Artifacts - Agent Development Kit

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

Artifacts¶

In ADK, **Artifacts** represent a crucial mechanism for managing named, versioned binary data associated either with a specific user interaction session or persistently with a user across multiple sessions. They allow your agents and tools to handle data beyond simple text strings, enabling richer interactions involving files, images, audio, and other binary formats.

Note

The specific parameters or method names for the primitives may vary slightly by SDK language (e.g., `save_artifact` in Python, `saveArtifact` in Java). Refer to the language-specific API documentation for details.

What are Artifacts?¶

- **Definition:** An Artifact is essentially a piece of binary data (like the content of a file) identified by a unique `filename` string within a specific scope (session or user). Each time you save an artifact with the same filename, a new version is created.
- **Representation:** Artifacts are consistently represented using the standard `google.genai.types.Part` object. The core data is typically stored within an inline data structure of the `Part` (accessed via `inline_data`), which itself contains:
- `data`: The raw binary content as bytes.
- `mime_type`: A string indicating the type of the data (e.g., `"image/png"`, `"application/pdf"`). This is essential for correctly interpreting the data later.

PythonJava

```python
# Example of how an artifact might be represented as a types.Part
import google.genai.types as types

# Assume 'image_bytes' contains the binary data of a PNG image
image_bytes = b'\x89PNG\r\n\x1a\n...' # Placeholder for actual image b

image_artifact = types.Part(
    inline_data=types.Blob(
        mime_type="image/png",
        data=image_bytes
    )
)

# You can also use the convenience constructor:
# image_artifact_alt = types.Part.from_bytes(data=image_bytes, mime_ty

print(f"Artifact MIME Type: {image_artifact.inline_data.mime_type}")
print(f"Artifact Data (first 10 bytes): {image_artifact.inline_data.da
```

```java
import com.google.genai.types.Part;
import java.nio.charset.StandardCharsets;

public class ArtifactExample {
    public static void main(String[] args) {
        // Assume 'imageBytes' contains the binary data of a PNG image
        byte[] imageBytes = {(byte) 0x89, (byte) 0x50, (byte) 0x4E, (b

        // Create an image artifact using Part.fromBytes
        Part imageArtifact = Part.fromBytes(imageBytes, "image/png");

        System.out.println("Artifact MIME Type: " + imageArtifact.inli
        System.out.println(
            "Artifact Data (first 10 bytes): "
                + new String(imageArtifact.inlineData().get().data().g
```

```
                    + "...");
        }
}
```

- **Persistence & Management:** Artifacts are not stored directly within the agent or session state. Their storage and retrieval are managed by a dedicated **Artifact Service** (an implementation of `BaseArtifactService`, defined in `google.adk.artifacts`. ADK provides various implementations, such as:
- An in-memory service for testing or temporary storage (e.g., `InMemoryArtifactService` in Python, defined in `google.adk.artifacts.in_memory_artifact_service.py`).
- A service for persistent storage using Google Cloud Storage (GCS) (e.g., `GcsArtifactService` in Python, defined in `google.adk.artifacts.gcs_artifact_service.py`). The chosen service implementation handles versioning automatically when you save data.

Why Use Artifacts?¶

While session `state` is suitable for storing small pieces of configuration or conversational context (like strings, numbers, booleans, or small dictionaries/ lists), Artifacts are designed for scenarios involving binary or large data:

1. **Handling Non-Textual Data:** Easily store and retrieve images, audio clips, video snippets, PDFs, spreadsheets, or any other file format relevant to your agent's function.
2. **Persisting Large Data:** Session state is generally not optimized for storing large amounts of data. Artifacts provide a dedicated mechanism for persisting larger blobs without cluttering the session state.
3. **User File Management:** Provide capabilities for users to upload files (which can be saved as artifacts) and retrieve or download files generated by the agent (loaded from artifacts).
4. **Sharing Outputs:** Enable tools or agents to generate binary outputs (like a PDF report or a generated image) that can be saved via `save_artifact` and later accessed by other parts of the application or even in subsequent sessions (if using user namespacing).

5. **Caching Binary Data:** Store the results of computationally expensive operations that produce binary data (e.g., rendering a complex chart image) as artifacts to avoid regenerating them on subsequent requests.

