ARK Assistant SDK User Guide

ARK Assistant SDK

User Guide for Android

V1.4

Contents

1 COPYRIGHT NOTICE

Published by Cerence, Inc.

15 Wayside Road
Burlington, MA-01803
U.S.A.

2 OVERVIEW

2.1 Purpose of this Document

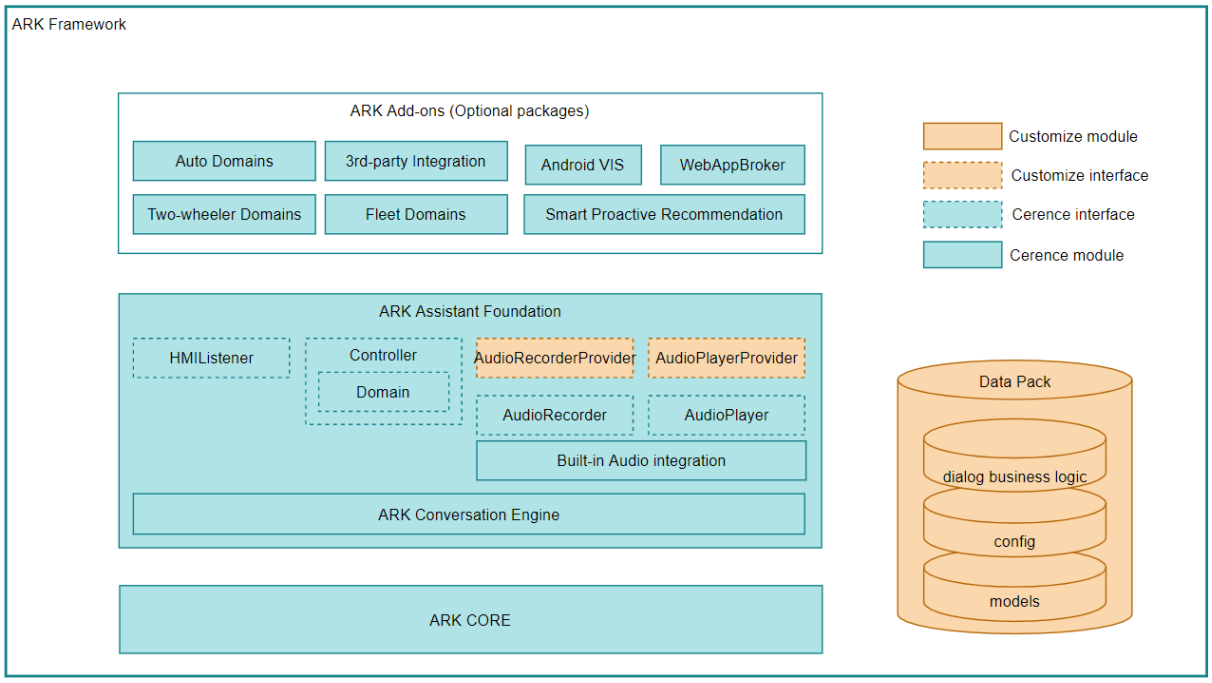
This document is used as developer user guide for developing voice assistant application based on ARK Assistant SDK. It will give an overview of ARK Assistant SDK, the interfaces used for communicating with dialog system.

2.2 Introduction

ARK Assistant SDK is a speech dialog system solution which integrated the following core technologies/features:

- Hybrid speech recognition.
- Text to speech engine.
- Speech signal enhancement (SSE) for noise suppression and echo cancellation.
- Built-in well-designed dialog flow (VUI).
- Built-in dialog arbitration.
- Supports contextual dialog and cross domain.
- Built-in audio recording/playback integration.
- Built-in wake up word detection.
- Built-in Cerence cloud service connection for cloud content integration.

