Runtime Config - Agent Development Kit

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

Runtime Configuration¶

`RunConfig` defines runtime behavior and options for agents in the ADK. It controls speech and streaming settings, function calling, artifact saving, and limits on LLM calls.

When constructing an agent run, you can pass a `RunConfig` to customize how the agent interacts with models, handles audio, and streams responses. By default, no streaming is enabled and inputs aren't retained as artifacts. Use `RunConfig` to override these defaults.

Class Definition¶

The `RunConfig` class holds configuration parameters for an agent's runtime behavior.

- Python ADK uses Pydantic for this validation.
- Java ADK typically uses immutable data classes.

PythonJava

```
class RunConfig(BaseModel):
    """Configs for runtime behavior of agents."""

    model_config = ConfigDict(
        extra='forbid',
    )

    speech_config: Optional[types.SpeechConfig] = None
    response_modalities: Optional[list[str]] = None
    save_input_blobs_as_artifacts: bool = False
    support_cfc: bool = False
    streaming_mode: StreamingMode = StreamingMode.NONE
    output_audio_transcription: Optional[types.AudioTranscriptionConfi
    max_llm_calls: int = 500
```

```
public abstract class RunConfig {

  public enum StreamingMode {
    NONE,
    SSE,
    BIDI
  }

  public abstract @Nullable SpeechConfig speechConfig();

  public abstract ImmutableList<Modality> responseModalities();

  public abstract boolean saveInputBlobsAsArtifacts();

  public abstract @Nullable AudioTranscriptionConfig outputAudioTransc

  public abstract int maxLlmCalls();

  // ...
}
```

Runtime Parameters¶

Parameter	Python Type
`speech_config`	`Optional[types.SpeechConfig]`
`response_modalities`	`Optional[list[str]]`

Parameter	Python Type
`save_input_blobs_as_artifacts`	`bool`
`streaming_mode`	`StreamingMode`
`output_audio_transcription`	`Optional[types.AudioTranscriptionConfi`
`max_llm_calls`	`int`
`support_cfc`	`bool`

`speech_config`¶

Note

The interface or definition of `SpeechConfig` is the same, irrespective of the language.

Speech configuration settings for live agents with audio capabilities. The `SpeechConfig` class has the following structure:

```
class SpeechConfig(_common.BaseModel):
    """The speech generation configuration."""

    voice_config: Optional[VoiceConfig] = Field(
        default=None,
        description="""The configuration for the speaker to use.""",
    )
    language_code: Optional[str] = Field(
        default=None,
        description="""Language code (ISO 639. e.g. en-US) for the spe
        Only available for Live API.""",
    )
```

The `voice_config` parameter uses the `VoiceConfig` class:

```
class VoiceConfig(_common.BaseModel):
    """The configuration for the voice to use."""

    prebuilt_voice_config: Optional[PrebuiltVoiceConfig] = Field(
        default=None,
        description="""The configuration for the speaker to use.""",
    )
```

And `PrebuiltVoiceConfig` has the following structure:

```
class PrebuiltVoiceConfig(_common.BaseModel):
    """The configuration for the prebuilt speaker to use."""

    voice_name: Optional[str] = Field(
        default=None,
        description="""The name of the prebuilt voice to use.""",
    )
```

These nested configuration classes allow you to specify:

- `voice_config`: The name of the prebuilt voice to use (in the `PrebuiltVoiceConfig`)
- `language_code`: ISO 639 language code (e.g., "en-US") for speech synthesis

When implementing voice-enabled agents, configure these parameters to control how your agent sounds when speaking.

`response_modalities`¶

Defines the output modalities for the agent. If not set, defaults to AUDIO. Response modalities determine how the agent communicates with users through various channels (e.g., text, audio).

`save_input_blobs_as_artifacts`¶

When enabled, input blobs will be saved as artifacts during agent execution. This is useful for debugging and audit purposes, allowing developers to review the exact data received by agents.

`support_cfc`¶

Enables Compositional Function Calling (CFC) support. Only applicable when using StreamingMode.SSE. When enabled, the LIVE API will be invoked as only it supports CFC functionality.

Warning

The `support_cfc` feature is experimental and its API or behavior might change in future releases.

`streaming_mode`¶

Configures the streaming behavior of the agent. Possible values:

- `StreamingMode.NONE`: No streaming; responses delivered as complete units
- `StreamingMode.SSE`: Server-Sent Events streaming; one-way streaming from server to client
- `StreamingMode.BIDI`: Bidirectional streaming; simultaneous communication in both directions

Streaming modes affect both performance and user experience. SSE streaming lets users see partial responses as they're generated, while BIDI streaming enables real-time interactive experiences.

