExecutor],
    instruction="""You are a calculator agent.
    When given a mathematical expression, write and execute Python coo
    Return only the final numerical result as plain text, without mark
    """,
    description="Executes Python code to perform calculations.",
)

# Session and Runner
session_service = InMemorySessionService()
session = session_service.create_session(
    app_name=APP_NAME, user_id=USER_ID, session_id=SESSION_ID
)
runner = Runner(agent=code_agent, app_name=APP_NAME, session_service=s

# Agent Interaction (Async)
async def call_agent_async(query):
    content = types.Content(role="user", parts=[types.Part(text=query)
    print(f"\n--- Running Query: {query} ---")
    final_response_text = "No final text response captured."
    try:
        # Use run_async
```

```
        async for event in runner.run_async(
            user_id=USER_ID, session_id=SESSION_ID, new_message=conten
        ):
            print(f"Event ID: {event.id}, Author: {event.author}")

            # --- Check for specific parts FIRST ---
            has_specific_part = False
            if event.content and event.content.parts:
                for part in event.content.parts:  # Iterate through al
                    if part.executable_code:
                        # Access the actual code string via .code
                        print(
                            f"  Debug: Agent generated code:\n```pytho
                        )
                        has_specific_part = True
                    elif part.code_execution_result:
                        # Access outcome and output correctly
                        print(
                            f"  Debug: Code Execution Result: {part.co
                        )
                        has_specific_part = True
                    # Also print any text parts found in any event for
                    elif part.text and not part.text.isspace():
                        print(f"  Text: '{part.text.strip()}'")
                        # Do not set has_specific_part=True here, as w

            # --- Check for final response AFTER specific parts ---
            # Only consider it final if it doesn't have the specific c
            if not has_specific_part and event.is_final_response():
                if (
                    event.content
                    and event.content.parts
                    and event.content.parts[0].text
                ):
                    final_response_text = event.content.parts[0].text.
                    print(f"==> Final Agent Response: {final_response_
```

```
                else:
                    print("==> Final Agent Response: [No text content

    except Exception as e:
        print(f"ERROR during agent run: {e}")
    print("-" * 30)

# Main async function to run the examples
async def main():
    await call_agent_async("Calculate the value of (5 + 7) * 3")
    await call_agent_async("What is 10 factorial?")

# Execute the main async function
try:
    asyncio.run(main())
except RuntimeError as e:
    # Handle specific error when running asyncio.run in an already run
    if "cannot be called from a running event loop" in str(e):
        print("\nRunning in an existing event loop (like Colab/Jupyter
        print("Please run `await main()` in a notebook cell instead.")
        # If in an interactive environment like a notebook, you might
        # await main()
    else:
        raise e  # Re-raise other runtime errors
```

```
import com.google.adk.agents.BaseAgent;
import com.google.adk.agents.LlmAgent;
import com.google.adk.runner.Runner;
import com.google.adk.sessions.InMemorySessionService;
import com.google.adk.sessions.Session;
import com.google.adk.tools.BuiltInCodeExecutionTool;
import com.google.common.collect.ImmutableList;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
```

```
public class CodeExecutionAgentApp {

  private static final String AGENT_NAME = "calculator_agent";
  private static final String APP_NAME = "calculator";
  private static final String USER_ID = "user1234";
  private static final String SESSION_ID = "session_code_exec_sync";
  private static final String GEMINI_MODEL = "gemini-2.0-flash";

  /**
   * Calls the agent with a query and prints the interaction events an
   *
   * @param runner The runner instance for the agent.
   * @param query The query to send to the agent.
   */
  public static void callAgent(Runner runner, String query) {
    Content content =
        Content.builder().role("user").parts(ImmutableList.of(Part.fro

    InMemorySessionService sessionService = (InMemorySessionService) r
    Session session =
        sessionService
            .createSession(APP_NAME, USER_ID, /* state= */ null, SESSI
            .blockingGet();

    System.out.println("\n--- Running Query: " + query + " ---");
    final String[] finalResponseText = {"No final text response captur

    try {
      runner
          .runAsync(session.userId(), session.id(), content)
          .forEach(
              event -> {
                System.out.println("Event ID: " + event.id() + ", Auth

                boolean hasSpecificPart = false;
```

```
                if (event.content().isPresent() && event.content().get
                  for (Part part : event.content().get().parts().get()
                    if (part.executableCode().isPresent()) {
                      System.out.println(
                          "  Debug: Agent generated code:\n```python\n
                              + part.executableCode().get().code()
                              + "\n```");
                      hasSpecificPart = true;
                    } else if (part.codeExecutionResult().isPresent())
                      System.out.println(
                          "  Debug: Code Execution Result: "
                              + part.codeExecutionResult().get().outco
                              + " - Output:\n"
                              + part.codeExecutionResult().get().outpu
                      hasSpecificPart = true;
                    } else if (part.text().isPresent() && !part.text()
                      System.out.println("  Text: '" + part.text().get
                    }
                  }
                }

                if (!hasSpecificPart && event.finalResponse()) {
                  if (event.content().isPresent()
                      && event.content().get().parts().isPresent()
                      && !event.content().get().parts().get().isEmpty(
                      && event.content().get().parts().get().get(0).te
                    finalResponseText[0] =
                        event.content().get().parts().get().get(0).tex
                    System.out.println("==> Final Agent Response: " +
                  } else {
                    System.out.println(
                        "==> Final Agent Response: [No text content in
                  }
                }
              });
    } catch (Exception e) {
```