Overview of the architecture



- ARK CORE is the component which integrates the core engines (SSE, ASR, NLU, TTS, etc.)
- ARK Conversation Engine is the component which manages the speech dialog. It is the base of the ARK Assistant Foundation.
- ARK Assistant Foundation is the component which handles dialog requests & notifications and provides smart functions (e.g., SPR, WebAppBroker).
- ARK add-ons are optional packages to make the integration more easily. It
  - Translates the request/response in Json format into Java object
  - Provides the default implementation for task completion by using OS's standard APIs if available.
  - Provides the default implementation for integrating with 3rd party application.
  - Provides the implementation for Android Voice Interaction Service for android assistant application compliant.
- Data Pack contains the files loaded by Assistant SDK at runtime, it contains:
  - Dialog flow implementation.
  - Models for ASR, NLU, TTS etc.
  - Configuration files.

Advantages of using ARK Assistant SDK:

- Easy to integrate into an application. Developers can build an assistant application very quickly.
- Developer does not need to know the details of speech core technologies.
- Ready to use well-designed dialog flow.
- Built-in default implementations for domains based on standard Android API.
- Developers can override the default implementations very easily.
- Developers only focus on the implementation of user requested actions.
- Android Automotive compatible.

3 SYSTEM REQUIREMENTS

To achieve a good performance for an application which integrated ARK Assistant SDK, the system at least meets the following requirements:

3.1 Audio

- Supports 16K 16bit mono audio input.
- Supports 22K 16bit mono audio output.
- If application wants integrate SSE echo cancellation algorithm, multi-channel audio signal input must be supported with at least one channel reference signal.
- If application wants integrate SSE beam forming algorithm, multi-channel mic. audio signal input must be supported.
- APQM (Audio Path Quality Measurement) should be performed by Cerence audio expert to check the audio requirement in more professional way.

3.2 OS

Android 9.0 and above.

3.3 Recommended hardware specification.

- 2GB flash available
- 4GB RAM
- Cortex A53, 4 cores with 1.5 GHz

3.4 Footprint

Exact memory usage on platform depends on the actual platform design, languages, complexity of voice recognition tasks, quality of TTS voice etc. The following data is measured on Android platform with language eng-USA.

3.4.1 SDK Libraries

ARK Assistant SDK	RAM	FLASH
Libs (plus core engine libs)	Depends on application	< 80 MB

3.4.2 ASR Data

The table gives the information of Cerence ASR required data size. The total size of data depends on the complexity of recognition tasks. E.g., if the application just wants to perform basic command recognition, the grammar might be the option for the recognition tasks, if the application wants to perform embedded dictation and support many domains (e.g., Embedded UDE, Embedded Music etc.), the statistical language model, guest contexts buffer and NLU model should be used for such complex recognition tasks.

Cerence ASR	RAM	FLASH
Acoustic Model	9.4 MB	9.4M
CLC	4.5 MB	4.5 MB
Context	Variable 4 KB for a context for recognition of digits, 5 – 10 MB for a context for recognition of a music selection of up to 5000 simultaneously active music items	
Grammars	Varies based on grammar compilation options, estimate about 400 kB for a 1000 words grammar in which 400 words are active.	
Exception Dictionaries	Depends on the number of exception dictionary entries.	
Language model	Statistical language model, the size varies based on the supported domains, for embedded dictation task, the estimated size is about 170 MB	
NLU Model	Varies based on the supported domains and commands for each domain.	

3.4.3 TTS Data

Embedded TTS data size varies based on the voice quality, the larger size, the better quality. The table gives the information of the size for voice “Ava(enu)” with operating point “embedded pro” (the smallest size for voice “Ava”).

Cerence TTS Voice	RAM	FLASH
enu_ava-ml_22_embedded-pro_2-1-0	65 MB	65 MB
Tuning files	Varies based on the number of prompts	

4 CONTENT OF THE SDK

The delivery of the SDK contains the following contents.

4.1 Android archive

The .aar files which contains all the necessary libs and Android resource files.

4.2 Resource data

The resource package which contains all the data for the dialog system. Those resource files will be loaded by the SDK at runtime. Developer should put the resource root folder into a readable location on the device. We suggest that package the resource files as .obb file and mount the obb file at runtime by application. For more information regarding obb file, please refer to https://developer.android.com/google/play/expansion-files.

