Introduction to Conversational Context: Session, State, and Memory - Agent Development Kit

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

Introduction to Conversational Context: Session, State, and Memory¶

Why Context Matters¶

Meaningful, multi-turn conversations require agents to understand context. Just like humans, they need to recall the conversation history: what's been said and done to maintain continuity and avoid repetition. The Agent Development Kit (ADK) provides structured ways to manage this context through `Session`, `State`, and `Memory`.

Core Concepts¶

Think of different instances of your conversations with the agent as distinct **conversation threads**, potentially drawing upon **long-term knowledge**.

1. **`Session`**: The Current Conversation Thread
2. Represents a *single, ongoing interaction* between a user and your agent system.
3. Contains the chronological sequence of messages and actions taken by the agent (referred to `Events`) during *that specific interaction*.
4. A `Session` can also hold temporary data (`State`) relevant only *during this conversation*.
5. **`State` (`session.state`)**: Data Within the Current Conversation
6. Data stored within a specific `Session`.

7. Used to manage information relevant *only* to the *current, active* conversation thread (e.g., items in a shopping cart *during this chat*, user preferences mentioned *in this session*).

8. **`Memory`**: Searchable, Cross-Session Information

9. Represents a store of information that might span *multiple past sessions* or include external data sources.

10. It acts as a knowledge base the agent can *search* to recall information or context beyond the immediate conversation.

Managing Context: Services¶

ADK provides services to manage these concepts:

1. **`SessionService`**: Manages the different conversation threads (`Session` objects)

2. Handles the lifecycle: creating, retrieving, updating (appending `Events`, modifying `State`), and deleting individual `Session`s.

3. **`MemoryService`**: Manages the Long-Term Knowledge Store (`Memory`)

4. Handles ingesting information (often from completed `Session`s) into the long-term store.

5. Provides methods to search this stored knowledge based on queries.

Implementations: ADK offers different implementations for both `SessionService` and `MemoryService`, allowing you to choose the storage backend that best fits your application's needs. Notably, **in-memory implementations** are provided for both services; these are designed specifically for **local testing and fast development**. It's important to remember that **all data stored using these in-memory options (sessions, state, or long-term knowledge) is lost when your application restarts**. For persistence and scalability beyond local testing, ADK also offers cloud-based and database service options.

In Summary:

- **`Session` & `State`**: Focus on the **current interaction** – the history and data of the *single, active conversation*. Managed primarily by a `SessionService`.
- **Memory**: Focuses on the **past and external information** – a *searchable archive* potentially spanning across conversations. Managed by a `MemoryService`.

What's Next?¶

In the following sections, we'll dive deeper into each of these components:

- **`Session`**: Understanding its structure and `Events`.
- **`State`**: How to effectively read, write, and manage session-specific data.
- **`SessionService`**: Choosing the right storage backend for your sessions.
- **`MemoryService`**: Exploring options for storing and retrieving broader context.

Understanding these concepts is fundamental to building agents that can engage in complex, stateful, and context-aware conversations.