`output_audio_transcription`¶

Configuration for transcribing audio outputs from live agents with audio response capability. This enables automatic transcription of audio responses for accessibility, record-keeping, and multi-modal applications.

`max_llm_calls`¶

Sets a limit on the total number of LLM calls for a given agent run.

- Values greater than 0 and less than `sys.maxsize`: Enforces a bound on LLM calls
- Values less than or equal to 0: Allows unbounded LLM calls *(not recommended for production)*

This parameter prevents excessive API usage and potential runaway processes. Since LLM calls often incur costs and consume resources, setting appropriate limits is crucial.

Validation Rules¶

The `RunConfig` class validates its parameters to ensure proper agent operation. While Python ADK uses `Pydantic` for automatic type validation, Java ADK relies on its static typing and may include explicit checks in the RunConfig's construction. For the `max_llm_calls` parameter specifically:

1. Extremely large values (like `sys.maxsize` in Python or `Integer.MAX_VALUE` in Java) are typically disallowed to prevent issues.
2. Values of zero or less will usually trigger a warning about unlimited LLM interactions.

Examples¶

Basic runtime configuration¶

PythonJava

```
from google.genai.adk import RunConfig, StreamingMode

config = RunConfig(
    streaming_mode=StreamingMode.NONE,
    max_llm_calls=100
)
```

```
import com.google.adk.agents.RunConfig;
import com.google.adk.agents.RunConfig.StreamingMode;

RunConfig config = RunConfig.builder()
        .setStreamingMode(StreamingMode.NONE)
        .setMaxLlmCalls(100)
        .build();
```

This configuration creates a non-streaming agent with a limit of 100 LLM calls, suitable for simple task-oriented agents where complete responses are preferable.

Enabling streaming¶

PythonJava

```
from google.genai.adk import RunConfig, StreamingMode

config = RunConfig(
    streaming_mode=StreamingMode.SSE,
    max_llm_calls=200
)
```

```
import com.google.adk.agents.RunConfig;
import com.google.adk.agents.RunConfig.StreamingMode;

RunConfig config = RunConfig.builder()
    .setStreamingMode(StreamingMode.SSE)
    .setMaxLlmCalls(200)
    .build();
```

Using SSE streaming allows users to see responses as they're generated, providing a more responsive feel for chatbots and assistants.

Enabling speech support¶

PythonJava

```
from google.genai.adk import RunConfig, StreamingMode
from google.genai import types

config = RunConfig(
    speech_config=types.SpeechConfig(
        language_code="en-US",
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name="Kore"
            )
        ),
    ),
    response_modalities=["AUDIO", "TEXT"],
    save_input_blobs_as_artifacts=True,
    support_cfc=True,
    streaming_mode=StreamingMode.SSE,
    max_llm_calls=1000,
)
```

```
import com.google.adk.agents.RunConfig;
import com.google.adk.agents.RunConfig.StreamingMode;
import com.google.common.collect.ImmutableList;
import com.google.genai.types.Content;
import com.google.genai.types.Modality;
import com.google.genai.types.Part;
import com.google.genai.types.PrebuiltVoiceConfig;
import com.google.genai.types.SpeechConfig;
import com.google.genai.types.VoiceConfig;

RunConfig runConfig =
    RunConfig.builder()
        .setStreamingMode(StreamingMode.SSE)
        .setMaxLlmCalls(1000)
        .setSaveInputBlobsAsArtifacts(true)
        .setResponseModalities(ImmutableList.of(new Modality("AUDIO"),
        .setSpeechConfig(
            SpeechConfig.builder()
                .voiceConfig(
                    VoiceConfig.builder()
                        .prebuiltVoiceConfig(
                            PrebuiltVoiceConfig.builder().voiceName("K
                        .build())
                .languageCode("en-US")
```

```
                    .build())
            .build();
```

This comprehensive example configures an agent with:

- Speech capabilities using the "Kore" voice (US English)
- Both audio and text output modalities
- Artifact saving for input blobs (useful for debugging)
- Experimental CFC support enabled **(Python only)**
- SSE streaming for responsive interaction
- A limit of 1000 LLM calls

Enabling Experimental CFC Support¶

Currently supported in Python

```
 from google.genai.adk import RunConfig, StreamingMode

config = RunConfig(
    streaming_mode=StreamingMode.SSE,
    support_cfc=True,
    max_llm_calls=150
)
```

Enabling Compositional Function Calling creates an agent that can dynamically execute functions based on model outputs, powerful for applications requiring complex workflows.