4.3 Javadoc

Java API documentation in HTML format.

4.4 ARK Assistant Conversation API doc

The document describes the interfaces and JSON format used for communication between dialog system and application.

4.5 Sample application

The sample application to demonstrate how to integrate the SDK.

5 COMMUNICATION WITH DIALOG SYSTEM

5.1 Request and Response

5.1.1 Request

- Request represents an action triggered by system or users' interaction needs to be executed by the application.
- Different request may have different parameters which depends on what action needs to be performed by the application.
- There are two types of request in the system, one is HMI request which request application to show some GUI to user, such as show picklist, show weather information etc. Another one is task request which needs application to complete specific task, such as turn on the AC, play music, etc.
- Each request has a timeout setting. If application does not send the response within the timeout time, abort request will be sent to application for aborting the previous request.

5.1.2 Response

- Response represents the execution result for the requested action, such as whether AC is turned on successfully, is there any music for playing etc. Besides the execution status, some response may also contain the data which requested by the dialog system, such as POI search list provided by navigation application, phone number of the selected contact etc.
- What response should be replied to the system depends on what request received by the application.

5.1.3 JSON schema

Requests and Responses are expressed by JSON string like following:

```
{
 "id": "number",
 "name": "getPhoneNumber",
 "domain": "Phone",
 "type": "REQUEST",
 "timeout": 5000,
 "params": {
  "contact_id": "number",
  "contact_name": "string"
 },
 "results": {
  "result_code": "ResultCode_enum",
  "error_msg": "string",
  "data": {
   "phone_number_list": [
    {
     "number": "string",
```

```
        "number_type": "PhoneNumberType_enum",
        "carrier": "PhoneCarrier_enum"
       }
      ]
     }
    }
   }
}
```

- id: The unique id for each request.
- name: The name of the request.
- domain: The domain of the request.
- type: The type of the JSON string.
- timeout: Timeout setting for this request, if timeout occurred, dialog system will send abort request to application to abort the ongoing request if possible.
- params: Parameters of the request.
- results: Results is the response that application should send back to dialog system for next step processing. If application failed to do so, it will result in non-responsive issue.
- result_code: The result code of the execution for the request.
- error_msg: The error message in string, the detailed error information that application wants to prompt to user.
- data: The parameters in the response.

5.1.4 Note

After a request is sent out to the application, the dialog system will wait for the response from the application before proceeds to the next step. **To achieve the best performance, it is very important to send back corresponding response to dialog system when it is available in timely manner**. Otherwise, the dialog system will be stuck in waiting response state and the application will become non-responsive.

5.2 Notification

5.2.1 Overview

- Notification is the message generated by the dialog system and will be sent to application to notify the application what happened in the dialog system. Such as ASR result, dialog state, energy level of the input audio etc.
- Application could make use of its interested notifications to implement the graphic user interface.
- Unlike request, application does **NOT** need to send response back to the system for the received notification.

5.2.2 JSON schema

Notification is expressed by JSON string like following:

```
{
 "name": "AsrResultUpdated",
 "type": "NOTIFICATION",
 "params": {
  "utterance": "string",
  "is_final": "boolean"
 }
}
```

- name: The name of the notification
- type: The type of the JSON string.
- params: Parameters of the notification.

5.3 AppEvent

5.3.1 Overview

AppEvent represents event happened in the application which needs to notify the dialog system by sending corresponding event to the dialog system, e.g., user clicked PTT button, user selected one item in the picklist by haptic etc. Application is responsible for sending the correct AppEvent to dialog system when corresponding event happened.

5.3.2 JSON schema

AppEvent is expressed by JSON string like following:

```
{
 "name": "PTT",
 "interrupt_dialog": “boolean”,
 "params": {}
}
```

- name: The name of the AppEvent.
- interrupt_dialog: The boolean value indicates if the AppEvent will interrupt the dialog, if it is true, dialog system will abort the ongoing conversation and process the AppEvent immediately. If it is false, dialog system will handle the AppEvent silently without interrupting the ongoing conversation.
- data: The data for the AppEvent.