```
      System.err.println("ERROR during agent run: " + e.getMessage());
      e.printStackTrace();
    }
    System.out.println("------------------------------");
  }

  public static void main(String[] args) {
    BuiltInCodeExecutionTool codeExecutionTool = new BuiltInCodeExecut

    BaseAgent codeAgent =
        LlmAgent.builder()
            .name(AGENT_NAME)
            .model(GEMINI_MODEL)
            .tools(ImmutableList.of(codeExecutionTool))
            .instruction(
                """
                                You are a calculator agent.
                                When given a mathematical expression,
                                Return only the final numerical result
                                """)
            .description("Executes Python code to perform calculations
            .build();

    InMemorySessionService sessionService = new InMemorySessionService
    Runner runner = new Runner(codeAgent, APP_NAME, null, sessionServi

    callAgent(runner, "Calculate the value of (5 + 7) * 3");
    callAgent(runner, "What is 10 factorial?");
  }
}
```

Vertex AI Search¶

The `vertex_ai_search_tool` uses Google Cloud's Vertex AI Search, enabling the agent to search across your private, configured data stores (e.g.,

internal documents, company policies, knowledge bases). This built-in tool requires you to provide the specific data store ID during configuration.

```
import asyncio

from google.adk.agents import LlmAgent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
from google.genai import types
from google.adk.tools import VertexAiSearchTool

# Replace with your actual Vertex AI Search Datastore ID
# Format: projects/<PROJECT_ID>/locations/<LOCATION>/collections/defau
# e.g., "projects/12345/locations/us-central1/collections/default_coll
YOUR_DATASTORE_ID = "YOUR_DATASTORE_ID_HERE"

# Constants
APP_NAME_VSEARCH = "vertex_search_app"
USER_ID_VSEARCH = "user_vsearch_1"
SESSION_ID_VSEARCH = "session_vsearch_1"
AGENT_NAME_VSEARCH = "doc_qa_agent"
GEMINI_2_FLASH = "gemini-2.0-flash"

# Tool Instantiation
# You MUST provide your datastore ID here.
vertex_search_tool = VertexAiSearchTool(data_store_id=YOUR_DATASTORE_I

# Agent Definition
doc_qa_agent = LlmAgent(
    name=AGENT_NAME_VSEARCH,
    model=GEMINI_2_FLASH, # Requires Gemini model
    tools=[vertex_search_tool],
    instruction=f"""You are a helpful assistant that answers questions
    Use the search tool to find relevant information before answering.
    If the answer isn't in the documents, say that you couldn't find t
    """,
```

```
    description="Answers questions using a specific Vertex AI Search d
)

# Session and Runner Setup
session_service_vsearch = InMemorySessionService()
runner_vsearch = Runner(
    agent=doc_qa_agent, app_name=APP_NAME_VSEARCH, session_service=ses
)
session_vsearch = session_service_vsearch.create_session(
    app_name=APP_NAME_VSEARCH, user_id=USER_ID_VSEARCH, session_id=SES
)

# Agent Interaction Function
async def call_vsearch_agent_async(query):
    print("\n--- Running Vertex AI Search Agent ---")
    print(f"Query: {query}")
    if "YOUR_DATASTORE_ID_HERE" in YOUR_DATASTORE_ID:
        print("Skipping execution: Please replace YOUR_DATASTORE_ID_HE
        print("-" * 30)
        return

    content = types.Content(role='user', parts=[types.Part(text=query)
    final_response_text = "No response received."
    try:
        async for event in runner_vsearch.run_async(
            user_id=USER_ID_VSEARCH, session_id=SESSION_ID_VSEARCH, ne
        ):
            # Like Google Search, results are often embedded in the mo
            if event.is_final_response() and event.content and event.c
                final_response_text = event.content.parts[0].text.stri
                print(f"Agent Response: {final_response_text}")
                # You can inspect event.grounding_metadata for source
                if event.grounding_metadata:
                    print(f"  (Grounding metadata found with {len(even

    except Exception as e:
```

```
        print(f"An error occurred: {e}")
        print("Ensure your datastore ID is correct and the service acc
    print("-" * 30)

# --- Run Example ---
async def run_vsearch_example():
    # Replace with a question relevant to YOUR datastore content
    await call_vsearch_agent_async("Summarize the main points about th
    await call_vsearch_agent_async("What safety procedures are mentior

# Execute the example
# await run_vsearch_example()

# Running locally due to potential colab asyncio issues with multiple
try:
    asyncio.run(run_vsearch_example())
except RuntimeError as e:
    if "cannot be called from a running event loop" in str(e):
        print("Skipping execution in running event loop (like Colab/Ju
    else:
        raise e
```

Use Built-in tools with other tools¶

The following code sample demonstrates how to use multiple built-in tools or how to use built-in tools with other tools by using multiple agents:

PythonJava