In essence, whenever your agent needs to work with file-like binary data that needs to be persisted, versioned, or shared, Artifacts managed by an `ArtifactService` are the appropriate mechanism within ADK.

Common Use Cases¶

Artifacts provide a flexible way to handle binary data within your ADK applications.

Here are some typical scenarios where they prove valuable:

- **Generated Reports/Files:**
- A tool or agent generates a report (e.g., a PDF analysis, a CSV data export, an image chart).
- **Handling User Uploads:**
- A user uploads a file (e.g., an image for analysis, a document for summarization) through a front-end interface.
- **Storing Intermediate Binary Results:**
- An agent performs a complex multi-step process where one step generates intermediate binary data (e.g., audio synthesis, simulation results).
- **Persistent User Data:**
- Storing user-specific configuration or data that isn't a simple key-value state.
- **Caching Generated Binary Content:**
- An agent frequently generates the same binary output based on certain inputs (e.g., a company logo image, a standard audio greeting).

Core Concepts¶

Understanding artifacts involves grasping a few key components: the service that manages them, the data structure used to hold them, and how they are identified and versioned.

Artifact Service (`BaseArtifactService`)¶

- **Role:** The central component responsible for the actual storage and retrieval logic for artifacts. It defines *how* and *where* artifacts are persisted.
- **Interface:** Defined by the abstract base class `BaseArtifactService`. Any concrete implementation must provide methods for:
- `Save Artifact`: Stores the artifact data and returns its assigned version number.
- `Load Artifact`: Retrieves a specific version (or the latest) of an artifact.
- `List Artifact keys`: Lists the unique filenames of artifacts within a given scope.
- `Delete Artifact`: Removes an artifact (and potentially all its versions, depending on implementation).
- `List versions`: Lists all available version numbers for a specific artifact filename.
- **Configuration:** You provide an instance of an artifact service (e.g., `InMemoryArtifactService`, `GcsArtifactService`) when initializing the `Runner`. The `Runner` then makes this service available to agents and tools via the `InvocationContext`.

PythonJava

```
from google.adk.runners import Runner
from google.adk.artifacts import InMemoryArtifactService # Or GcsArtif
from google.adk.agents import LlmAgent # Any agent
from google.adk.sessions import InMemorySessionService

# Example: Configuring the Runner with an Artifact Service
```

```
my_agent = LlmAgent(name="artifact_user_agent", model="gemini-2.0-flas
artifact_service = InMemoryArtifactService() # Choose an implementatio
session_service = InMemorySessionService()

runner = Runner(
    agent=my_agent,
    app_name="my_artifact_app",
    session_service=session_service,
    artifact_service=artifact_service # Provide the service instance h
)
# Now, contexts within runs managed by this runner can use artifact me
```

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.runner.Runner;
import com.google.adk.sessions.InMemorySessionService;
import com.google.adk.artifacts.InMemoryArtifactService;

// Example: Configuring the Runner with an Artifact Service
LlmAgent myAgent =  LlmAgent.builder()
  .name("artifact_user_agent")
  .model("gemini-2.0-flash")
  .build();
InMemoryArtifactService artifactService = new InMemoryArtifactService(
InMemorySessionService sessionService = new InMemorySessionService();

Runner runner = new Runner(myAgent, "my_artifact_app", artifactService
// Now, contexts within runs managed by this runner can use artifact m
```

Artifact Data¶

- **Standard Representation:** Artifact content is universally represented using the `google.genai.types.Part` object, the same structure used for parts of LLM messages.

- **Key Attribute (`inline_data`):** For artifacts, the most relevant attribute is `inline_data`, which is a `google.genai.types.Blob` object containing:
- `data` (`bytes`): The raw binary content of the artifact.
- `mime_type` (`str`): A standard MIME type string (e.g., `'application/pdf'`, `'image/png'`, `'audio/mpeg'`) describing the nature of the binary data. **This is crucial for correct interpretation when loading the artifact.**

PythonJava

```
import google.genai.types as types

# Example: Creating an artifact Part from raw bytes
pdf_bytes = b'%PDF-1.4...' # Your raw PDF data
pdf_mime_type = "application/pdf"