6 CONCEPT

6.1 RequestHandler

The speech request is represented by a JSON string which contains the request name and parameters like following:

```
{
        "name": "playMusic",
        "domain": "Music",
        "id": "number",
        "type": "REQUEST",
        "params": {
                "title": "string",
                "artist": "string",
                "album": "string",
                "genre": "string",
                "search_phrase": "string",
                "source": "MediaSource_enum"
        },
        "results": {
                "result_code": "ResultCode_enum",
                "error_msg": "string",
                "data": {}
        }
}
```

RequestHandler is an interface used for processing the received request from dialog system by a method call and send the result as response to the dialog system. Application developer implements the method according to the received parameter to complete the user requested task and send the response to dialog system for next step.

6.2 Domain

A Domain represents the host domain-specific business logic. It is used for linking the request with the RequestHandler by using a HashMap (The key is the name of the request, and the value is the RequestHandler object). Each request has corresponding RequestHandler object linked with it. In other words, each request has corresponding method/function to process it and send the response to dialog system.
SDK provides standard implementation for some domains (such as phone domain), if the standard implementation does not meet application's requirements, developer can override the standard implementation by extending the desired domain class.

Developers can also override the default RequestHandler implemented by the SDK in the existing Domain or create their own Domain to handle their interested requests by the registered RequestHandler.

6.3 HmiListener

HmiListener is used for receiving:

- GUI rendering request (e.g., show a picklist, show the weather information etc.) from dialog.
- Notifications (e.g., Dialog state, ASR result etc.) from dialog system.

6.4 Controller

Controller helps building the relationship between domain and its implementation by the HashMap (the key is domain name, and the value is the Domain object). With the relationship, the system knows which speech request should be handled by which Domain, then the Domain implementation helps to translate the request into method call according to the request name and registered RequestHandler.

6.5 Dhi

In most cases, the Hmi request is from dialog, however, Domain implementation still has the requirement of requesting rending the GUI (e.g., after handling the music play request, the application wants to display the music information to user.). Dhi defines the Domain Hmi Interface (Dhi) which is the bridge between the Domain and the Hmi. Each Domain hold the Dhi object to make the Hmi request through the HmiListener.

6.6 SpeechService

SpeechService is an Android Service, it is a bridge between dialog system and application layer. It can be configured to run in a separated process. The system should make sure the SpeechService is alive if the assistant application is active. The declaration of the service in Android manifest:

```
<application>
   <service android:name="cerence.ark.assistant.service.SpeechService" />
</application>
```

6.7 DCP

Dynamic content provider provides a mechanism to get the dynamic data (such as contacts, music, location etc.) for voice recognition from the device. The SDK has the default implementation of DCP by using standard Android API for data acquiring. Developer can override the default implementation if it does not meet the requirement, or the OS has specific APIs for data acquiring.

If application provides its own implementation of DCP, it is important to provide the correct data in timely manner, otherwise the dialog system will not be able to recognize the updated content.

Application is responsible for observing the data change and notifying the dialog system to update the content through the API of the SDK.

6.8 Prompter

Prompter provides the interfaces for generating speech audio data from text and play the audio data via device's speaker.

It is designed to be used by application for prompting only scenario, e.g., turn by turn navigation prompting.

Although the SDK provides the default implementation for audio playback via speaker, application can override the default implementation by providing its own audio player which implements IAudioPlayer through PromptPlayerProvider.

Even if application could play prompt at any time, the care must be taken that when is suitable to play a prompt. E.g., while a user is interacting with the assistant, trying to play a prompt by Prompter may interrupt the ongoing conversation between user and assistant. The best practice is that cancel the conversation between user and assistant before playing an urgent prompt while queuing the low priority prompt for later playing when assistant is in IDLE or LISTENING_WUW state.

