



google / adk-docs

 Type























 Code

 Issues 113

 Pull requests 116

 Actions

 Projects

 Wiki



# Update llms.txt to align with the llms.txt standard and act as a sitemap for models #726

[Open](#)

derailed-dash opened on Sep 27, 2025 · edited by derailed-dash

Edits ▾ ⋮

As per [llmstxt.org](https://llmstxt.org) the `llms.txt` file should ideally provide an index to the key resources in folder, site or repo. It should provide a list of links, where each link has a summary of the content that can be found by following that link. This allows the model to determine which resources it should read in order to respond to a prompt.

From there, we can easily give our agents - like Gemini CLI - the ability to leverage the `llms.txt`. For example, we can use an extension like [adk-docs-ext](#) which uses an MCP server to read the `llms.txt` file and guides the model to identify and read appropriate pages.

Unfortunately, the `llms.txt` in this adk-docs repo is currently an overview of capabilities, but doesn't itself provide any links to the internal content.

When I use my extension with this `llms.txt` page and ask ADK-related questions, Gemini CLI responds with:

The `llms.txt` file contains a table of contents and summaries but **\*\*no actual link to the detailed documentation\*\***. For example, it lists a section on "Defining Effective Tool Functions," which is exactly what we need, but I have no way to navigate to it.

Because of this, I cannot give you a definitive, verified answer based on the official ADK documentation.

I spotted there is also an `llms-full.txt` file, but this file is huge with 85K lines and 3.2 million characters. It is too large to be useful in model context.

To address this problem I created an ADK agentic solution that can trawl any folder or repo, and produce an `llms.txt` file that contains both a repo summary, and a summary for all internal md and code resources. The application is [here](#).

I ran it for the adk-docs repo, and created the attached `llms.txt`, which may be more suitable. (I've trimmed off the last few entries, to make it fit in this feature request.)

I've created a couple of blogs on this topic that might be useful and relevant:

- [Give Your AI Agents Deep Understanding With LLMS.txt](#)
- [Give Your AI Agents Deep Understanding — Creating the LLMS.txt with a Multi-Agent ADK Solution](#)

The second one contains the full `llms.txt` generated for adk-docs.

# adk-docs Sitemap



The Agent Development Kit (ADK) is a comprehensive, open-source framework designed to streamline the development, evaluation, and deployment of AI agents in both Python and Java. It emphasizes a code-first approach, offering modularity, flexibility, and compatibility with various models and deployment environments like Google Cloud's Vertex AI Agent Engine, Cloud Run, and GKE. ADK provides core primitives such as Agents (LLM, Workflow, Custom), Tools (Function, Built-in, Third-Party, OpenAPI, Google Cloud, MCP), Callbacks, Session Management, Memory, Artifact Management, and a robust Runtime with an Event Loop for orchestrating complex workflows.

The documentation covers a wide array of functionalities, from basic agent creation and local testing using the Dev UI and CLI, to advanced topics like multi-agent system design, authentication for tools, and performance optimization through parallel execution. It also delves into crucial aspects of agent development such as observability with integrations like AgentOps, Cloud Trace, Phoenix, and Weave, and implementing strong safety and security guardrails using callbacks and plugins. Furthermore, ADK supports grounding capabilities with Google Search and Vertex AI Search for accurate, real-time information retrieval.

Overall, the ADK aims to empower developers to build sophisticated, context-aware, and reliable AI applications. The provided tutorials and quickstarts guide users through progressive learning, enabling them to master agent composition, state management, and deployment strategies for diverse use cases, ensuring agents can operate effectively and securely in production environments.

## ## Home

- [CONTRIBUTING.md](https://github.com/google/adk-docs/blob/main/CONTRIBUTING.md): Provides guidelines for contributing to the project, including signing a Contributor License Agreement, reviewing community guidelines, and setting up the development environment. It details the contribution workflow, from finding tasks to code reviews and merging pull requests.
- [README.md](https://github.com/google/adk-docs/blob/main/README.md): Introduces the Agent Development Kit (ADK) as an open-source, code-first framework for building, evaluating, and deploying AI agents. It highlights key features such as a rich tool ecosystem, modular multi-agent systems, tracing, monitoring, and flexible deployment options. Installation instructions for Python and Java are provided, along with links to documentation and contributing guidelines.

## ## Docs

- [community.md](https://github.com/google/adk-docs/blob/main/docs/community.md): Lists community resources for the Agent Development Kit, including translations of the ADK documentation in Chinese, Korean, and Japanese. It also features community-written tutorials, guides, blog posts, and videos showcasing ADK features and use cases. The document encourages contributions of new resources and directs users to the contributing guidelines.
- [contributing-guide.md](https://github.com/google/adk-docs/blob/main/docs/contributing-guide.md): Details how to contribute to the Agent Development Kit, covering contributions to core Python/Java frameworks and documentation. It outlines prerequisites like signing a Contributor License Agreement and reviewing community guidelines. The guide also explains how to report issues, suggest enhancements, improve documentation, and contribute code via pull requests.
- [index.md](https://github.com/google/adk-docs/blob/main/docs/index.md): Describes the Agent Development Kit (ADK) as a flexible, modular, and model-agnostic framework for developing and deploying AI agents, optimized for Google's ecosystem. It outlines key features including flexible orchestration, multi-agent architecture, a rich tool ecosystem, deployment readiness, built-in evaluation, and safety features. Provides installation commands for Python and Java, along with links to quickstarts, tutorials, API references, and contribution guides.

## ## Docs A2A

- [index.md](https://github.com/google/adk-docs/blob/main/docs/a2a/index.md): Introduces the Agent2Agent (A2A) Protocol within ADK, designed for building complex multi-agent systems where agents collaborate securely and efficiently. It provides links to guides on the fundamentals of A2A, quickstarts for exposing and consuming remote agents, and the official A2A Protocol website.
- [intro.md](https://github.com/google/adk-docs/blob/main/docs/a2a/intro.md): Introduces the Agent2Agent (A2A) Protocol for building collaborative multi-agent systems in ADK, contrasting it with local sub-agents. It outlines when to use A2A (separate services, different teams/languages, formal contracts) versus local sub-agents (internal organization, performance-critical operations, shared memory). The document visualizes the A2A workflow for exposing and consuming agents and provides

document visualizes the A2A workflow for exposing and consuming agents, and provides a concrete customer service example.

- [quickstart-consuming.md](https://github.com/google/adk-docs/blob/main/docs/a2a/quickstart-consuming.md):

Provides a quickstart guide on consuming a remote agent via the Agent-to-Agent (A2A) Protocol within ADK. It demonstrates how a local root agent can use a remote prime agent hosted on a separate A2A server. The guide covers setting up dependencies, starting the remote server, understanding agent cards, and running the main consuming agent.

- [quickstart-exposing.md](https://github.com/google/adk-docs/blob/main/docs/a2a/quickstart-exposing.md):

Provides a quickstart guide on exposing an ADK agent as a remote agent via the Agent-to-Agent (A2A) Protocol. It details two methods: using the `to\_a2a()` function for automatic agent card generation and `uvicorn` serving, or manually creating an agent card for `adk api\_server --a2a`. The guide includes steps for setting up dependencies, starting the remote server, verifying its operation, and running a consuming agent.