# Using the constructor
pdf_artifact_py = types.Part(
    inline_data=types.Blob(data=pdf_bytes, mime_type=pdf_mime_type)
)

# Using the convenience class method (equivalent)
pdf_artifact_alt_py = types.Part.from_bytes(data=pdf_bytes, mime_type=

print(f"Created Python artifact with MIME type: {pdf_artifact_py.inlin
```

```
import com.google.genai.types.Blob;
import com.google.genai.types.Part;
import java.nio.charset.StandardCharsets;

public class ArtifactDataExample {
  public static void main(String[] args) {
    // Example: Creating an artifact Part from raw bytes
```

```
    byte[] pdfBytes = "%PDF-1.4...".getBytes(StandardCharsets.UTF_8);
    String pdfMimeType = "application/pdf";

    // Using the Part.fromBlob() constructor with a Blob
    Blob pdfBlob = Blob.builder()
        .data(pdfBytes)
        .mimeType(pdfMimeType)
        .build();
    Part pdfArtifactJava = Part.builder().inlineData(pdfBlob).build();

    // Using the convenience static method Part.fromBytes() (equivalen
    Part pdfArtifactAltJava = Part.fromBytes(pdfBytes, pdfMimeType);

    // Accessing mimeType, note the use of Optional
    String mimeType = pdfArtifactJava.inlineData()
        .flatMap(Blob::mimeType)
        .orElse("unknown");
    System.out.println("Created Java artifact with MIME type: " + mime

    // Accessing data
    byte[] data = pdfArtifactJava.inlineData()
        .flatMap(Blob::data)
        .orElse(new byte[0]);
    System.out.println("Java artifact data (first 10 bytes): "
        + new String(data, 0, Math.min(data.length, 10), StandardChars
  }
}
```

Filename¶

- **Identifier:** A simple string used to name and retrieve an artifact within its specific namespace.
- **Uniqueness:** Filenames must be unique within their scope (either the session or the user namespace).

- **Best Practice:** Use descriptive names, potentially including file extensions (e.g., `"monthly_report.pdf"`, `"user_avatar.jpg"`), although the extension itself doesn't dictate behavior – the `mime_type` does.

Versioning¶

- **Automatic Versioning:** The artifact service automatically handles versioning. When you call `save_artifact`, the service determines the next available version number (typically starting from 0 and incrementing) for that specific filename and scope.
- **Returned by `save_artifact`:** The `save_artifact` method returns the integer version number that was assigned to the newly saved artifact.
- **Retrieval:**
- `load_artifact(..., version=None)` (default): Retrieves the *latest* available version of the artifact.
- `load_artifact(..., version=N)`: Retrieves the specific version `N`.
- **Listing Versions:** The `list_versions` method (on the service, not context) can be used to find all existing version numbers for an artifact.

Namespacing (Session vs. User)¶

- **Concept:** Artifacts can be scoped either to a specific session or more broadly to a user across all their sessions within the application. This scoping is determined by the `filename` format and handled internally by the `ArtifactService`.
- **Default (Session Scope):** If you use a plain filename like `"report.pdf"`, the artifact is associated with the specific `app_name`, `user_id`, *and* `session_id`. It's only accessible within that exact session context.
- **User Scope (`"user:"` prefix):** If you prefix the filename with `"user:"`, like `"user:profile.png"`, the artifact is associated only with the `app_name` and `user_id`. It can be accessed or updated from *any* session belonging to that user within the app.

PythonJava

```python
# Example illustrating namespace difference (conceptual)

# Session-specific artifact filename
session_report_filename = "summary.txt"

# User-specific artifact filename
user_config_filename = "user:settings.json"

# When saving 'summary.txt' via context.save_artifact,
# it's tied to the current app_name, user_id, and session_id.

# When saving 'user:settings.json' via context.save_artifact,
# the ArtifactService implementation should recognize the "user:" pref
# and scope it to app_name and user_id, making it accessible across se
```

```java
// Example illustrating namespace difference (conceptual)

// Session-specific artifact filename
String sessionReportFilename = "summary.txt";

// User-specific artifact filename
String userConfigFilename = "user:settings.json"; // The "user:" prefi

