Built-in tools

These are ready-to-use functionalities, like Google Search, Code Executors, that provide agents with common capabilities immediately. For instance, an agent needing to retrieve information from the web can directly use the **built_in_google_search** tool without any additional setup.

How to Use

1. **Import:** Import the desired tool from the `agents.tools` module.
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.

Available Built-in tools

Google Search

`google_search`: Allows the agent to perform web searches using Google Search. You simply add this tool to the agent's tools list. It is compatible with Gemini 2 models.

```
from google.adk.agents import Agent
```

```python
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-exp",
    description="Agent to answer questions using Google Search.",
    instruction="I can answer your questions by searching the internet. Just ask me anything!",
    # google_search is a pre-built tool which allows the agent to perform Google searches.
    tools=[google_search]
)

# 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=root_agent, app_name=APP_NAME, session_service=session_service)


# 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_message=content)

    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?")
```

Code Execution

`built_in_code_execution`: 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. This capability is built-in and automatically activated for compatible models, requiring no explicit configuration.

```
import asyncio
from google.adk.agents import LlmAgent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
from google.adk.tools import built_in_code_execution
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-exp"

# Agent Definition
code_agent = LlmAgent(
    name=AGENT_NAME,
    model=GEMINI_MODEL,
    tools=[built_in_code_execution],
    instruction="""You are a calculator agent.
    When given a mathematical expression, write and execute Python code to calculate the result.
    Return only the final numerical result as plain text, without markdown or code blocks.
    """,
    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=session_service)

# 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=content):
            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 all parts
                    if part.executable_code:
                        # Access the actual code string via .code
                        print(f"  Debug: Agent generated
code:\n```python\n{part.executable_code.code}\n```")
                        has_specific_part = True
                    elif part.code_execution_result:
                        # Access outcome and output correctly
                        print(f"  Debug: Code Execution Result:
{part.code_execution_result.outcome} - Output:\n{part.code_execution_result.output}")
                        has_specific_part = True
                    # Also print any text parts found in any event for debugging
                    elif part.text and not part.text.isspace():
                        print(f"  Text: '{part.text.strip()}'")
                        # Do not set has_specific_part=True here, as we want the final response
logic below

            # --- Check for final response AFTER specific parts ---
```

```
            # Only consider it final if it doesn't have the specific code parts we just handled
            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.strip()
                    print(f"==> Final Agent Response: {final_response_text}")
                else:
                    print("==> Final Agent Response: [No text content in final event]")


    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 running loop (like
Jupyter/Colab)
    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 need to run:
        # await main()
    else:
        raise e # Re-raise other runtime errors
```

Retrieval tools

This is a category of tools designed to fetch information from various sources. Examples include:

Vertex AI Search

`built_in_vertexai_search`: Leverages 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/default_collection/dataStores/<DATASTORE_ID>
# e.g., "projects/12345/locations/us-central1/collections/default_collection/dataStores/my-
datastore-123"
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-exp"

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

# 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 based on information found
in the document store: {YOUR_DATASTORE_ID}.
    Use the search tool to find relevant information before answering.
    If the answer isn't in the documents, say that you couldn't find the information.
    """,
    description="Answers questions using a specific Vertex AI Search datastore.",
)

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

# 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_HERE with your actual datastore
ID.")
        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, new_message=content
        ):
```

```
            # Like Google Search, results are often embedded in the model's response.
            if event.is_final_response() and event.content and event.content.parts:
                final_response_text = event.content.parts[0].text.strip()
                print(f"Agent Response: {final_response_text}")
                # You can inspect event.grounding_metadata for source citations
                if event.grounding_metadata:
                    print(f"  (Grounding metadata found with
{len(event.grounding_metadata.grounding_attributions)} attributions)")

    except Exception as e:
        print(f"An error occurred: {e}")
        print("Ensure your datastore ID is correct and the service account has permissions.")
    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 the Q2 strategy document.")
    await call_vsearch_agent_async("What safety procedures are mentioned for lab X?")

# Execute the example
# await run_vsearch_example()

# Running locally due to potential colab asyncio issues with multiple awaits
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/Jupyter). Run locally.")
    else:
        raise e
```