## ## Docs Agents

- [config.md](https://github.com/google/adk-docs/blob/main/docs/agents/config.md):

Introduces the experimental Agent Config feature in ADK, allowing users to build and run agents using YAML text files without writing code. It outlines the setup process, including installing ADK Python libraries and configuring Gemini API access via environment variables. The document provides examples for building agents with built-in tools, custom tools, and sub-agents, and discusses deployment options and current limitations.

- [custom-agents.md](https://github.com/google/adk-docs/blob/main/docs/agents/custom-agents.md):

Explains how to create custom agents in ADK by inheriting from BaseAgent and implementing `\_run\_async\_impl` (or `runAsyncImpl` in Java) for arbitrary orchestration logic. It highlights use cases for custom agents, such as conditional logic, complex state management, external integrations, and dynamic agent selection, which go beyond predefined workflow patterns. The document provides a detailed example of a StoryFlowAgent to illustrate multi-stage content generation with conditional regeneration.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/agents/index.md):

Provides an overview of agents in the Agent Development Kit (ADK), defining an agent as a self-contained execution unit for specific goals. It categorizes agents into LLM Agents (for dynamic reasoning), Workflow Agents (for structured execution flow), and Custom Agents (for unique logic). The document emphasizes that combining these agent types in multi-agent systems allows for sophisticated and collaborative AI applications.

- [llm-agents.md](https://github.com/google/adk-docs/blob/main/docs/agents/llm-agents.md):

Describes `LlmAgent` as a core component in ADK for AI agent reasoning, response generation, and tool interaction. It covers defining an agent's identity, guiding its behavior with `instruction` parameters, and equipping it with `tools`. The document also details advanced configurations such as `generate\_content\_config` for LLM response tuning, `input\_schema`/`output\_schema` for structured data, context management with `include\_contents` and the use of planners and code executors.

management with `include_contents`, and the use of planners and code executors.

- [models.md](https://github.com/google/adk-docs/blob/main/docs/agents/models.md):

Explains how to integrate various Large Language Models (LLMs) with the Agent Development Kit (ADK), supporting both Google Gemini and Anthropic models, with future plans for more. It details model integration via direct string/registry for Google Cloud models and wrapper classes for external models like those accessed through LiteLLM. Provides comprehensive authentication setups for Google AI Studio, Vertex AI, and outlines how to use LiteLLM for other cloud or local models.

- [multi-agents.md](https://github.com/google/adk-docs/blob/main/docs/agents/multi-agents.md):

Explains how to build multi-agent systems in ADK by composing multiple ``BaseAgent`` instances for enhanced modularity and specialization. It details ADK primitives for agent composition, including agent hierarchy (parent/sub-agents) and workflow agents (``SequentialAgent``, ``ParallelAgent``, ``LoopAgent``) for orchestration. The document also covers interaction and communication mechanisms like shared session state, LLM-driven delegation (agent transfer), and explicit invocation using ``AgentTool``.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/agents/workflow-agents/index.md):

Introduces workflow agents in ADK as specialized components for orchestrating the execution flow of sub-agents. It explains that these agents operate based on predefined, deterministic logic, unlike LLM Agents. Three core types are presented: Sequential Agents (execute in order), Loop Agents (repeatedly execute until a condition is met), and Parallel Agents (execute concurrently).

- [loop-agents.md](https://github.com/google/adk-docs/blob/main/docs/agents/workflow-agents/loop-agents.md):

Describes the ``LoopAgent`` in ADK, a workflow agent that iteratively executes its sub-agents. It highlights its use for repetitive or iterative refinement workflows, such as document improvement. The document explains that ``LoopAgent`` is deterministic and relies on termination mechanisms like ``max_iterations`` or escalation signals from sub-agents to prevent infinite loops.

- [parallel-agents.md](https://github.com/google/adk-docs/blob/main/docs/agents/workflow-agents/parallel-agents.md):

Describes the ``ParallelAgent`` in ADK, a workflow agent that executes its sub-agents concurrently to improve performance for independent tasks. It explains that sub-agents operate in independent branches without automatic shared history, requiring explicit communication or external state management if data sharing is needed. The document provides examples like parallel web research to illustrate its use for concurrent data retrieval or computations.

- [sequential-agents.md](https://github.com/google/adk-docs/blob/main/docs/agents/workflow-agents/sequential-agents.md):

Details the ``SequentialAgent`` in ADK, a workflow agent that executes its sub-agents in a fixed, strict order. It emphasizes its deterministic nature, making it suitable for workflows requiring predictable execution paths. The document illustrates its use with a code development pipeline example, where output from one sub-agent is passed to the next via shared session state.

## ## Docs Api-Reference

- [index.md](https://github.com/google/adk-docs/blob/main/docs/api-reference/index.md):

Provides an overview of the API references for the Agent Development Kit (ADK) for

Provides an overview of the API references for the Agent Development Kit (ADK) for both Python and Java. It directs users to detailed documentation for Python API, Java API (Javadoc), CLI, Agent Config YAML, and REST API.

- [dejavufonts.md](https://github.com/google/adk-docs/blob/main/docs/api-reference/java/legal/dejavufonts.md):

Provides licensing information for DejaVu fonts (v2.37), Bitstream Vera Fonts, Arev Fonts, and TeX Gyre DJV Math fonts. It details permissions for reproduction, distribution, and modification, with specific renaming requirements for modified versions. The document also includes copyright notices for the respective font creators.

- [jquery.md](https://github.com/google/adk-docs/blob/main/docs/api-reference/java/legal/jquery.md):

Contains the MIT License for jQuery version 3.7.1. It grants permission to use, copy, modify, merge, publish, distribute, sublicense, and sell copies of the software. The license also includes a disclaimer of warranty and liability.

- [jqueryUI.md](https://github.com/google/adk-docs/blob/main/docs/api-reference/java/legal/jqueryUI.md):

Contains the Apache 2.0 license for jQuery UI version 1.13.2. It specifies the terms for using, copying, modifying, and distributing the software, and notes that sample code is waived via CC0. The document also advises reviewing licenses for externally maintained libraries in `node\_modules` and `external` directories.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/api-reference/rest/index.md):

Provides a reference for the REST API exposed by the ADK web server, detailing endpoints for agent execution and session management. It describes the `/run` and `/run\_sse` endpoints for executing agent runs, including request and response body structures. The document also outlines the `Content` and `Event` objects used in API interactions.

## ## Docs Artifacts

- [index.md](https://github.com/google/adk-docs/blob/main/docs/artifacts/index.md):

Explains Artifacts in ADK as a mechanism for managing named, versioned binary data associated with sessions or users. It defines artifacts as `google.genai.types.Part` objects with `inline\_data` (raw bytes, MIME type) and discusses their persistence and management by `ArtifactService` implementations like `InMemoryArtifactService` and `GcsArtifactService`. The document highlights use cases for handling non-textual data, persisting large data, and managing user files.

