Quickstart - Agent Development Kit

Source URL: https://google.github.io/adk-docs/get-started/quickstart/

Quickstart¶

This quickstart guides you through installing the Agent Development Kit (ADK), setting up a basic agent with multiple tools, and running it locally either in the terminal or in the interactive, browser-based dev UI.

This quickstart assumes a local IDE (VS Code, PyCharm, IntelliJ IDEA, etc.) with Python 3.9+ or Java 17+ and terminal access. This method runs the application entirely on your machine and is recommended for internal development.

1. Set up Environment & Install ADK¶

PythonJava

Create & Activate Virtual Environment (Recommended):

```
 # Create
python -m venv .venv
# Activate (each new terminal)
# macOS/Linux: source .venv/bin/activate
# Windows CMD: .venv\Scripts\activate.bat
# Windows PowerShell: .venv\Scripts\Activate.ps1
```

Install ADK:

```
pip install google-adk
```

To install ADK and setup the environment, proceed to the following steps.

2. Create Agent Project¶

Project structure¶

PythonJava

You will need to create the following project structure:

```
parent_folder/
    multi_tool_agent/
        __init__.py
        agent.py
        .env
```

Create the folder `multi_tool_agent`:

```
mkdir multi_tool_agent/
```

Note for Windows users

When using ADK on Windows for the next few steps, we recommend creating Python files using File Explorer or an IDE because the following commands (`mkdir`, `echo`) typically generate files with null bytes and/or incorrect encoding.

`__init__.py`¶

Now create an `__init__.py` file in the folder:

```
echo "from . import agent" > multi_tool_agent/__init__.py
```

Your `__init__.py` should now look like this:

multi_tool_agent/__init__.py

```
from . import agent
```

`agent.py` ¶

Create an `agent.py` file in the same folder:

```
touch multi_tool_agent/agent.py
```

Copy and paste the following code into `agent.py`:

multi_tool_agent/agent.py

```
import datetime
from zoneinfo import ZoneInfo
from google.adk.agents import Agent

def get_weather(city: str) -> dict:
    """Retrieves the current weather report for a specified city.

    Args:
        city (str): The name of the city for which to retrieve the wea

    Returns:
        dict: status and result or error msg.
    """
    if city.lower() == "new york":
        return {
            "status": "success",
            "report": (
                "The weather in New York is sunny with a temperature c
                " Celsius (77 degrees Fahrenheit)."
            ),
        }
    else:
```

```
        return {
            "status": "error",
            "error_message": f"Weather information for '{city}' is not
        }

def get_current_time(city: str) -> dict:
    """Returns the current time in a specified city.

    Args:
        city (str): The name of the city for which to retrieve the cur

    Returns:
        dict: status and result or error msg.
    """

    if city.lower() == "new york":
        tz_identifier = "America/New_York"
    else:
        return {
            "status": "error",
            "error_message": (
                f"Sorry, I don't have timezone information for {city}.
            ),
        }

    tz = ZoneInfo(tz_identifier)
    now = datetime.datetime.now(tz)
    report = (
        f'The current time in {city} is {now.strftime("%Y-%m-%d %H:%M:
    )
    return {"status": "success", "report": report}

root_agent = Agent(
    name="weather_time_agent",
    model="gemini-2.0-flash",
    description=(
```

```
        "Agent to answer questions about the time and weather in a cit
    ),
    instruction=(
        "You are a helpful agent who can answer user questions about t
    ),
    tools=[get_weather, get_current_time],
)
```

`.env` ¶

Create a `.env` file in the same folder:

```
touch multi_tool_agent/.env
```

More instructions about this file are described in the next section on Set up the model.

Java projects generally feature the following project structure:

```
project_folder/
├── pom.xml (or build.gradle)
├── src/
├── └── main/
│       └── java/
│           └── agents/
│               └── multitool/
└── test/
```

Create `MultiToolAgent.java` ¶

Create a `MultiToolAgent.java` source file in the `agents.multitool` package in the `src/main/java/agents/multitool/` directory.

