OpenAPI tools - Agent Development Kit

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

OpenAPI Integration¶

Currently supported in Python

Integrating REST APIs with OpenAPI¶

ADK simplifies interacting with external REST APIs by automatically generating callable tools directly from an OpenAPI Specification (v3.x). This eliminates the need to manually define individual function tools for each API endpoint.

Core Benefit

Use `OpenAPIToolset` to instantly create agent tools (`RestApiTool`) from your existing API documentation (OpenAPI spec), enabling agents to seamlessly call your web services.

Key Components¶

- **`OpenAPIToolset`**: This is the primary class you'll use. You initialize it with your OpenAPI specification, and it handles the parsing and generation of tools.
- **`RestApiTool`**: This class represents a single, callable API operation (like `GET /pets/{petId}` or `POST /pets`). `OpenAPIToolset` creates one `RestApiTool` instance for each operation defined in your spec.

How it Works¶

The process involves these main steps when you use `OpenAPIToolset`:

1. **Initialization & Parsing**:

2. You provide the OpenAPI specification to `OpenAPIToolset` either as a Python dictionary, a JSON string, or a YAML string.
3. The toolset internally parses the spec, resolving any internal references (`$ref`) to understand the complete API structure.
4. **Operation Discovery**:
5. It identifies all valid API operations (e.g., `GET`, `POST`, `PUT`, `DELETE`) defined within the `paths` object of your specification.
6. **Tool Generation**:
7. For each discovered operation, `OpenAPIToolset` automatically creates a corresponding `RestApiTool` instance.
8. **Tool Name**: Derived from the `operationId` in the spec (converted to `snake_case`, max 60 chars). If `operationId` is missing, a name is generated from the method and path.
9. **Tool Description**: Uses the `summary` or `description` from the operation for the LLM.
10. **API Details**: Stores the required HTTP method, path, server base URL, parameters (path, query, header, cookie), and request body schema internally.
11. **`RestApiTool` Functionality**: Each generated `RestApiTool`:
12. **Schema Generation**: Dynamically creates a `FunctionDeclaration` based on the operation's parameters and request body. This schema tells the LLM how to call the tool (what arguments are expected).
13. **Execution**: When called by the LLM, it constructs the correct HTTP request (URL, headers, query params, body) using the arguments provided by the LLM and the details from the OpenAPI spec. It handles authentication (if configured) and executes the API call using the `requests` library.
14. **Response Handling**: Returns the API response (typically JSON) back to the agent flow.
15. **Authentication**: You can configure global authentication (like API keys or OAuth - see Authentication for details) when initializing

`OpenAPIToolset`. This authentication configuration is automatically applied to all generated `RestApiTool` instances.

Usage Workflow¶

Follow these steps to integrate an OpenAPI spec into your agent:

1. **Obtain Spec**: Get your OpenAPI specification document (e.g., load from a `.json` or `.yaml` file, fetch from a URL).
2. **Instantiate Toolset**: Create an `OpenAPIToolset` instance, passing the spec content and type (`spec_str` / `spec_dict`, `spec_str_type`). Provide authentication details (`auth_scheme`, `auth_credential`) if required by the API.

``` from google.adk.tools.openapi_tool.openapi_spec_parser.openapi_toolset import OpenAPIToolset

# Example with a JSON string openapi_spec_json = '...' # Your OpenAPI JSON string toolset = OpenAPIToolset(spec_str=openapi_spec_json, spec_str_type="json")

# Example with a dictionary # openapi_spec_dict = {...} # Your OpenAPI spec as a dict # toolset = OpenAPIToolset(spec_dict=openapi_spec_dict)

`` 3. **Add to Agent**: Include the retrieved tools in your LlmAgent's tools` list.

``` from google.adk.agents import LlmAgent

my_agent = LlmAgent( name="api_interacting_agent", model="gemini-2.0-flash", # Or your preferred model tools=[toolset], # Pass the toolset # ... other agent config ... )

`` 4. **Instruct Agent**: Update your agent's instructions to inform it about the new API capabilities and the names of the tools it can use (e.g., list_pets, create_pet). The tool descriptions generated from the spec will also help the LLM. 5. **Run Agent**: Execute your agent using the Runner. When the LLM determines it needs to call one of the APIs, it will generate a function call targeting the

`appropriate` RestApiTool`, which will then handle the HTTP request automatically.

## Example¶

This example demonstrates generating tools from a simple Pet Store OpenAPI spec (using `httpbin.org` for mock responses) and interacting with them via an agent.