## ## Docs Callbacks

- [design-patterns-and-best-practices.md](https://github.com/google/adk-docs/blob/main/docs/callbacks/design-patterns-and-best-practices.md):

Outlines common design patterns and best practices for leveraging ADK callbacks to enhance agent behavior. It covers patterns like guardrails, dynamic state management, logging, caching, request/response modification, conditional skipping of steps, tool-specific actions (authentication/summarization control), and artifact handling. Best practices emphasize focus, performance, error handling, state management, idempotency, testing, and clarity.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/callbacks/index.md):

Introduces callbacks as a fundamental ADK feature for observing, customizing, and

introduces callbacks as a fundamental ADK feature for observing, customizing, and controlling an agent's execution process. It explains how callbacks act as checkpoints at various stages, such as before/after agent runs, LLM calls, and tool executions. The document highlights their utility for debugging, implementing guardrails, managing state, and integrating external actions, emphasizing how their return value influences the agent's flow.

- [types-of-callbacks.md](https://github.com/google/adk-docs/blob/main/docs/callbacks/types-of-callbacks.md):

Describes different types of callbacks in ADK that trigger at various stages of an agent's execution, including agent lifecycle callbacks (`before\_agent\_callback`, `after\_agent\_callback`), LLM interaction callbacks (`before\_model\_callback`, `after\_model\_callback`), and tool execution callbacks (`before\_tool\_callback`, `after\_tool\_callback`). It explains the purpose and return value effects of each callback, emphasizing their role in inspecting, modifying, or skipping execution steps.

## ## Docs Context

- [index.md](https://github.com/google/adk-docs/blob/main/docs/context/index.md):

Explains "context" in ADK as the essential information available to agents and tools during operations, enabling state maintenance, data passing, and service access. It introduces `InvocationContext` as the comprehensive container, with specialized `ReadOnlyContext`, `CallbackContext`, and `ToolContext` objects for specific use cases. The document details common tasks like accessing information, managing state, working with artifacts, handling tool authentication, and leveraging memory.

## ## Docs Deploy

- [agent-engine.md](https://github.com/google/adk-docs/blob/main/docs/deploy/agent-engine.md):

Describes deploying ADK agents to Vertex AI Agent Engine, a fully managed Google Cloud service for scaling AI agents in production. It provides accelerated deployment instructions using the Agent Starter Pack CLI tool for quick setup and standard step-by-step instructions for manual management. The guide covers prerequisites, preparing the ADK project, connecting to Google Cloud, deploying, and testing the deployed agent via REST calls or Python SDK.

- [cloud-run.md](https://github.com/google/adk-docs/blob/main/docs/deploy/cloud-run.md):

Explains how to deploy ADK agents to Google Cloud Run, a managed auto-scaling platform for container-based applications. It provides instructions for both `adk deploy cloud\_run` CLI command and manual deployment using `gcloud run deploy` with a `Dockerfile`. The guide covers agent sample setup, environment variables, deployment payload, commands, and testing the deployed agent via UI or API.

- [gke.md](https://github.com/google/adk-docs/blob/main/docs/deploy/gke.md):

Explains how to deploy ADK agents to Google Kubernetes Engine (GKE), a managed Kubernetes service, for more control and running open models. It covers environment variable setup, enabling APIs and permissions, and two deployment options: manual deployment using `gcloud` and `kubectl` with Kubernetes manifests, or automated deployment using the `adk deploy gke` command. The guide also includes troubleshooting tips for common deployment issues and cleanup instructions.

Troubleshooting tips for common deployment issues and cleanup instructions.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/deploy/index.md):

Introduces the deployment options for ADK agents, emphasizing moving agents from local development to scalable and reliable production environments. It outlines three main deployment targets: Vertex AI Agent Engine (fully managed), Cloud Run (container-based, auto-scaling), and Google Kubernetes Engine (GKE) for more control. The document also mentions other container-friendly infrastructure options.

## ## Docs Evaluate

- [index.md](https://github.com/google/adk-docs/blob/main/docs/evaluate/index.md):

Explains the importance of evaluating AI agents in ADK due to LLM variability, going beyond traditional pass/fail tests to qualitative assessment of both final output and agent trajectory. It defines what to evaluate (trajectory and tool use, final response), discusses evaluation criteria (tool trajectory avg score, response match score), and outlines three methods for running evaluations: web-based UI (`adk web`), programmatically (`pytest`), and via CLI (`adk eval`).

## ## Docs Events

- [index.md](https://github.com/google/adk-docs/blob/main/docs/events/index.md):

Explains that Events are fundamental units of information flow in ADK, representing occurrences during an agent's interaction lifecycle. It details the `Event` object's structure, including `author`, `invocation\_id`, `id`, `timestamp`, and `actions` for side-effects and control flow. The document describes how to identify event origin and type, extract key information (text, function calls/responses), detect actions/side effects (state/artifact changes, control signals), and determine if an event is a "final" response.

## ## Docs Get-Started

- [about.md](https://github.com/google/adk-docs/blob/main/docs/get-started/about.md):

Provides an overview of the Agent Development Kit (ADK), an open-source, code-first toolkit for building, evaluating, and deploying AI agents. It outlines core concepts like Agents, Tools, Callbacks, Session Management, Memory, Artifact Management, Code Execution, Planning, Models, Events, and the Runner. The document highlights key capabilities such as multi-agent system design, a rich tool ecosystem, flexible orchestration, integrated developer tooling, native streaming support, built-in evaluation, broad LLM support, extensibility, and state/memory management.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/get-started/index.md):

Serves as a starting point for the Agent Development Kit (ADK), designed to help developers build, manage, evaluate, and deploy AI-powered agents. It provides links to installation guides, quickstarts for basic and streaming agents, a multi-agent tutorial, and sample agents. The page also offers an "About" section to learn core components of ADK.

- [installation.md](https://github.com/google/adk-docs/blob/main/docs/get-started/installation.md):

Provides instructions for installing the Agent Development Kit (ADK) for both

Provides instructions for installing the Agent Development Kit (ADK) for both Python and Java. For Python, it recommends creating and activating a virtual environment using `venv` before installing via `pip`. For Java, it outlines adding `google-adk` and `google-adk-dev` dependencies using Maven or Gradle.

- [quickstart.md](https://github.com/google/adk-docs/blob/main/docs/get-started/quickstart.md):

Provides a quickstart guide for the Agent Development Kit (ADK), detailing how to set up the environment, install ADK, create a basic agent project with multiple tools, and configure model authentication. It demonstrates running the agent locally using the terminal (`adk run`), the interactive web UI (`adk web`), or as an API server (`adk api\_server`). The guide includes examples for defining tools and integrating them into an agent.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/get-started/streaming/index.md):

Introduces streaming quickstarts for the Agent Development Kit (ADK), emphasizing real-time interactive experiences through live voice conversations, tool use, and continuous updates. It clarifies that this section focuses on bidi-streaming (live) rather than token-level streaming. The page provides links to Python and Java quickstarts for streaming capabilities, custom audio streaming app samples (SSE and WebSockets), a bidi-streaming development guide series, and a related blog post.