// When saving 'summary.txt' via context.save_artifact,
// it's tied to the current app_name, user_id, and session_id.
// artifactService.saveArtifact(appName, userId, sessionId1, sessionRe

// When saving 'user:settings.json' via context.save_artifact,
// the ArtifactService implementation should recognize the "user:" pre
// and scope it to app_name and user_id, making it accessible across s
// artifactService.saveArtifact(appName, userId, sessionId1, userConfi
```

These core concepts work together to provide a flexible system for managing binary data within the ADK framework.

Interacting with Artifacts (via Context Objects)¶

The primary way you interact with artifacts within your agent's logic (specifically within callbacks or tools) is through methods provided by the `CallbackContext` and `ToolContext` objects. These methods abstract away the underlying storage details managed by the `ArtifactService`.

Prerequisite: Configuring the `ArtifactService`¶

Before you can use any artifact methods via the context objects, you **must** provide an instance of a `BaseArtifactService` implementation (like `InMemoryArtifactService` or `GcsArtifactService`) when initializing your `Runner`.

PythonJava

In Python, you provide this instance when initializing your `Runner`.

```
from google.adk.runners import Runner
from google.adk.artifacts import InMemoryArtifactService # Or GcsArtif
from google.adk.agents import LlmAgent
from google.adk.sessions import InMemorySessionService

# Your agent definition
agent = LlmAgent(name="my_agent", model="gemini-2.0-flash")

# Instantiate the desired artifact service
artifact_service = InMemoryArtifactService()

# Provide it to the Runner
runner = Runner(
    agent=agent,
    app_name="artifact_app",
    session_service=InMemorySessionService(),
    artifact_service=artifact_service # Service must be provided here
```

```
)
```

If no `artifact_service` is configured in the `InvocationContext` (which happens if it's not passed to the `Runner`), calling `save_artifact`, `load_artifact`, or `list_artifacts` on the context objects will raise a `ValueError`.

In Java, you would instantiate a `BaseArtifactService` implementation and then ensure it's accessible to the parts of your application that manage artifacts. This is often done through dependency injection or by explicitly passing the service instance.

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.artifacts.InMemoryArtifactService; // Or GcsArti
import com.google.adk.runner.Runner;
import com.google.adk.sessions.InMemorySessionService;

public class SampleArtifactAgent {

  public static void main(String[] args) {

    // Your agent definition
    LlmAgent agent = LlmAgent.builder()
        .name("my_agent")
        .model("gemini-2.0-flash")
        .build();

    // Instantiate the desired artifact service
    InMemoryArtifactService artifactService = new InMemoryArtifactServ

    // Provide it to the Runner
    Runner runner = new Runner(agent,
        "APP_NAME",
        artifactService, // Service must be provided here
        new InMemorySessionService());
```

```
    }
}
```

In Java, if an `ArtifactService` instance is not available (e.g., `null`) when artifact operations are attempted, it would typically result in a `NullPointerException` or a custom error, depending on how your application is structured. Robust applications often use dependency injection frameworks to manage service lifecycles and ensure availability.

Accessing Methods¶

The artifact interaction methods are available directly on instances of `CallbackContext` (passed to agent and model callbacks) and `ToolContext` (passed to tool callbacks). Remember that `ToolContext` inherits from `CallbackContext`.

- **Code Example:**

PythonJava

``` import google.genai.types as types from google.adk.agents.callback_context import CallbackContext # Or ToolContext

async def save_generated_report_py(context: CallbackContext, report_bytes: bytes): """Saves generated PDF report bytes as an artifact.""" report_artifact = types.Part.from_data( data=report_bytes, mime_type="application/pdf" ) filename = "generated_report.pdf"

```
    try:
        version = await context.save_artifact(filename=filename, artifac
        print(f"Successfully saved Python artifact '{filename}' as versi
        # The event generated after this callback will contain:
        # event.actions.artifact_delta == {"generated_report.pdf": versi
    except ValueError as e:
        print(f"Error saving Python artifact: {e}. Is ArtifactService co
    except Exception as e:
```
```

```
        # Handle potential storage errors (e.g., GCS permissions)
        print(f"An unexpected error occurred during Python artifact save
```