Copy and paste the following code into `MultiToolAgent.java`:

agents/multitool/MultiToolAgent.java

```
package agents.multitool;

import com.google.adk.agents.BaseAgent;
import com.google.adk.agents.LlmAgent;
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.FunctionTool;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;
import java.nio.charset.StandardCharsets;
import java.text.Normalizer;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Map;
import java.util.Scanner;

public class MultiToolAgent {

    private static String USER_ID = "student";
    private static String NAME = "multi_tool_agent";

    // The run your agent with Dev UI, the ROOT_AGENT should be a glob
    public static BaseAgent ROOT_AGENT = initAgent();

    public static BaseAgent initAgent() {
        return LlmAgent.builder()
            .name(NAME)
            .model("gemini-2.0-flash")
            .description("Agent to answer questions about the time and
            .instruction(
```

```
                "You are a helpful agent who can answer user questions
                    + " in a city.")
            .tools(
                FunctionTool.create(MultiToolAgent.class, "getCurrentT
                FunctionTool.create(MultiToolAgent.class, "getWeather"
            .build();
    }

    public static Map<String, String> getCurrentTime(
        @Schema(description = "The name of the city for which to retri
        String city) {
        String normalizedCity =
            Normalizer.normalize(city, Normalizer.Form.NFD)
                .trim()
                .toLowerCase()
                .replaceAll("(\\p{IsM}+|\\p{IsP}+)", "")
                .replaceAll("\\s+", "_");

        return ZoneId.getAvailableZoneIds().stream()
            .filter(zid -> zid.toLowerCase().endsWith("/" + normalized
            .findFirst()
            .map(
                zid ->
                    Map.of(
                        "status",
                        "success",
                        "report",
                        "The current time in "
                            + city
                            + " is "
                            + ZonedDateTime.now(ZoneId.of(zid))
                            .format(DateTimeFormatter.ofPattern("HH:mm
                            + "."))
            .orElse(
                Map.of(
                    "status",
```

```
                    "error",
                    "report",
                    "Sorry, I don't have timezone information for " +
    }

    public static Map<String, String> getWeather(
        @Schema(description = "The name of the city for which to retri
        String city) {
        if (city.toLowerCase().equals("new york")) {
            return Map.of(
                "status",
                "success",
                "report",
                "The weather in New York is sunny with a temperature o
                    + " Fahrenheit).");

        } else {
            return Map.of(
                "status", "error", "report", "Weather information for
        }
    }

    public static void main(String[] args) throws Exception {
        InMemoryRunner runner = new InMemoryRunner(ROOT_AGENT);

        Session session =
            runner
                .sessionService()
                .createSession(NAME, USER_ID)
                .blockingGet();

        try (Scanner scanner = new Scanner(System.in, StandardCharsets
            while (true) {
                System.out.print("\nYou > ");
                String userInput = scanner.nextLine();
```

```
                if ("quit".equalsIgnoreCase(userInput)) {
                    break;
                }

                Content userMsg = Content.fromParts(Part.fromText(user
                Flowable<Event> events = runner.runAsync(USER_ID, sess

                System.out.print("\nAgent > ");
                events.blockingForEach(event -> System.out.println(eve
            }
        }
    }
}
```

intro_components.png

3. Set up the model¶

Your agent's ability to understand user requests and generate responses is powered by a Large Language Model (LLM). Your agent needs to make secure calls to this external LLM service, which requires authentication credentials. Without valid authentication, the LLM service will deny the agent's requests, and the agent will be unable to function.

Gemini - Google AI StudioGemini - Google Cloud Vertex AI

1. Get an API key from Google AI Studio.
2. When using Python, open the **`.env`** file located inside (`multi_tool_agent/`) and copy-paste the following code.

multi_tool_agent/.env

``` GOOGLE_GENAI_USE_VERTEXAI=FALSE
GOOGLE_API_KEY=PASTE_YOUR_ACTUAL_API_KEY_HERE

```

When using Java, define environment variables:

terminal

``` export GOOGLE_GENAI_USE_VERTEXAI=FALSE export GOOGLE_API_KEY=PASTE_YOUR_ACTUAL_API_KEY_HERE

`` 3. Replace PASTE_YOUR_ACTUAL_API_KEY_HERE with your actual API KEY`.

1. You need an existing Google Cloud account and a project.
2. Set up a Google Cloud project
3. Set up the gcloud CLI
4. Authenticate to Google Cloud, from the terminal by running `gcloud auth login`.
5. Enable the Vertex AI API.
6. When using Python, open the **`.env`** file located inside (`multi_tool_agent/`). Copy-paste the following code and update the project ID and location.

multi_tool_agent/.env

``` GOOGLE_GENAI_USE_VERTEXAI=TRUE GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID GOOGLE_CLOUD_LOCATION=LOCATION

```

When using Java, define environment variables:

terminal

``` export GOOGLE_GENAI_USE_VERTEXAI=TRUE export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID export GOOGLE_CLOUD_LOCATION=LOCATION

```

4. Run Your Agent¶

PythonJava

Using the terminal, navigate to the parent directory of your agent project (e.g. using `cd ..`):
```

```
parent_folder/      <-- navigate to this directory
    multi_tool_agent/
        __init__.py
        agent.py
        .env
```

There are multiple ways to interact with your agent:

Dev UI (adk web)Terminal (adk run)API Server (adk api_server)

Run the following command to launch the **dev UI**.

```
adk web
```

Note for Windows users

When hitting the `_make_subprocess_transport NotImplementedError`, consider using `adk web --no-reload` instead.