- [quickstart-streaming-java.md](https://github.com/google/adk-docs/blob/main/docs/get-started/streaming/quickstart-streaming-java.md):

Provides a quickstart guide for building a basic agent with ADK Streaming in Java, focusing on low-latency, bidirectional voice interactions. It covers setting up the Java/Maven environment, creating a `ScienceTeacherAgent`, testing text-based streaming with the Dev UI, and enabling live audio communication. The guide includes code examples for agent definition, `pom.xml` configuration, and a custom live audio application.

- [quickstart-streaming.md](https://github.com/google/adk-docs/blob/main/docs/get-started/streaming/quickstart-streaming.md):

Provides a quickstart guide for ADK Streaming in Python, demonstrating how to create a simple agent and enable low-latency, bidirectional voice and video communication. It covers environment setup, ADK installation, agent project structure with a Google Search tool, and platform configuration (Google AI Studio or Vertex AI). The guide shows how to test the agent using `adk web` for interactive UI, `adk run` for terminal interaction, and `adk api\_server` for API exposure, and notes supported models for live API.

- [testing.md](https://github.com/google/adk-docs/blob/main/docs/get-started/testing.md):

Explains how to test ADK agents in a local development environment using the ADK API server. It provides commands for launching the server, creating sessions, and sending queries via `curl` for both single-response and streaming endpoints. The document also discusses integrations with third-party observability tools and options for deploying agents.

## ## Docs Grounding

- [google\_search\_grounding.md](https://github.com/google/adk-docs/blob/main/docs/grounding/google\_search\_grounding.md):

Describes Google Search Grounding in ADK, a feature allowing AI agents to access real-time web information for more accurate responses. It details quick setup

real-time web information for more accurate responses. It details quick setup, grounding architecture (user query, LLM tool-calling, grounding service interaction, context injection), and response structure including metadata for source attribution. The guide provides best practices for displaying search suggestions and citations, emphasizing the tool's value for time-sensitive queries.

- [vertex\_ai\_search\_grounding.md](https://github.com/google/adk-docs/blob/main/docs/grounding/vertex\_ai\_search\_grounding.md):

Explains Vertex AI Search Grounding in ADK, a feature enabling AI agents to access private enterprise documents for grounded responses. It covers quick setup, grounding architecture (data flow, LLM analysis, document retrieval, context injection), and response structure including metadata for citations. The guide emphasizes best practices for displaying citations and outlines the process for integrating and using Vertex AI Search with ADK agents.

## ## Docs MCP

- [index.md](https://github.com/google/adk-docs/blob/main/docs/mcp/index.md):

Introduces the Model Context Protocol (MCP) as an open standard for LLMs to communicate with external applications, data sources, and tools. It explains MCP's client-server architecture and how ADK facilitates using and exposing MCP tools. The document highlights MCP Toolbox for Databases, an open-source server for securely exposing various data sources as Gen AI tools, and mentions MCP Servers for Google Cloud Genmedia.

## ## Docs Observability

- [agentops.md](https://github.com/google/adk-docs/blob/main/docs/observability/agentops.md):

Describes integrating AgentOps with Google ADK for enhanced agent observability, including session replays, metrics, and monitoring. It highlights AgentOps' benefits over ADK's native tracing, such as unified tracing, rich visualization, detailed debugging, and cost/latency tracking. The document provides straightforward installation and initialization steps, explaining how AgentOps instruments ADK by neutralizing native telemetry and controlling span creation.

- [arize-ax.md](https://github.com/google/adk-docs/blob/main/docs/observability/arize-ax.md):

Describes how to integrate Agent Development Kit (ADK) with Arize AX for LLM application observability. It explains how Arize AX automatically collects traces for agent interactions, tool calls, and model requests, enabling performance evaluation, debugging, and production monitoring. The document provides installation instructions, environment variable setup, and code examples for connecting an ADK application to Arize AX.

- [cloud-trace.md](https://github.com/google/adk-docs/blob/main/docs/observability/cloud-trace.md):

Explains how to integrate Agent Development Kit (ADK) with Google Cloud Trace for agent observability. It details how Cloud Trace, built on OpenTelemetry, enables comprehensive tracing of agent interactions, debugging latency issues, and visualizing execution flows. The document provides setup instructions for Agent Engine and Cloud Run deployments, as well as customized setups, and guides on inspecting traces in the Cloud Trace Explorer.

- [logging.md](https://github.com/google/adk-

`logging.md](https://github.com/google/adk-docs/blob/main/docs/observability/logging.md):`

Explains how the Agent Development Kit (ADK) uses Python's standard `logging` module for flexible and powerful logging. It covers configuring logging levels (DEBUG, INFO, WARNING, ERROR, CRITICAL) programmatically and via the `adk CLI` commands. The document details how to read and understand ADK logs, providing a practical example for debugging agent behavior by analyzing LLM request prompts.

– [phoenix.md](https://github.com/google/adk-docs/blob/main/docs/observability/phoenix.md):

Describes how to integrate Google ADK with Phoenix, an open-source, self-hosted observability platform for LLM applications and AI Agents. It explains that Phoenix automatically collects traces from ADK using OpenInference instrumentation, enabling tracing, performance evaluation, and debugging. The document provides installation and setup instructions for Phoenix Cloud, along with a code example for observing agent interactions.

– [weave.md](https://github.com/google/adk-docs/blob/main/docs/observability/weave.md):

Describes how to integrate Google ADK with Weave by Weights & Biases (WandB) for logging and visualizing model calls and OpenTelemetry traces. It outlines prerequisites, installation steps, and how to configure OpenTelemetry to send ADK traces to the Weave dashboard. The document includes a Python code example for a math agent using a calculator tool and provides notes on environment variables and project configuration.

## ## Docs Plugins

– [index.md](https://github.com/google/adk-docs/blob/main/docs/plugins/index.md):

Defines Plugins in ADK as custom code modules executed at various stages of an agent workflow lifecycle using callback hooks, offering functionality applicable across the entire workflow. It explains that Plugins build on Callbacks but have a global scope, applying to every agent, tool, and LLM call managed by a `Runner`. The document details how to define and register Plugin classes, discusses their precedence over agent-level callbacks, and describes various callback hooks for user messages, runner, agent, model, tool, and event operations.

## ## Docs Runtime

– [index.md](https://github.com/google/adk-docs/blob/main/docs/runtime/index.md):

Describes the ADK Runtime as the engine orchestrating agent applications, managing information flow, state changes, and external interactions. It explains the core concept of an Event Loop, where the `Runner` coordinates with "Execution Logic" components (Agents, Tools, Callbacks) by yielding and processing `Event` objects. The document details the roles of `Runner`, `Execution Logic`, `Event`, `Services`, `Session`, and `Invocation` within this cycle, emphasizing state commitment timing and streaming behavior.

– [runconfig.md](https://github.com/google/adk-docs/blob/main/docs/runtime/runconfig.md):

Explains `RunConfig`, which defines runtime behavior and options for ADK agents, controlling aspects like speech, streaming, function calling, artifact saving, and LLM call limits. It details the `RunConfig` class definition and its parameters, including `speech config`, `response modalities`, `save input blobs as artifacts`

including `speech_config`, `response_modalities`, `save_input_blocks_as_attachments`, `'streaming_mode'`, `'output_audio_transcription'`, and `'max_llm_calls'`. The document provides examples for basic, streaming, and speech-enabled configurations, along with validation rules.