7 DIALOG STATE

7.1 Overview

Dialog state represents the current state of the dialog system, the dialog system only has one state at a time. Application could make use of these states for GUI rendering or animation. Dialog system defines the following states:

- **IDLE**: In this state, dialog system is doing nothing, neither recognition nor prompting. If application set the interaction mode to “PTT”, dialog system will enter this state if there is no active conversation with dialog system.

- **LISTENING_WUW**: The system is listening for wakeup word. If application set the interaction mode to “WUW”, dialog system will enter this state if there is no active conversation with dialog system.

- **LISTENING_CMD**: The system is listening for user’s command. Dialog system will enter this state if system received “PTT” event or is woken up by wakeup word.

- **CAPTURING:** Dialog system will enter this state when the system detects that user has started speaking.

- **PROCESSING:** Dialog system will enter this state when the system detects that user has stopped speaking. In this state, the system will wait for recognition result and process it, execute the dialog flow (like play prompt, send request to application for task completion). In this state, the system is not listening user’s speech input.

7.2 JSON schema

Dialog state will be sent to application as notification JSON string when state changed like following:

```
{
 "name": "DialogStateChanged",
 "type": "NOTIFICATION",
 "params": {
  "state": "DialogState_enum"
 }
}
```

7.3 Difference on barge-in mode

There is a slight difference on barge-in mode settings.

7.3.1 Barge-in off

If application switches off the barge-in mode, that means the system will not listen to user's speech input while the system is prompting. The PROCESSING state will last until dialog system finishes prompting, then the dialog state will change from PROCESSING to the next state (one of LISTENING_WUW, LISTENING_CMD and IDLE).

7.3.2 Barge-in on

If application switches on the barge-in mode, that means user can input his/her speech while the system is prompting. The dialog state will immediately change from PROCESSING to the next state (one of LISTENING_WUW, LISTENING_CMD and IDLE) if the requested action (if it has) has been completed by the application, regardless of whether there is ongoing prompting or not.

8 DIALOG EVENT

8.1 Overview

Unlike dialog state, dialog event is more like an instantaneous state, such as the begin/end of conversation, begin/end of prompting etc. Application could also make use of the dialog event to control its GUI if applicable. Dialog system defines the following events:

- **CONVERSATION_BEGIN and CONVERSATION_END**: CONVERSATION_BEGIN indicates that the dialog system starts to listen/talk to user. CONVERSATION_BEGIN is usually triggered by wakeup word or user clicked “PTT” button. CONVERSATION_END indicates that there is no further conversation with user and the dialog system will switch to IDLE or LISTENING_WUW state.

- **SESSION_BEGIN and SESSION_END**: A session represents user’s requested task in a specific domain by voice command, there could be multiple sessions in one conversation. SESSION_BEGIN is usually triggered by user’s voice command (e.g., Call John, Take me to New York etc.). SESSION_END indicates that the current session has finished (e.g., user requested task has been completed by application) or user cancels current session explicitly (e.g., user said “cancel”) or implicitly (e.g., user inputs new voice command related to other domain, user does not speak anything until timeout or user’s voice command cannot be recognized by the system for multiple times).
  If application receives SESSION_END, the application might update the GUI to notify the user the dialog session has ended. For example, the application should hide the POI picklist if application receives SESSION_END for navigation domain. Otherwise, user might still speak the picklist command which cannot be recognized by the system that will confuse user.

- **SNOOZING_BEGIN and SNOOZING_END**: If short snoozing feature is enabled, dialog system will continue listening for user’s speech input for a while after there is no active session. In this case, user can input voice command directly without waking up the system firstly. Dialog system will send SNOOZING_BEGIN notification when it enters the snoozing state and will send SNOOZING_END notification before a new session begins or timeout detected for SNOOZING.

  PROMPT_BEGIN and PROMPT_END: PROMPT_BEGIN indicates that the dialog system starts to play prompt, PROMPT_END indicates that the dialog system has finished prompting.