Code: Pet Store API

openapi_example.py

```
 # 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
import uuid # For unique session IDs
from dotenv import load_dotenv

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

--- OpenAPI Tool Imports ---
```

```
from google.adk.tools.openapi_tool.openapi_spec_parser.openapi_toolset

# --- Load Environment Variables (If ADK tools need them, e.g., API ke
load_dotenv() # Create a .env file in the same directory if needed

# --- Constants ---
APP_NAME_OPENAPI = "openapi_petstore_app"
USER_ID_OPENAPI = "user_openapi_1"
SESSION_ID_OPENAPI = f"session_openapi_{uuid.uuid4()}" # Unique sessio
AGENT_NAME_OPENAPI = "petstore_manager_agent"
GEMINI_MODEL = "gemini-2.0-flash"

# --- Sample OpenAPI Specification (JSON String) ---
# A basic Pet Store API example using httpbin.org as a mock server
openapi_spec_string = """
{
  "openapi": "3.0.0",
  "info": {
    "title": "Simple Pet Store API (Mock)",
    "version": "1.0.1",
    "description": "An API to manage pets in a store, using httpbin fo
  },
  "servers": [
    {
      "url": "https://httpbin.org",
      "description": "Mock server (httpbin.org)"
    }
  ],
  "paths": {
    "/get": {
      "get": {
        "summary": "List all pets (Simulated)",
        "operationId": "listPets",
        "description": "Simulates returning a list of pets. Uses httpb
        "parameters": [
          {
```

```
          "name": "limit",
          "in": "query",
          "description": "Maximum number of pets to return",
          "required": false,
          "schema": { "type": "integer", "format": "int32" }
        },
        {
           "name": "status",
           "in": "query",
           "description": "Filter pets by status",
           "required": false,
           "schema": { "type": "string", "enum": ["available", "pend
        }
      ],
      "responses": {
        "200": {
          "description": "A list of pets (echoed query params).",
          "content": { "application/json": { "schema": { "type": "ob
        }
      }
    }
  },
  "/post": {
    "post": {
      "summary": "Create a pet (Simulated)",
      "operationId": "createPet",
      "description": "Simulates adding a new pet. Uses httpbin's /po
      "requestBody": {
        "description": "Pet object to add",
        "required": true,
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "required": ["name"],
              "properties": {
```

```
                "name": {"type": "string", "description": "Name of t
                "tag": {"type": "string", "description": "Optional t
              }
            }
          }
        }
      },
      "responses": {
        "201": {
          "description": "Pet created successfully (echoed request b
          "content": { "application/json": { "schema": { "type": "ob
        }
      }
    }
  },
  "/get?petId={petId}": {
    "get": {
      "summary": "Info for a specific pet (Simulated)",
      "operationId": "showPetById",
      "description": "Simulates returning info for a pet ID. Uses ht
      "parameters": [
        {
          "name": "petId",
          "in": "path",
          "description": "This is actually passed as a query param t
          "required": true,
          "schema": { "type": "integer", "format": "int64" }
        }
      ],
      "responses": {
        "200": {
          "description": "Information about the pet (echoed query pa
          "content": { "application/json": { "schema": { "type": "ob
        },
        "404": { "description": "Pet not found (simulated)" }
      }
```

```
      }
    }
  }
}
"""

# --- Create OpenAPIToolset ---
petstore_toolset = OpenAPIToolset(
    spec_str=openapi_spec_string,
    spec_str_type='json',
    # No authentication needed for httpbin.org
)

# --- Agent Definition ---
root_agent = LlmAgent(
    name=AGENT_NAME_OPENAPI,
    model=GEMINI_MODEL,
    tools=[petstore_toolset], # Pass the list of RestApiTool objects
    instruction="""You are a Pet Store assistant managing pets via an
    Use the available tools to fulfill user requests.
    When creating a pet, confirm the details echoed back by the API.
    When listing pets, mention any filters used (like limit or status)
    When showing a pet by ID, state the ID you requested.
    """,
    description="Manages a Pet Store using tools generated from an Ope
)

# --- Session and Runner Setup ---
async def setup_session_and_runner():
    session_service_openapi = InMemorySessionService()
    runner_openapi = Runner(
        agent=root_agent,
        app_name=APP_NAME_OPENAPI,
        session_service=session_service_openapi,
    )
    await session_service_openapi.create_session(
```

```
        app_name=APP_NAME_OPENAPI,
        user_id=USER_ID_OPENAPI,
        session_id=SESSION_ID_OPENAPI,
    )
    return runner_openapi

# --- Agent Interaction Function ---
async def call_openapi_agent_async(query, runner_openapi):
    print("\n--- Running OpenAPI Pet Store Agent ---")
    print(f"Query: {query}")

    content = types.Content(role='user', parts=[types.Part(text=query)
    final_response_text = "Agent did not provide a final text response
    try:
        async for event in runner_openapi.run_async(
            user_id=USER_ID_OPENAPI, session_id=SESSION_ID_OPENAPI, ne
            ):
            # Optional: Detailed event logging for debugging
            # print(f"  DEBUG Event: Author={event.author}, Type={'Fin
            if event.get_function_calls():
                call = event.get_function_calls()[0]
                print(f"  Agent Action: Called function '{call.name}'
            elif event.get_function_responses():
                response = event.get_function_responses()[0]
                print(f"  Agent Action: Received response for '{respon
                # print(f"  Tool Response Snippet: {str(response.respo
            elif event.is_final_response() and event.content and event
                # Capture the last final text response
                final_response_text = event.content.parts[0].text.stri

        print(f"Agent Final Response: {final_response_text}")

    except Exception as e:
        print(f"An error occurred during agent run: {e}")
        import traceback
        traceback.print_exc() # Print full traceback for errors
```

```
    print("-" * 30)

# --- Run Examples ---
async def run_openapi_example():
    runner_openapi = await setup_session_and_runner()

    # Trigger listPets
    await call_openapi_agent_async("Show me the pets available.", runn
    # Trigger createPet
    await call_openapi_agent_async("Please add a new dog named 'Dukey'
    # Trigger showPetById
    await call_openapi_agent_async("Get info for pet with ID 123.", ru

# --- Execute ---
if __name__ == "__main__":
    print("Executing OpenAPI example...")
    # Use asyncio.run() for top-level execution
    try:
        asyncio.run(run_openapi_example())
    except RuntimeError as e:
        if "cannot be called from a running event loop" in str(e):
            print("Info: Cannot run asyncio.run from a running event l
            # If in Jupyter/Colab, you might need to run like this:
            # await run_openapi_example()
        else:
            raise e
    print("OpenAPI example finished.")
```