## ## Docs Safety

- [index.md](https://github.com/google/adk-docs/blob/main/docs/safety/index.md):

Provides an overview of safety and security for AI agents in ADK, focusing on mitigating risks like misalignment and harmful content generation. It outlines a multi-layered approach including identity and authorization, guardrails (in-tool, Gemini safety features, callbacks, plugins, Gemini as a safety guardrail), sandboxed code execution, evaluation/tracing, and network controls. The document also discusses sources and categories of risk, and best practices for secure agent design.

## ## Docs Sessions

- [express-mode.md](https://github.com/google/adk-docs/blob/main/docs/sessions/express-mode.md):

Explains how to use Vertex AI Express Mode to access `'VertexAiSessionService'` and `'VertexAiMemoryBankService'` for free. It details the process of creating an Agent Engine instance and configuring environment variables (API keys) for both session and memory management. The document also outlines the quotas and limitations of the free Express Mode.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/sessions/index.md):

Introduces the core concepts of conversational context in ADK: `'Session'`, `'State'`, and `'Memory'`. It defines `'Session'` as a current conversation thread with `'Events'` and temporary `'State'`, and `'Memory'` as searchable, cross-session long-term knowledge. The document explains `'SessionService'` and `'MemoryService'` as managers for these concepts, highlighting in-memory implementations for development and persistent options for production.

- [memory.md](https://github.com/google/adk-docs/blob/main/docs/sessions/memory.md):

Explains ADK's `'MemoryService'` for managing long-term knowledge beyond individual sessions, contrasting it with session-specific state. It outlines the two primary responsibilities of `'MemoryService'`: ingesting information from sessions and searching the knowledge store. The document compares `'InMemoryMemoryService'` for prototyping with `'VertexAiMemoryBankService'` for persistent, sophisticated memory, detailing prerequisites and usage examples for each.

- [session.md](https://github.com/google/adk-docs/blob/main/docs/sessions/session.md):

Details the `'Session'` object in ADK, which tracks individual conversation threads with properties like `'id'`, `'appName'`, `'userId'`, `'events'` (history), `'state'` (temporary data), and `'lastUpdateTime'`. It explains that `'SessionService'` manages the lifecycle of these sessions, including creation, retrieval, updating (via appending events), and deletion. The document describes various `'SessionService'` implementations like `'InMemorySessionService'`, `'VertexAiSessionService'`, and `'DatabaseSessionService'`, each offering different persistence capabilities.

- [state.md](https://github.com/google/adk-docs/blob/main/docs/sessions/state.md):

Describes `'session.state'` as a key-value scratchpad within an ADK `'Session'` for dynamic data during a conversation. It outlines serializable key-value pair

dynamic data during a conversation. It outlines serializable key value pair structure, mutability, and persistence based on the `SessionService`. The document details organizing state with prefixes (`user:`, `app:`, `temp:`) and accessing state in agent instructions via templating. It also provides recommended methods for updating state, emphasizing `EventActions.state\_delta` and `ToolContext`/`CallbackContext`.

## ## Docs Streaming

- [configuration.md](https://github.com/google/adk-docs/blob/main/docs/streaming/configuration.md):

Explains how to configure streaming behavior for live agents in ADK using `RunConfig`. It specifically details how to set `speech\_config` for voice customization (voice name, language code) within `Runner.run\_live(...)`.

- [custom-streaming-ws.md](https://github.com/google/adk-docs/blob/main/docs/streaming/custom-streaming-ws.md):

Overviews a custom asynchronous web application built with ADK Streaming and FastAPI, enabling real-time, bidirectional audio and text communication using WebSockets. It details server-side components (ADK agent sessions, WebSocket connections, message relay) and client-side components (JavaScript, Web Audio API, UI management). The document also includes supported models for voice/video streaming, troubleshooting tips, and next steps for production.

- [custom-streaming.md](https://github.com/google/adk-docs/blob/main/docs/streaming/custom-streaming.md):

Overviews a custom asynchronous web application built with ADK Streaming and FastAPI, enabling real-time, bidirectional audio and text communication using Server-Sent Events (SSE). It details server-side components (FastAPI, SSE, session management, Google Search integration) and client-side components (JavaScript, Web Audio API, automatic reconnection). The document also covers session resumption configuration, message sending/processing, and next steps for production deployment.

- [part1.md](https://github.com/google/adk-docs/blob/main/docs/streaming/dev-guide/part1.md):

Introduces bidirectional streaming in ADK, highlighting its shift from traditional request-response to real-time, two-way communication with interruption support for natural AI conversations. It differentiates bidi-streaming from other streaming types (server-side, token-level) and provides real-world application examples in customer service, field service, healthcare, and finance. The document also outlines the high-level ADK bidi-streaming architecture and steps for setting up the development environment.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/streaming/index.md):

Introduces bidi-streaming (live streaming) in ADK as an experimental feature for low-latency bidirectional voice and video interaction with AI agents, leveraging Gemini Live API. It differentiates bidi-streaming from token-level streaming and highlights its benefits for natural human-like conversations and multimodal support. The page provides quickstarts, custom audio streaming app samples, a development guide series, and a related blog post.

- [streaming-tools.md](https://github.com/google/adk-docs/blob/main/docs/streaming/streaming-tools.md):

Explains how to define streaming tools in ADK for live agents, allowing functions to stream intermediate results back to agents. It specifies that streaming tools must be asynchronous Python functions returning an `AsyncGenerator`. The document

must be asynchronous Python functions returning an `AsyncGenerator`. The document provides examples for monitoring stock prices and video streams, and notes a reserved parameter ``input_stream: LiveRequestQueue`` for video streaming.

## ## Docs Tools

- [authentication.md](https://github.com/google/adk-docs/blob/main/docs/tools/authentication.md):

Explains ADK's system for handling tool authentication for protected resources, detailing ``AuthScheme`` and ``AuthCredential`` components. It describes supported credential types (API\_KEY, OAuth2, Service Account, OpenID Connect) and how to configure authentication on various toolsets (OpenAPI, Google API, APIHub, ApplicationIntegration). The document provides a step-by-step guide for handling interactive OAuth/OIDC flows client-side and building custom tools requiring authentication.

- [built-in-tools.md](https://github.com/google/adk-docs/blob/main/docs/tools/built-in-tools.md):

Describes built-in tools in ADK, providing ready-to-use functionalities like Google Search, Code Execution, and Vertex AI RAG/Search. It explains how to import, configure, and register these tools with agents, noting current limitations on combining built-in tools with other tool types or using them within sub-agents. Examples are provided for Python and Java implementations.

- [confirmation.md](https://github.com/google/adk-docs/blob/main/docs/tools/confirmation.md):

Describes the Tool Confirmation feature in ADK, which allows tools to pause execution and request confirmation or structured data from a user or supervising system. It covers boolean confirmation for simple yes/no responses and advanced confirmation for complex data. The document explains how to define confirmation requests with hints and payloads, and how to handle remote confirmations via the ADK API server.