--- Example Usage Concept (Python) --- # async def main_py(): # callback_context: CallbackContext = ... # obtain context # report_data = b'...' # Assume this holds the PDF bytes # await save_generated_report_py(callback_context, report_data)

```

``` import com.google.adk.agents.CallbackContext; import com.google.adk.artifacts.BaseArtifactService; import com.google.adk.artifacts.InMemoryArtifactService; import com.google.genai.types.Part; import java.nio.charset.StandardCharsets;

public class SaveArtifactExample {

public void saveGeneratedReport(CallbackContext callbackContext, byte[] reportBytes) { // Saves generated PDF report bytes as an artifact. Part reportArtifact = Part.fromBytes(reportBytes, "application/pdf"); String filename = "generatedReport.pdf";

```
    callbackContext.saveArtifact(filename, reportArtifact);
    System.out.println("Successfully saved Java artifact '" + filename);
    // The event generated after this callback will contain:
    // event().actions().artifactDelta == {"generated_report.pdf": versi
```

}

// --- Example Usage Concept (Java) --- public static void main(String[] args) { BaseArtifactService service = new InMemoryArtifactService(); // Or GcsArtifactService SaveArtifactExample myTool = new SaveArtifactExample(); byte[] reportData = "...".getBytes(StandardCharsets.UTF_8); // PDF bytes CallbackContext callbackContext; // ... obtain callback context from your app myTool.saveGeneratedReport(callbackContext, reportData); // Due to async nature, in a real app, ensure program waits or handles completion. } }

```
```

Loading Artifacts¶

- **Code Example:**

PythonJava

``` import google.genai.types as types from google.adk.agents.callback_context import CallbackContext # Or ToolContext

async def process_latest_report_py(context: CallbackContext): """Loads the latest report artifact and processes its data.""" filename = "generated_report.pdf" try: # Load the latest version report_artifact = await context.load_artifact(filename=filename)

```
        if report_artifact and report_artifact.inline_data:
            print(f"Successfully loaded latest Python artifact '{filenam
            print(f"MIME Type: {report_artifact.inline_data.mime_type}")
            # Process the report_artifact.inline_data.data (bytes)
            pdf_bytes = report_artifact.inline_data.data
            print(f"Report size: {len(pdf_bytes)} bytes.")
            # ... further processing ...
        else:
            print(f"Python artifact '{filename}' not found.")

        # Example: Load a specific version (if version 0 exists)
        # specific_version_artifact = await context.load_artifact(filena
        # if specific_version_artifact:
        #   print(f"Loaded version 0 of '{filename}'.")

    except ValueError as e:
        print(f"Error loading Python artifact: {e}. Is ArtifactService c
    except Exception as e:
        # Handle potential storage errors
        print(f"An unexpected error occurred during Python artifact load
```

# --- Example Usage Concept (Python) --- # async def main_py(): # callback_context: CallbackContext = ... # obtain context # await process_latest_report_py(callback_context)
```

```

``` import com.google.adk.artifacts.BaseArtifactService; import com.google.genai.types.Part; import io.reactivex.rxjava3.core.MaybeObserver; import io.reactivex.rxjava3.disposables.Disposable; import java.util.Optional;