Step 1: Open the URL provided (usually `http://localhost:8000` or `http://127.0.0.1:8000`) directly in your browser.

Step 2. In the top-left corner of the UI, you can select your agent in the dropdown. Select "multi_tool_agent".

Troubleshooting

If you do not see "multi_tool_agent" in the dropdown menu, make sure you are running `adk web` in the **parent folder** of your agent folder (i.e. the parent folder of multi_tool_agent).

Step 3. Now you can chat with your agent using the textbox:

adk-web-dev-ui-chat.png

Step 4. By using the `Events` tab at the left, you can inspect individual function calls, responses and model responses by clicking on the actions:

adk-web-dev-ui-function-call.png

On the `Events` tab, you can also click the `Trace` button to see the trace logs for each event that shows the latency of each function calls:

adk-web-dev-ui-trace.png

Step 5. You can also enable your microphone and talk to your agent:

Model support for voice/video streaming

In order to use voice/video streaming in ADK, you will need to use Gemini models that support the Live API. You can find the **model ID(s)** that supports the Gemini Live API in the documentation:

- Google AI Studio: Gemini Live API
- Vertex AI: Gemini Live API

You can then replace the `model` string in `root_agent` in the `agent.py` file you created earlier (jump to section). Your code should look something like:

```
root_agent = Agent(
    name="weather_time_agent",
    model="replace-me-with-model-id", #e.g. gemini-2.0-flash-live-001
    ...
```

adk-web-dev-ui-audio.png

Run the following command, to chat with your Weather agent.

```
adk run multi_tool_agent
```

adk-run.png

To exit, use Cmd/Ctrl+C.

`adk api_server` enables you to create a local FastAPI server in a single command, enabling you to test local cURL requests before you deploy your agent.

adk-api-server.png

To learn how to use `adk api_server` for testing, refer to the documentation on testing.

Using the terminal, navigate to the parent directory of your agent project (e.g. using `cd ..`):

```
project_folder/                <-- navigate to this directory
├── pom.xml (or build.gradle)
├── src/
├── └── main/
│       └── java/
│           └── agents/
│               └── multitool/
│                   └── MultiToolAgent.java
└── test/
```

Dev UIMavenGradle

Run the following command from the terminal to launch the Dev UI.

DO NOT change the main class name of the Dev UI server.

terminal

```
mvn exec:java \
    -Dexec.mainClass="com.google.adk.web.AdkWebServer" \
    -Dexec.args="--adk.agents.source-dir=src/main/java" \
    -Dexec.classpathScope="compile"
```

Step 1: Open the URL provided (usually `http://localhost:8080` or `http://127.0.0.1:8080`) directly in your browser.

Step 2. In the top-left corner of the UI, you can select your agent in the dropdown. Select "multi_tool_agent".

Troubleshooting

If you do not see "multi_tool_agent" in the dropdown menu, make sure you are running the `mvn` command at the location where your Java source code is located (usually `src/main/java`).

Step 3. Now you can chat with your agent using the textbox:

adk-web-dev-ui-chat.png

Step 4. You can also inspect individual function calls, responses and model responses by clicking on the actions:

adk-web-dev-ui-function-call.png

With Maven, run the `main()` method of your Java class with the following command:

terminal

```
mvn compile exec:java -Dexec.mainClass="agents.multitool.MultiToolAgen
```

With Gradle, the `build.gradle` or `build.gradle.kts` build file should have the following Java plugin in its `plugins` section:

```
plugins {
    id("java")
    // other plugins
}
```

Then, elsewhere in the build file, at the top-level, create a new task to run the `main()` method of your agent:

```
task runAgent(type: JavaExec) {
    classpath = sourceSets.main.runtimeClasspath
    mainClass = "agents.multitool.MultiToolAgent"
```

```
}
```

Finally, on the command-line, run the following command:

```
gradle runAgent
```

📝 Example prompts to try¶

- What is the weather in New York?
- What is the time in New York?
- What is the weather in Paris?
- What is the time in Paris?

🎉 Congratulations¶

You've successfully created and interacted with your first agent using ADK!

🛣️ Next steps¶

- **Go to the tutorial**: Learn how to add memory, session, state to your agent: tutorial.
- **Delve into advanced configuration:** Explore the setup section for deeper dives into project structure, configuration, and other interfaces.
- **Understand Core Concepts:** Learn about agents concepts.