- [function-tools.md](https://github.com/google/adk-docs/blob/main/docs/tools/function-tools.md):

Explains how to create custom function tools in ADK to integrate tailored logic into agents, including standard Python functions, long-running functions, and agents-as-tools. It details how the ADK framework inspects function signatures and docstrings to generate tool schemas for LLMs. The document provides guidelines for defining parameters, return types, and docstrings, and discusses passing data between tools using session state.

- [google-cloud-tools.md](https://github.com/google/adk-docs/blob/main/docs/tools/google-cloud-tools.md):

Describes how Google Cloud tools in ADK simplify connecting agents to Google Cloud products and services, including custom APIs via Apigee API Hub, prebuilt connectors for enterprise systems, and databases via MCP Toolbox. It provides prerequisites and steps for creating ``ApiHubToolset`` and ``ApplicationIntegrationToolset`` for various integrations, emphasizing authentication configurations. The document also covers loading and utilizing Toolbox tools for databases.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/tools/index.md):

Introduces "Tools" in ADK as capabilities that extend an AI agent's functionality beyond core text generation and reasoning, enabling interaction with external systems. It categorizes tools into Function Tools (custom Python/Java functions, agents-as-tools, long-running functions), Built-in Tools (Google Search, Code

agents as tools, long running functions, built-in tools (Google Search, Code Execution), and Third-Party Tools (LangChain, CrewAI). The document explains how agents use tools through function calling and emphasizes effective tool definition and the role of `ToolContext`.

- [mcp-tools.md](https://github.com/google/adk-docs/blob/main/docs/tools/mcp-tools.md):

This guide covers two primary integration patterns for Model Context Protocol (MCP) with ADK: using ADK as an MCP client to leverage tools from external MCP servers, and building an MCP server to expose ADK tools. It details prerequisites like Node.js and `npx`, and explains the `MCPToolset` class for connection management, tool discovery, and proxying calls. The document provides examples for file system and Google Maps MCP servers, and discusses considerations for deploying agents with MCP tools.

- [openapi-tools.md](https://github.com/google/adk-docs/blob/main/docs/tools/openapi-tools.md):

Explains how ADK integrates with external REST APIs by automatically generating callable tools from OpenAPI Specifications (v3.x) using `OpenAPIToolset`. It details how `OpenAPIToolset` parses specs, discovers operations, and creates `RestApiTool` instances with dynamic schema generation and execution. The document outlines the usage workflow, including obtaining specs, instantiating the toolset with authentication, adding tools to agents, and instructing agents.

- [performance.md](https://github.com/google/adk-docs/blob/main/docs/tools/performance.md):

Explains how ADK (version 1.10.0+) supports parallel execution of agent-requested function tools to improve performance and responsiveness. It details how to build parallel-ready tools using asynchronous Python functions (`async def`, `await`) for HTTP/database calls, yielding behavior for long loops, and thread pools for intensive operations. The guide also advises on writing parallel-ready prompts and tool descriptions.

- [third-party-tools.md](https://github.com/google/adk-docs/blob/main/docs/tools/third-party-tools.md):

Describes how ADK integrates with third-party tools from other AI Agent frameworks like LangChain and CrewAI for enhanced extensibility and reusability. It provides examples for using `LangchainTool` to wrap LangChain's Tavily search tool and `CrewaiTool` to wrap CrewAI's Serper API for web search, detailing installation, instantiation, and agent integration steps.

## ## Docs Tutorials

- [agent-team.md](https://github.com/google/adk-docs/blob/main/docs/tutorials/agent-team.md):

Presents a tutorial on building an intelligent multi-agent Weather Bot team using ADK, progressively adding features like multi-LLM support, agent delegation, session state for memory, and safety guardrails with callbacks. It emphasizes practical application of core ADK concepts for complex, real-world agentic systems. The tutorial is structured for interactive notebook environments and provides an alternative for ADK's built-in tools.

- [index.md](https://github.com/google/adk-docs/blob/main/docs/tutorials/index.md):

Provides an introduction to ADK tutorials, emphasizing a progressive, step-by-step learning approach for various ADK features and capabilities. It highlights the "Agent Team" tutorial, which covers tools, multi-LLM usage, agent delegation

Agent team tutorial, which covers tools, multi LLM usage, agent delegation, session state, and callbacks.

## ## Examples Java

- [README.md](https://github.com/google/adk-docs/blob/main/examples/java/demos/README.md):

No meaningful summary available.

- [README.md](https://github.com/google/adk-docs/blob/main/examples/java/demos/patent-search-agent/README.md):

Provides instructions for setting up and deploying a Patent Search Agent using ADK Java SDK, covering API key configuration, AlloyDB setup, and Cloud Run Function creation. It guides users through testing agent interaction with sample inputs and deploying the agent to Cloud Run for web UI access.

- [README.md](https://github.com/google/adk-

docs/blob/main/examples/java/demos/patent-search-agent/src/main/java/tools/CloudRunFunction/README.md):

Provides detailed instructions for setting up AlloyDB database objects and creating a Cloud Run Function for a patent search application. It covers creating AlloyDB clusters/instances, ingesting patent data using SQL scripts, granting necessary IAM permissions, and deploying the Cloud Run Function with VPC connector configuration.

## ## Examples Python

- [agent.py](https://github.com/google/adk-docs/blob/main/examples/python/agent-samples/youtube-shorts-assistant/agent.py):

Defines a YouTube Shorts assistant agent (`youtube\_shorts\_agent`) implemented as an `LlmAgent` that orchestrates scriptwriting, visualizing, and formatting tasks. It uses `AgentTool` to delegate these tasks to specialized `LlmAgent` sub-agents (ShortsScriptwriter, ShortsVisualizer, ConceptFormatter). The agent loads instructions from files and saves intermediate results to session state.

- [loop\_agent.py](https://github.com/google/adk-docs/blob/main/examples/python/agent-samples/youtube-shorts-assistant/loop\_agent.py):

Defines a YouTube Shorts assistant agent (`youtube\_shorts\_agent`) implemented as a `LoopAgent` that iteratively calls sub-agents for scriptwriting, visualizing, and formatting. It sets a maximum of 3 iterations for the loop. The agent uses `LlmAgent`s for each sub-task, loads instructions from files, and saves intermediate results to session state.

- [loop\_agent\_runner.py](https://github.com/google/adk-docs/blob/main/examples/python/agent-samples/youtube-shorts-assistant/loop\_agent\_runner.py):

Demonstrates a YouTube Shorts assistant agent (`youtube\_shorts\_agent`) implemented as a `LoopAgent` that iteratively calls sub-agents for scriptwriting, visualizing, and formatting. It uses `LlmAgent`s for each sub-task, loads instructions from files, and saves intermediate results to session state. The script includes setup for running the agent programmatically with `InMemorySessionService`.

- [util.py](https://github.com/google/adk-docs/blob/main/examples/python/agent-samples/youtube-shorts-assistant/util.py):

Provides a utility function `load\_instruction\_from\_file` to read instruction text

Provides a utility function `load_instruction_from_file` to load instruction text from a specified file. It constructs the file path relative to the script's location and includes error handling for file not found or other exceptions. This function helps manage agent instructions externally.