public class MyArtifactLoaderService {

```
 private final BaseArtifactService artifactService;
private final String appName;

public MyArtifactLoaderService(BaseArtifactService artifactService,
    this.artifactService = artifactService;
    this.appName = appName;
}

public void processLatestReportJava(String userId, String sessionId,
    // Load the latest version by passing Optional.empty() for the v
    artifactService
            .loadArtifact(appName, userId, sessionId, filename, Opti
            .subscribe(
                    new MaybeObserver<Part>() {
                        @Override
                        public void onSubscribe(Disposable d) {
                            // Optional: handle subscription
                        }

                        @Override
                        public void onSuccess(Part reportArtifact) {
                            System.out.println(
                                    "Successfully loaded latest Java
                            reportArtifact
                                    .inlineData()
                                    .ifPresent(
                                            blob -> {
                                                System.out.println(
                                                        "MIME Type:
```

```
                                        byte[] pdfBytes = bl
                                        System.out.println("
                                        // ... further proce
                                    });
                        }

                        @Override
                        public void onError(Throwable e) {
                            // Handle potential storage errors or ot
                            System.err.println(
                                    "An error occurred during Java a
                                            + filename
                                            + "': "
                                            + e.getMessage());
                        }

                        @Override
                        public void onComplete() {
                            // Called if the artifact (latest versio
                            System.out.println("Java artifact '" + f
                        }
                    });

        // Example: Load a specific version (e.g., version 0)
        /*
        artifactService.loadArtifact(appName, userId, sessionId, filenam
            .subscribe(part -> {
                System.out.println("Loaded version 0 of Java artifact '"
            }, throwable -> {
                System.err.println("Error loading version 0 of '" + file
            }, () -> {
                System.out.println("Version 0 of Java artifact '" + file
            });
        */
    }
```

```
    // --- Example Usage Concept (Java) ---
    public static void main(String[] args) {
        // BaseArtifactService service = new InMemoryArtifactService();
        // MyArtifactLoaderService loader = new MyArtifactLoaderService(
        // loader.processLatestReportJava("user123", "sessionABC", "java
        // Due to async nature, in a real app, ensure program waits or h
    }
```

}

```

### Listing Artifact Filenames¶

- **Code Example:**

PythonJava

``` from google.adk.tools.tool_context import ToolContext

def list_user_files_py(tool_context: ToolContext) -> str: """Tool to list available artifacts for the user.""" try: available_files = await tool_context.list_artifacts() if not available_files: return "You have no saved artifacts." else: # Format the list for the user/LLM file_list_str = "\n".join([f"- {fname}" for fname in available_files]) return f"Here are your available Python artifacts:\n{file_list_str}" except ValueError as e: print(f"Error listing Python artifacts: {e}. Is ArtifactService configured?") return "Error: Could not list Python artifacts." except Exception as e: print(f"An unexpected error occurred during Python artifact list: {e}") return "Error: An unexpected error occurred while listing Python artifacts."

# This function would typically be wrapped in a FunctionTool # from google.adk.tools import FunctionTool # list_files_tool = FunctionTool(func=list_user_files_py)

```

``` import com.google.adk.artifacts.BaseArtifactService; import com.google.adk.artifacts.ListArtifactsResponse; import com.google.common.collect.ImmutableList; import

io.reactivex.rxjava3.core.SingleObserver; import
io.reactivex.rxjava3.disposables.Disposable;

public class MyArtifactListerService {

```
 private final BaseArtifactService artifactService;
private final String appName;

public MyArtifactListerService(BaseArtifactService artifactService,
    this.artifactService = artifactService;
    this.appName = appName;
}

// Example method that might be called by a tool or agent logic
public void listUserFilesJava(String userId, String sessionId) {
    artifactService
            .listArtifactKeys(appName, userId, sessionId)
            .subscribe(
                    new SingleObserver<ListArtifactsResponse>() {
                        @Override
                        public void onSubscribe(Disposable d) {
                            // Optional: handle subscription
                        }

                        @Override
                        public void onSuccess(ListArtifactsResponse
                            ImmutableList<String> availableFiles = r
                            if (availableFiles.isEmpty()) {
                                System.out.println(
                                        "User "
                                                + userId
                                                + " in session "
                                                + sessionId
                                                + " has no saved Jav
                            } else {
                                StringBuilder fileListStr =
```

```
                                        new StringBuilder(
                                                "Here are the availa
                                                        + userId
                                                        + " in sessi
                                                        + sessionId
                                                        + ":\n");
                                        for (String fname : availableFiles)
                                            fileListStr.append("- ").append(
                                        }
                                        System.out.println(fileListStr.toStr
                                    }
                                }

                                @Override
                                public void onError(Throwable e) {
                                    System.err.println(
                                            "Error listing Java artifacts fo
                                                    + userId
                                                    + " in session "
                                                    + sessionId
                                                    + ": "
                                                    + e.getMessage());
                                    // In a real application, you might retu
                                }
                            });
    }

    // --- Example Usage Concept (Java) ---
    public static void main(String[] args) {
        // BaseArtifactService service = new InMemoryArtifactService();
        // MyArtifactListerService lister = new MyArtifactListerService(
        // lister.listUserFilesJava("user123", "sessionABC");
        // Due to async nature, in a real app, ensure program waits or h
    }
```

