Reference Guide

> **Description:** Quick reference for ADK patterns, configurations, and common implementations

Purpose: Quick reference for common ADK patterns, configurations, and implementations.

Source of Truth: google/adk-python/src/google/adk/ (https://github.com/google/adk-python/tree/main/src/google/adk/) (ADK 1.15) + production examples

Table of Contents

Use this guide to find code patterns you need for common ADK tasks. Each section includes ready-to-use code examples.

Agent Patterns

Problem: Build an agent that takes user input and generates responses

Solution

```
from google.adk.agents import Agent

agent = Agent(
    name="basic_agent",
    model="gemini-2.0-flash",
    instruction="You are a helpful assistant.",
    tools=[],  # Add tools here
    output_key="response"
)
```

Sequential Workflow

```
from google.adk.agents import SequentialAgent

workflow = SequentialAgent(
    name="data_pipeline",
    sub_agents=[extract_agent, transform_agent, load_agent],
    description="ETL data processing pipeline"
)
```

Parallel Processing

```
from google.adk.agents import ParallelAgent

parallel = ParallelAgent(
    name="multi_source_analysis",
    sub_agents=[web_agent, database_agent, api_agent],
    description="Gather data from multiple sources simultaneously"
)
```

Iterative Refinement

```
from google.adk.agents import LoopAgent

refiner = LoopAgent(
    sub_agents=[critic_agent, improvement_agent],
    max_iterations=5,
    description="Iteratively improve content quality"
)
```

🔧 Tool Implementations

Function Tool

```python
def search_database(query: str) -> Dict[str, Any]:
    """
    Search the database for relevant information.

    Args:
        query: Search query string

    Returns:
        Dict with status, report, and data
    """
    try:
        results = db.search(query)
        return {
            'status': 'success',
            'report': f'Found {len(results)} results for "{query}"',
            'data': results
        }
    except Exception as e:
        return {
            'status': 'error',
            'error': str(e),
            'report': f'Database search failed: {str(e)}'
        }
```

OpenAPI Tool

```
from google.adk.tools import OpenAPIToolset

# Auto-generate tools from OpenAPI spec
weather_tools = OpenAPIToolset.from_url(
    "https://api.weatherapi.com/v1/swagger.json",
    api_key="your_api_key"
)

agent = Agent(
    name="weather_agent",
    tools=weather_tools,
    instruction="Provide weather information and forecasts."
)
```

MCP Tool

```
from google.adk.tools import MCPToolset
from google.adk.tools.mcp import StdioConnectionParams

# Connect to MCP server
filesystem_tools = MCPToolset(
    connection_params=StdioConnectionParams(
        command='npx',
        args=['-y', '@modelcontextprotocol/server-filesystem', '/data']
    )
)
```

💾 State Management

State Scopes

```
# Session state (conversation-scoped)
state['session:user_query'] = query
state['session:conversation_history'] = messages

# User state (cross-session user data)
state['user:preferences'] = user_prefs
state['user:subscription_tier'] = 'premium'

# App state (global application data)
state['app:version'] = '1.0.0'
state['app:feature_flags'] = flags

# Temporary state (request-only)
state['temp:cache'] = cached_data
```

State Interpolation

```
agent = Agent(
    name="personalized_agent",
    instruction=f"""
    Welcome back {state['user:name']}!
    Your last query was: {state['session:last_query']}

    Current preferences: {state['user:preferences']}
    """,
    tools=[personalized_tools]
)
```

🚀 Deployment Configurations

Local Development

```
# Install ADK
pip install google-adk

# Run web interface
adk web

# Run specific agent
adk web agent_name
```

Cloud Run Deployment

```
# Dockerfile
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["adk", "serve", "--host", "0.0.0.0", "--port", "8000"]
```

```
# Deploy
gcloud run deploy agent-service \
  --source . \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated
```

Vertex AI Agent Engine

```yaml
# agent.yaml
name: my-agent
description: My ADK Agent
model: gemini-2.0-flash
instruction: You are a helpful assistant.
tools: []
```

```bash
# Deploy to Agent Engine
adk deploy agent_engine --config agent.yaml
```

📊 Monitoring & Observability

Event Tracking

```python
from google.adk.observability import EventTracker

tracker = EventTracker()

# Track agent interactions
@tracker.track_agent_calls
def run_agent(query):
    result = agent.run(query)
    return result

# Custom events
tracker.track_event(
    event_type="user_interaction",
    properties={
        "query_length": len(query),
        "response_time": response_time,
        "agent_name": agent.name
    }
)
```

Error Handling

```
from google.adk.error_handling import ErrorHandler

error_handler = ErrorHandler()

@error_handler.catch_and_log
def safe_agent_call(query):
    try:
        return agent.run(query)
    except Exception as e:
        # Log error and return fallback response
        logger.error(f"Agent error: {e}")
        return {"error": "Service temporarily unavailable"}
```

🧪 Testing Patterns

Unit Test Structure

```
import pytest
from unittest.mock import Mock, patch

class TestMyAgent:

    def test_agent_initialization(self):
        agent = Agent(name="test", model="gemini-2.0-flash")
        assert agent.name == "test"
        assert agent.model == "gemini-2.0-flash"

    @patch('google.adk.agents.Agent.run')
    def test_agent_response(self, mock_run):
        mock_run.return_value = {"response": "Hello!"}

        agent = Agent(name="test")
        result = agent.run("Hi")

        assert result["response"] == "Hello!"
        mock_run.assert_called_once_with("Hi")

    def test_tool_execution(self):
        # Test tool functions in isolation
        result = search_database("test query")
        assert result["status"] in ["success", "error"]
        assert "report" in result
```