- [storyflow\_agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/agents/custom-agent/storyflow\_agent.py):

Implements a custom `StoryFlowAgent` in ADK that orchestrates a multi-stage story generation and refinement workflow. It defines several `LlmAgent` sub-agents (`StoryGenerator`, `Critic`, `Reviser`, `GrammarCheck`, `ToneCheck`) and uses `LoopAgent` and `SequentialAgent` for control flow, including conditional regeneration based on tone analysis. The example demonstrates how to pass data between agents using session state and manage complex workflows.

- [capital\_agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/agents/llm-agent/capital\_agent.py):

Demonstrates an `LlmAgent` in ADK, focusing on its ability to use tools versus output schemas for retrieving capital city information. It defines `CountryInput` and `CapitalInfoOutput` schemas, a `get_capital_city` tool, and configures two agents: one using the tool and another enforcing an output schema. The example sets up session management and runners to illustrate agent interaction and state storage.

- [loop\_agent\_doc\_improv\_agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/agents/workflow-agents/loop\_agent\_doc\_improv\_agent.py):

Demonstrates a `LoopAgent` in ADK for iteratively improving a document through a refinement loop. It defines `LlmAgent` sub-agents for initial writing, critiquing, and refining, along with an `exit_loop` tool to terminate the process based on critique. The example illustrates how the `LoopAgent` orchestrates these steps, passing document state and criticism to refine the output until a completion phrase is met or max iterations are reached.

- [parallel\_agent\_web\_research.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/agents/workflow-agents/parallel\_agent\_web\_research.py):

Demonstrates a `ParallelAgent` in ADK for concurrent web research, orchestrating three `LlmAgent` researchers (`RenewableEnergyResearcher`, `EVResearcher`, `CarbonCaptureResearcher`) using Google Search. It shows how a `SequentialAgent` can then combine the `ParallelAgent` with a `SynthesisAgent` to merge the concurrently obtained research findings from session state into a structured report.

- [sequential\_agent\_code\_development\_agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/agents/workflow-agents/sequential\_agent\_code\_development\_agent.py):

Demonstrates a `SequentialAgent` in ADK for orchestrating a code development pipeline. It defines three `LlmAgent` sub-agents: `CodeWriterAgent`, `CodeReviewerAgent`, and `CodeRefactorerAgent`, which execute in a fixed order. The example highlights how these agents interact by passing data through session state (`output_key`) to ensure a structured and predictable workflow.

- [after\_agent\_callback.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/callbacks/after\_agent\_callback.py):

Logs entry and checks `'add_concluding_note'` in session state. If True, returns new Content to `*replace*` the agent's original output. If False or not present, returns None, allowing the agent's original output to be used.

- [after\_model\_callback.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/callbacks/after\_model\_callback.py):

`docs/blob/main/examples/python/snippets/callbacks/after_model_callback.py`).

Demonstrates the `after\_model\_callback` in ADK for inspecting and modifying the LLM's response after it's received. The `simple\_after\_model\_modifier` callback inspects the original text, and if it contains "joke", replaces it with "funny story" before returning a new `LlmResponse` object. The example sets up an `LlmAgent` with this callback and shows its effect on the agent's final response.

- [after\_tool\_callback.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/callbacks/after\_tool\_callback.py):

Demonstrates the `after\_tool\_callback` in ADK for inspecting and modifying a tool's result after its execution. The `simple\_after\_tool\_modifier` callback intercepts the result from `get\_capital\_city`, and if the capital is "Washington, D.C.", it modifies the response to include a note. The example sets up an `LlmAgent` with this callback to illustrate its effect on the final tool output.

- [before\_agent\_callback.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/callbacks/before\_agent\_callback.py):

Demonstrates the `before\_agent\_callback` in ADK, which executes just before an agent's main logic. It shows how the callback can inspect session state and, based on a condition (`skip\_llm\_agent`), either allow the agent to run normally or return a `types.Content` object to skip its execution entirely. The example uses an `LlmAgent` with this callback and illustrates two scenarios: one where the agent runs, and one where it's skipped.

- [before\_model\_callback.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/callbacks/before\_model\_callback.py):

Inspects/modifies the LLM request or skips the call. It shows how to add a prefix to the system instruction or block the LLM call if a specific keyword is found in the user message. The callback returns an `LlmResponse` to skip the LLM call, or `None` to proceed.

- [before\_tool\_callback.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/callbacks/before\_tool\_callback.py):

Demonstrates the `before\_tool\_callback` in ADK for inspecting and modifying tool arguments or skipping tool execution before it runs. It defines a `get\_capital\_city` tool and a `simple\_before\_tool\_modifier` callback that intercepts calls to `get\_capital\_city` to modify arguments or block execution based on the input country. The example sets up an `LlmAgent` with this callback and illustrates its behavior with different queries.

- [callback\_basic.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/callbacks/callback\_basic.py):

Provides a basic example of defining and registering a `before\_model\_callback` function with an `LlmAgent` in ADK. The callback `my\_before\_model\_logic` prints a message before the model call and then returns `None` to allow the call to proceed. The example sets up a simple agent interaction to demonstrate the callback's execution.

- [agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/get-started/google\_search\_agent/agent.py):

Defines a `root\_agent` using `google.adk.agents.Agent` that answers questions using Google Search. The agent is configured with a Gemini model, instructions to cite sources, and includes the `google\_search` tool. It serves as a basic search assistant.

- [agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/get-started/multi\_tool\_agent/agent.py):

Defines a `root\_agent` using `google.adk.agents.Agent` that answers questions

Defines a `root_agent` using `google.adk.agents.Agent` that answers questions about time and weather in a city. It includes two Python functions, `get_weather` and `get_current_time`, as tools for the agent. The agent is configured with a Gemini model and instructions to use these tools.

- [agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/streaming/adk-streaming-ws/app/google\_search\_agent/agent.py):

Defines a `root_agent` using `google.adk.agents.Agent` that answers questions using Google Search. The agent is configured with a Gemini Live API model and instructions to use the `google_search` tool. It is designed for streaming text and audio/video input.

- [main.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/streaming/adk-streaming-ws/app/main.py):

This FastAPI application enables real-time, bidirectional streaming interaction with an ADK agent via WebSockets. It initializes ADK agent sessions, handles client WebSocket connections, relays client messages (text/audio) to the agent, and streams text/audio responses back to clients. The application supports session resumption and includes logging for communication flow.

- [agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/streaming/adk-streaming/app/google\_search\_agent/agent.py):

Defines a `root_agent` using `google.adk.agents.Agent` that answers questions using Google Search. The agent is configured with a Gemini Live API model and instructions to use the `google_search` tool. It is designed for streaming text and audio/video input.

- [main.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/streaming/adk-streaming/app/main.py):

This FastAPI application enables real-time, bidirectional streaming interaction with an ADK agent via Server-Sent Events (SSE). It initializes ADK agent sessions, handles client SSE connections, processes client messages (text/audio), and streams agent responses (text/audio) back to clients. The application supports session resumption and includes logging for communication flow.

- [agent\_cli.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/auth/agent\_cli.py):

Demonstrates an ADK agent authentication flow using a command-line interface, simulating an OAuth 2.0 process. It shows how the agent detects a need for authentication, prompts the user for interaction, and then resumes execution after receiving the authentication callback. The example uses in-memory services for session and artifact storage and highlights the role of helper functions in managing the authentication flow.