}

```

These methods for saving, loading, and listing provide a convenient and consistent way to manage binary data persistence within ADK, whether using Python's context objects or directly interacting with the `BaseArtifactService` in Java, regardless of the chosen backend storage implementation.

## Available Implementations¶

ADK provides concrete implementations of the `BaseArtifactService` interface, offering different storage backends suitable for various development stages and deployment needs. These implementations handle the details of storing, versioning, and retrieving artifact data based on the `app_name`, `user_id`, `session_id`, and `filename` (including the `user:` namespace prefix).

### InMemoryArtifactService¶

- **Storage Mechanism:**
- Python: Uses a Python dictionary (`self.artifacts`) held in the application's memory. The dictionary keys represent the artifact path, and the values are lists of `types.Part`, where each list element is a version.
- Java: Uses nested `HashMap` instances (`private final Map<String, Map<String, Map<String, Map<String, List<Part>>>>> artifacts;`) held in memory. The keys at each level are `appName`, `userId`, `sessionId`, and `filename` respectively. The innermost `List<Part>` stores the versions of the artifact, where the list index corresponds to the version number.
- **Key Features:**
- **Simplicity:** Requires no external setup or dependencies beyond the core ADK library.
- **Speed:** Operations are typically very fast as they involve in-memory map/dictionary lookups and list manipulations.
- **Ephemeral:** All stored artifacts are **lost** when the application process terminates. Data does not persist between application restarts.
- **Use Cases:**
```

- Ideal for local development and testing where persistence is not required.
- Suitable for short-lived demonstrations or scenarios where artifact data is purely temporary within a single run of the application.
- **Instantiation:**

PythonJava

``` from google.adk.artifacts import InMemoryArtifactService

# Simply instantiate the class in_memory_service_py = InMemoryArtifactService()

# Then pass it to the Runner # runner = Runner(..., artifact_service=in_memory_service_py)

```

``` import com.google.adk.artifacts.BaseArtifactService; import com.google.adk.artifacts.InMemoryArtifactService;

public class InMemoryServiceSetup { public static void main(String[] args) { // Simply instantiate the class BaseArtifactService inMemoryServiceJava = new InMemoryArtifactService();

```
        System.out.println("InMemoryArtifactService (Java) instantiated

        // This instance would then be provided to your Runner.
        // Runner runner = new Runner(
        //   /* other services */,
        //   inMemoryServiceJava
        //);
    }
```

}

```

GcsArtifactService¶

- **Storage Mechanism:** Leverages Google Cloud Storage (GCS) for persistent artifact storage. Each version of an artifact is stored as a separate object (blob) within a specified GCS bucket.
- **Object Naming Convention:** It constructs GCS object names (blob names) using a hierarchical path structure.
- **Key Features:**
- **Persistence:** Artifacts stored in GCS persist across application restarts and deployments.
- **Scalability:** Leverages the scalability and durability of Google Cloud Storage.
- **Versioning:** Explicitly stores each version as a distinct GCS object. The `saveArtifact` method in `GcsArtifactService`.
- **Permissions Required:** The application environment needs appropriate credentials (e.g., Application Default Credentials) and IAM permissions to read from and write to the specified GCS bucket.
- **Use Cases:**
- Production environments requiring persistent artifact storage.
- Scenarios where artifacts need to be shared across different application instances or services (by accessing the same GCS bucket).
- Applications needing long-term storage and retrieval of user or session data.
- **Instantiation:**

PythonJava

``` from google.adk.artifacts import GcsArtifactService

# Specify the GCS bucket name gcs_bucket_name_py = "your-gcs-bucket-for-adk-artifacts" # Replace with your bucket name

try: gcs_service_py = GcsArtifactService(bucket_name=gcs_bucket_name_py) print(f"Python GcsArtifactService initialized for bucket: {gcs_bucket_name_py}") # Ensure your environment has credentials to access this bucket. # e.g., via Application Default Credentials (ADC)

```
 # Then pass it to the Runner
runner = Runner(..., artifact_service=gcs_service_py)
```
```

except Exception as e: # Catch potential errors during GCS client initialization (e.g., auth issues) print(f"Error initializing Python GcsArtifactService: {e}") # Handle the error appropriately - maybe fall back to InMemory or raise

```