```
from google.adk.tools import agent_tool
from google.adk.agents import Agent
from google.adk.tools import google_search
from google.adk.code_executors import BuiltInCodeExecutor

search_agent = Agent(
```

```
    model='gemini-2.0-flash',
    name='SearchAgent',
    instruction="""
    You're a specialist in Google Search
    """,
    tools=[google_search],
)
coding_agent = Agent(
    model='gemini-2.0-flash',
    name='CodeAgent',
    instruction="""
    You're a specialist in Code Execution
    """,
    code_executor=[BuiltInCodeExecutor],
)
root_agent = Agent(
    name="RootAgent",
    model="gemini-2.0-flash",
    description="Root Agent",
    tools=[agent_tool.AgentTool(agent=search_agent), agent_tool.AgentT
)
```

```
import com.google.adk.agents.BaseAgent;
import com.google.adk.agents.LlmAgent;
import com.google.adk.tools.AgentTool;
import com.google.adk.tools.BuiltInCodeExecutionTool;
import com.google.adk.tools.GoogleSearchTool;
import com.google.common.collect.ImmutableList;

public class NestedAgentApp {

  private static final String MODEL_ID = "gemini-2.0-flash";

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

```
// Define the SearchAgent
LlmAgent searchAgent =
    LlmAgent.builder()
        .model(MODEL_ID)
        .name("SearchAgent")
        .instruction("You're a specialist in Google Search")
        .tools(new GoogleSearchTool()) // Instantiate GoogleSearch
        .build();

// Define the CodingAgent
LlmAgent codingAgent =
    LlmAgent.builder()
        .model(MODEL_ID)
        .name("CodeAgent")
        .instruction("You're a specialist in Code Execution")
        .tools(new BuiltInCodeExecutionTool()) // Instantiate Buil
        .build();

// Define the RootAgent, which uses AgentTool.create() to wrap Sea
BaseAgent rootAgent =
    LlmAgent.builder()
        .name("RootAgent")
        .model(MODEL_ID)
        .description("Root Agent")
        .tools(
            AgentTool.create(searchAgent), // Use create method
            AgentTool.create(codingAgent)   // Use create method
         )
        .build();

// Note: This sample only demonstrates the agent definitions.
// To run these agents, you'd need to integrate them with a Runner
// similar to the previous examples.
System.out.println("Agents defined successfully:");
System.out.println("  Root Agent: " + rootAgent.name());
```

```
    System.out.println("  Search Agent (nested): " + searchAgent.name(
    System.out.println("  Code Agent (nested): " + codingAgent.name())
  }
}
```

Limitations¶

Warning

Currently, for each root agent or single agent, only one built-in tool is supported. No other tools of any type can be used in the same agent.

For example, the following approach that uses ***a built-in tool along with other tools*** within a single agent is **not** currently supported:

PythonJava

```
root_agent = Agent(
    name="RootAgent",
    model="gemini-2.0-flash",
    description="Root Agent",
    tools=[custom_function],
    executor=[BuiltInCodeExecutor] # <-- not supported when used with
)
```

```
 LlmAgent searchAgent =
        LlmAgent.builder()
            .model(MODEL_ID)
            .name("SearchAgent")
            .instruction("You're a specialist in Google Search")
            .tools(new GoogleSearchTool(), new YourCustomTool()) // <-
            .build();
```

Warning

Built-in tools cannot be used within a sub-agent.

For example, the following approach that uses built-in tools within sub-agents is **not** currently supported:

PythonJava

```
search_agent = Agent(
    model='gemini-2.0-flash',
    name='SearchAgent',
    instruction="""
    You're a specialist in Google Search
    """,
    tools=[google_search],
)
coding_agent = Agent(
    model='gemini-2.0-flash',
    name='CodeAgent',
    instruction="""
    You're a specialist in Code Execution
    """,
    executor=[BuiltInCodeExecutor],
)
root_agent = Agent(
    name="RootAgent",
    model="gemini-2.0-flash",
    description="Root Agent",
    sub_agents=[
        search_agent,
        coding_agent
    ],
)
```

```
LlmAgent searchAgent =
    LlmAgent.builder()
```

```
        .model("gemini-2.0-flash")
        .name("SearchAgent")
        .instruction("You're a specialist in Google Search")
        .tools(new GoogleSearchTool())
        .build();

LlmAgent codingAgent =
    LlmAgent.builder()
        .model("gemini-2.0-flash")
        .name("CodeAgent")
        .instruction("You're a specialist in Code Execution")
        .tools(new BuiltInCodeExecutionTool())
        .build();

LlmAgent rootAgent =
    LlmAgent.builder()
        .name("RootAgent")
        .model("gemini-2.0-flash")
        .description("Root Agent")
        .subAgents(searchAgent, codingAgent) // Not supported, as the
        .build();
```