- [helpers.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/auth/helpers.py):

Provides helper functions for managing authentication flows in ADK agent applications, particularly for OAuth 2.0/OIDC. It includes functions to asynchronously get user input, check if an ADK Event is an `adk_request_credential` function call, and extract the function call ID and `AuthConfig` from such events. These helpers are crucial for client-side handling of interactive authentication.

- [tools\_and\_agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/auth/tools\_and\_agent.py):

Configures an `LlmAgent` with tools that require OpenID Connect (OIDC) authentication, often layered on OAuth 2.0. It defines the `AuthScheme` and

authentication, often layered on OAuth 2.0. It defines the `AuthScheme` and `AuthCredential` for the OIDC flow, including authorization and token endpoints and required scopes. The agent is then equipped with tools from an `OpenAPIToolset` that are associated with this authentication setup.

- [bigquery.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/built-in-tools/bigquery.py):

Demonstrates the `BigQueryToolset` in ADK, providing a set of tools for integrating with Google BigQuery. It shows how to configure credentials and tool permissions (e.g., blocking write operations) using `BigQueryCredentialsConfig` and `BigQueryToolConfig`. The example defines an `Agent` that uses these tools to answer questions about BigQuery data and execute SQL queries.

- [code\_execution.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/built-in-tools/code\_execution.py):

Demonstrates the `built\_in\_code\_execution` tool in ADK, enabling an `LlmAgent` to execute Python code for calculations. It configures a `BuiltInCodeExecutor` and defines an agent instructed to use code execution for mathematical expressions. The example shows how the agent processes queries, generates code, executes it, and returns the numerical result.

- [google\_search.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/built-in-tools/google\_search.py):

Defines a `root\_agent` using `google.adk.agents.Agent` that answers questions using Google Search. The agent is configured with a Gemini model, instructions to cite sources, and includes the `google\_search` tool. It serves as a basic search assistant.

- [vertexai\_rag\_engine.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/built-in-tools/vertexai\_rag\_engine.py):

Demonstrates the `VertexAiRagRetrieval` tool in ADK for performing private data retrieval using Vertex AI RAG Engine. It shows how to configure the tool with a RAG corpus, similarity top-k, and vector distance threshold. The example defines an `Agent` that uses this tool to retrieve documentation and reference materials for questions.

- [vertexai\_search.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/built-in-tools/vertexai\_search.py):

Demonstrates the `VertexAiSearchTool` in ADK for enabling agents to search private data stores using Vertex AI Search. It shows how to configure the tool with a `data\_store\_id` and defines an `LlmAgent` that uses this tool to answer questions based on information found in the document store. The example includes a check for proper `data\_store\_id` configuration and illustrates how grounding metadata is provided in the response.

- [func\_tool.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/function-tools/func\_tool.py):

Demonstrates a `stock\_price\_agent` in ADK that retrieves current stock prices using a `get\_stock\_price` function wrapped as a `FunctionTool`. The agent is configured with a Gemini model and instructions to use the tool for ticker symbols or company names. It sets up session management and a runner to interact with the agent and print its responses.

- [human\_in\_the\_loop.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/function-tools/human\_in\_the\_loop.py):

Demonstrates a human-in-the-loop workflow in ADK using a `LongRunningFunctionTool` for reimbursement approvals. It defines `ask\_for\_approval` as a long-running

for reimbursement approvals. It defines `ask_for_approval` as a long running function that simulates ticket creation, and `reimburse` for processing payments. The `reimbursement_agent` uses these tools, initiating approval for amounts over \$100 and allowing for intermediate/final result updates from a client.

- [summarizer.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/function-tools/summarizer.py):

Defines a `root_agent` in ADK that uses an `AgentTool` to delegate text summarization to a specialized `summary_agent`. The `root_agent` is instructed to forward user messages to the `summarize` tool without modification and present the tool's response. The example sets up session management and a runner to demonstrate this delegation for long text inputs.

- [openapi\_tool.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/openapi\_tool.py):

Demonstrates integrating OpenAPI specifications into ADK agents by creating an `OpenAPIToolset` from a sample Pet Store API. It shows how the toolset automatically generates `RestApiTool` instances for API operations like `listPets`, `createPet`, and `showPetById`. The example configures an `LlmAgent` to use these tools and interacts with them via a runner.

- [customer\_support\_agent.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/overview/customer\_support\_agent.py):

Demonstrates an ADK multi-agent system with a `main_agent` and a `support_agent`, illustrating LLM-driven delegation. The `check_and_transfer` tool, when called by the `main_agent`, checks for "urgent" queries and uses `tool_context.actions.transfer_to_agent` to hand off control to the `support_agent`. This example showcases dynamic conversation routing based on user intent.

- [doc\_analysis.py](https://github.com/google/adk-docs/blob/main/examples/python/snippets/tools/overview/doc\_analysis.py):

Demonstrates a `FunctionTool` in ADK for analyzing documents using context from memory. It shows how the tool loads an artifact (document), searches memory for related context, performs a placeholder analysis, and saves the analysis result as a new artifact. The example highlights the use of `ToolContext` for interacting with artifact and memory services.



- koverholt mentioned this [on Dec 3, 2025](#)

[Add autogeneration of llms.txt during mkdocs build; add docs page on using llms.txt #1023](#)
- derailed-dash mentioned this [on Dec 6, 2025](#)

[Update URL to use generated llms.txt in ADK docs derailed-dash/adk-docs-ext#2](#)

koverholt on Dec 8, 2025

Collaborator

⋮

Thanks for opening this issue (and for your Gemini CLI extension for ADK docs & related blog posts!).

As of [#1023](#), we moved from a manual process to generate `llms.txt` to an automated process. You can see the details in that PR, but the short version is that the `llms.txt` and `llms-full.txt` are now auto-generated with each docs build as part of the mkdocs build process using the `mkdocs-llmstxt` plugin, and the built files live at <https://google.github.io/adk-docs/llms.txt> and <https://google.github.io/adk-docs/llms-full.txt> instead of the Git repo.

This means that [#1023](#) resolves the biggest issues of 1) we now have links in `llms.txt` for coding agents to follow and explore the ADK docs, 2) no need to manually update `llms.txt`. But I will leave this issue open so that we can keep improving `llms.txt` and consider how to include summaries of docs pages among other improvements in the future, ideally without having to run against an agentic solution with each docs build.

derailed-dash on Dec 8, 2025

Author

⋮

Awesome! Thanks for the update.

Add a comment

Write

Preview

H

B

I

≡

<>

@

↶

Use Markdown to format your comment

 Paste, drop, or click to add files

 Close issue



Comment



Remember, contributions to this repository should follow its [contributing guidelines](#), [security policy](#) and [code of conduct](#).

Metadata

Assignees

No one assigned

Labels

No labels

Type

No type

Projects

No projects


Milestone


No milestone

Relationships

None yet

Development

 Code with agent mode



No branches or pull requests

Notifications

Customize

 Subscribe

You're not receiving notifications from this thread.

Participants



 Give feedback