``` import com.google.adk.artifacts.BaseArtifactService; import com.google.adk.artifacts.GcsArtifactService; import com.google.cloud.storage.Storage; import com.google.cloud.storage.StorageOptions;

public class GcsServiceSetup { public static void main(String[] args) { // Specify the GCS bucket name String gcsBucketNameJava = "your-gcs-bucket-for-adk-artifacts"; // Replace with your bucket name

```
try {
  // Initialize the GCS Storage client.
  // This will use Application Default Credentials by default.
  // Ensure the environment is configured correctly (e.g., GOOGLE_AP
  Storage storageClient = StorageOptions.getDefaultInstance().getSer

  // Instantiate the GcsArtifactService
  BaseArtifactService gcsServiceJava =
      new GcsArtifactService(gcsBucketNameJava, storageClient);

  System.out.println(
      "Java GcsArtifactService initialized for bucket: " + gcsBucket

  // This instance would then be provided to your Runner.
  // Runner runner = new Runner(
  //   /* other services */,
  //   gcsServiceJava
  //);

} catch (Exception e) {
  // Catch potential errors during GCS client initialization (e.g.,
  System.err.println("Error initializing Java GcsArtifactService: "
```

```
    e.printStackTrace();
    // Handle the error appropriately
  }
}
```

}

```

Choosing the appropriate `ArtifactService` implementation depends on your application's requirements for data persistence, scalability, and operational environment.

## Best Practices¶

To use artifacts effectively and maintainably:

- **Choose the Right Service:** Use `InMemoryArtifactService` for rapid prototyping, testing, and scenarios where persistence isn't needed. Use `GcsArtifactService` (or implement your own `BaseArtifactService` for other backends) for production environments requiring data persistence and scalability.
- **Meaningful Filenames:** Use clear, descriptive filenames. Including relevant extensions (`.pdf`, `.png`, `.wav`) helps humans understand the content, even though the `mime_type` dictates programmatic handling. Establish conventions for temporary vs. persistent artifact names.
- **Specify Correct MIME Types:** Always provide an accurate `mime_type` when creating the `types.Part` for `save_artifact`. This is critical for applications or tools that later `load_artifact` to interpret the `bytes` data correctly. Use standard IANA MIME types where possible.
- **Understand Versioning:** Remember that `load_artifact()` without a specific `version` argument retrieves the *latest* version. If your logic depends on a specific historical version of an artifact, be sure to provide the integer version number when loading.
- **Use Namespacing (`user:`) Deliberately:** Only use the `"user:"` prefix for filenames when the data truly belongs to the user and should be
```

accessible across all their sessions. For data specific to a single conversation or session, use regular filenames without the prefix.

- **Error Handling:**
- Always check if an `artifact_service` is actually configured before calling context methods (`save_artifact` , `load_artifact` , `list_artifacts`) – they will raise a `ValueError` if the service is `None` .
- Check the return value of `load_artifact` , as it will be `None` if the artifact or version doesn't exist. Don't assume it always returns a `Part` .
- Be prepared to handle exceptions from the underlying storage service, especially with `GcsArtifactService` (e.g., `google.api_core.exceptions.Forbidden` for permission issues, `NotFound` if the bucket doesn't exist, network errors).
- **Size Considerations:** Artifacts are suitable for typical file sizes, but be mindful of potential costs and performance impacts with extremely large files, especially with cloud storage. `InMemoryArtifactService` can consume significant memory if storing many large artifacts. Evaluate if very large data might be better handled through direct GCS links or other specialized storage solutions rather than passing entire byte arrays in-memory.
- **Cleanup Strategy:** For persistent storage like `GcsArtifactService` , artifacts remain until explicitly deleted. If artifacts represent temporary data or have a limited lifespan, implement a strategy for cleanup. This might involve:
- Using GCS lifecycle policies on the bucket.
- Building specific tools or administrative functions that utilize the `artifact_service.delete_artifact` method (note: delete is *not* exposed via context objects for safety).
- Carefully managing filenames to allow pattern-based deletion if needed.