Integration Testing

```
class TestAgentIntegration:

    def test_full_workflow(self):
        # Test complete agent workflows
        workflow = create_etl_pipeline()
        result = workflow.run(test_data)

        assert result["status"] == "success"
        assert len(result["processed_data"]) > 0

    def test_api_integration(self):
        # Test external API integrations
        with patch('requests.get') as mock_get:
            mock_get.return_value.json.return_value = {"weather": "sunny"}

            result = weather_agent.run("What's the weather?")
            assert "sunny" in result["response"]
```

⚙️ Configuration Templates

Environment Variables

```
# .env file
GOOGLE_API_KEY=your_api_key_here
OPENAI_API_KEY=your_openai_key
DATABASE_URL=postgresql://user:pass@localhost/db
REDIS_URL=redis://localhost:6379
LOG_LEVEL=INFO
```

YAML Configuration

```yaml
# config.yaml
agent:
  name: production_agent
  model: gemini-2.0-flash
  temperature: 0.7
  max_tokens: 1000

tools:
  - type: openapi
    url: https://api.example.com/swagger.json
    api_key: ${API_KEY}

  - type: function
    name: search_database
    function: myapp.tools.search_database

deployment:
  type: cloud_run
  region: us-central1
  memory: 1Gi
  cpu: 1

monitoring:
  enable_events: true
  log_level: INFO
  metrics:
    - response_time
    - error_rate
    - token_usage
```

🚨 Common Issues & Solutions

Agent Not Responding

Problem: Agent returns empty or null responses
Solutions:

- Check API key validity

- Verify model name spelling
- Ensure proper instruction formatting
- Check rate limits

Tool Execution Failures

Problem: Tools return error status
Solutions:

- Validate tool function signatures
- Check external service connectivity
- Verify authentication credentials
- Review error logs for details

State Persistence Issues

Problem: State not persisting between calls
Solutions:

- Use correct state scope prefixes
- Check state backend configuration
- Verify session/user identification
- Review state serialization

Performance Problems

Problem: Slow response times
Solutions:

- Switch to faster models (flash variants)
- Implement parallel processing
- Add caching layers
- Optimize tool implementations

API Reference

Core Classes

Class	Purpose	Key Methods
`Agent`	Basic agent implementation	`run()`, `run_async()`
`SequentialAgent`	Ordered workflow execution	`add_agent()`, `run()`
`ParallelAgent`	Concurrent task execution	`add_agent()`, `run()`
`LoopAgent`	Iterative refinement	`set_max_iterations()`, `run()`
`RemoteA2aAgent`	Distributed agent communication	`connect()`, `run()`

Tool Classes

Class	Purpose	Key Features
`FunctionTool`	Custom Python functions	Error handling, type hints
`OpenAPIToolset`	REST API integration	Auto-generation, auth
`MCPToolset`	Protocol-based tools	Interoperability, security
`AgentTool`	Agent-as-tool pattern	Composition, delegation

State Management

Scope	Lifetime	Use Case
`session:`	Conversation	Context, history
`user:`	User account	Preferences, settings
`app:`	Application	Global config, features
`temp:`	Request	Caching, temporary data

🎯 Quick Start Templates

Hello World Agent

```
from google.adk.agents import Agent

# Minimal agent
agent = Agent(
    name="hello_world",
    model="gemini-2.0-flash",
    instruction="You are a friendly assistant. Greet users warmly."
)

# Run locally
if __name__ == "__main__":
    result = agent.run("Hello!")
    print(result)
```

Tool-Enabled Agent

```python
from google.adk.agents import Agent

def get_weather(city: str) -> dict:
    # Implement weather API call
    return {"status": "success", "temperature": 72, "conditions": "sunny"}

agent = Agent(
    name="weather_assistant",
    model="gemini-2.0-flash",
    instruction="Provide weather information using the get_weather tool.",
    tools=[get_weather]
)
```

Workflow Agent

```python
from google.adk.agents import SequentialAgent, Agent

# Individual agents
researcher = Agent(name="researcher", instruction="Gather information")
writer = Agent(name="writer", instruction="Create content")
editor = Agent(name="editor", instruction="Review and improve")

# Composed workflow
workflow = SequentialAgent(
    name="content_pipeline",
    sub_agents=[researcher, writer, editor],
    description="Research → Write → Edit content pipeline"
)
```

🔗 Additional Resources

- **Official Documentation**: `https://github.com/google/adk-python/tree/main/docs`
- **Tutorial Implementations**: `tutorial_implementation/`
- **Research & Examples**: `research/`

- **Community Forums**: GitHub Issues, Stack Overflow
- **API Reference**: Inline code documentation
- **Contact the Author (contact.md)**: Get in touch with Raphaël MANSUY

🎯 **This completes the Mental Models section. Return to the main overview (overview.md) for navigation.**

Generated on 2025-10-21 09:03:27 from reference-guide.md

Source: Google ADK Training Hub