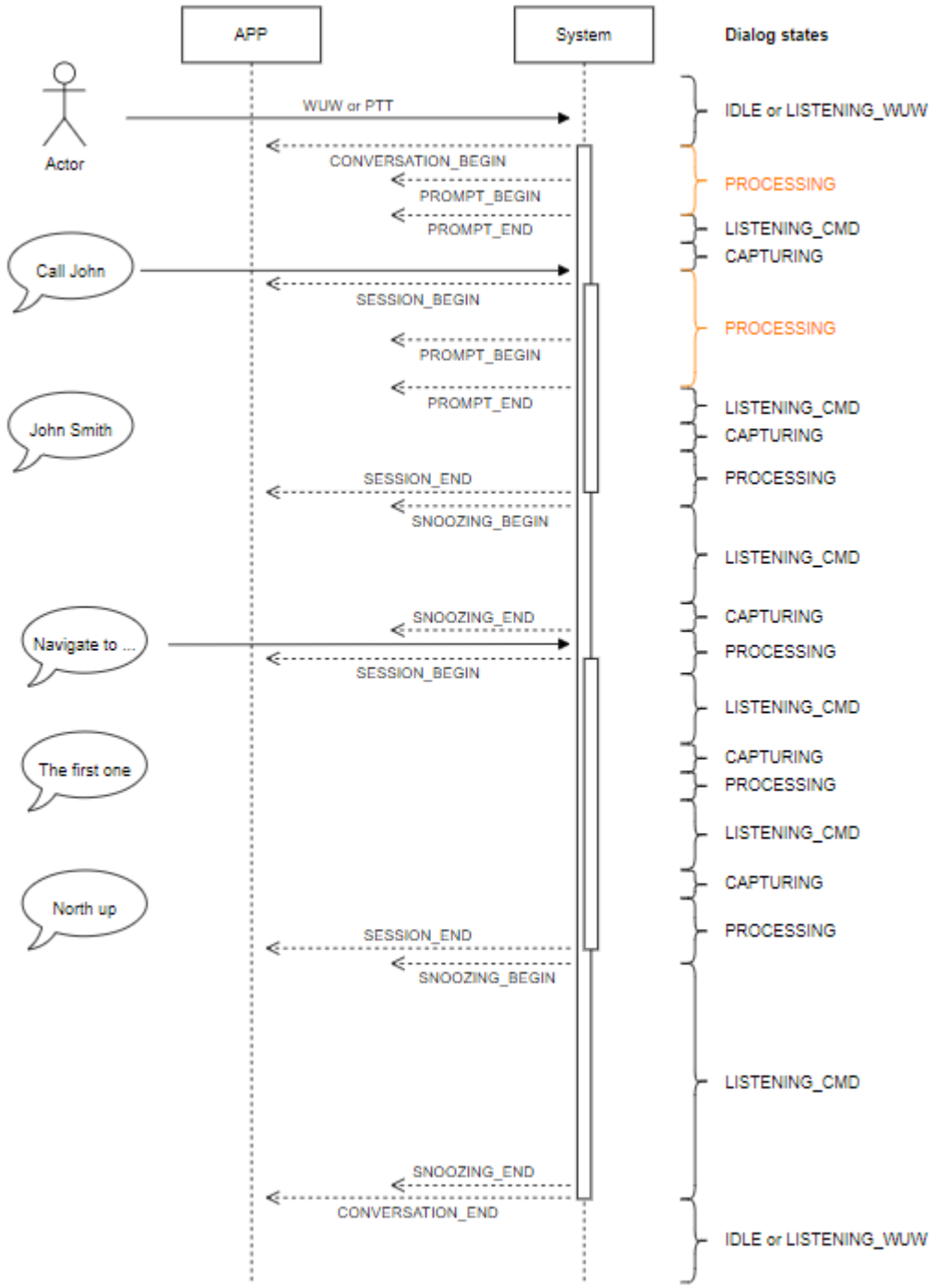
8.2 JSON schema

Dialog event will be sent to application as notification JSON string like following:

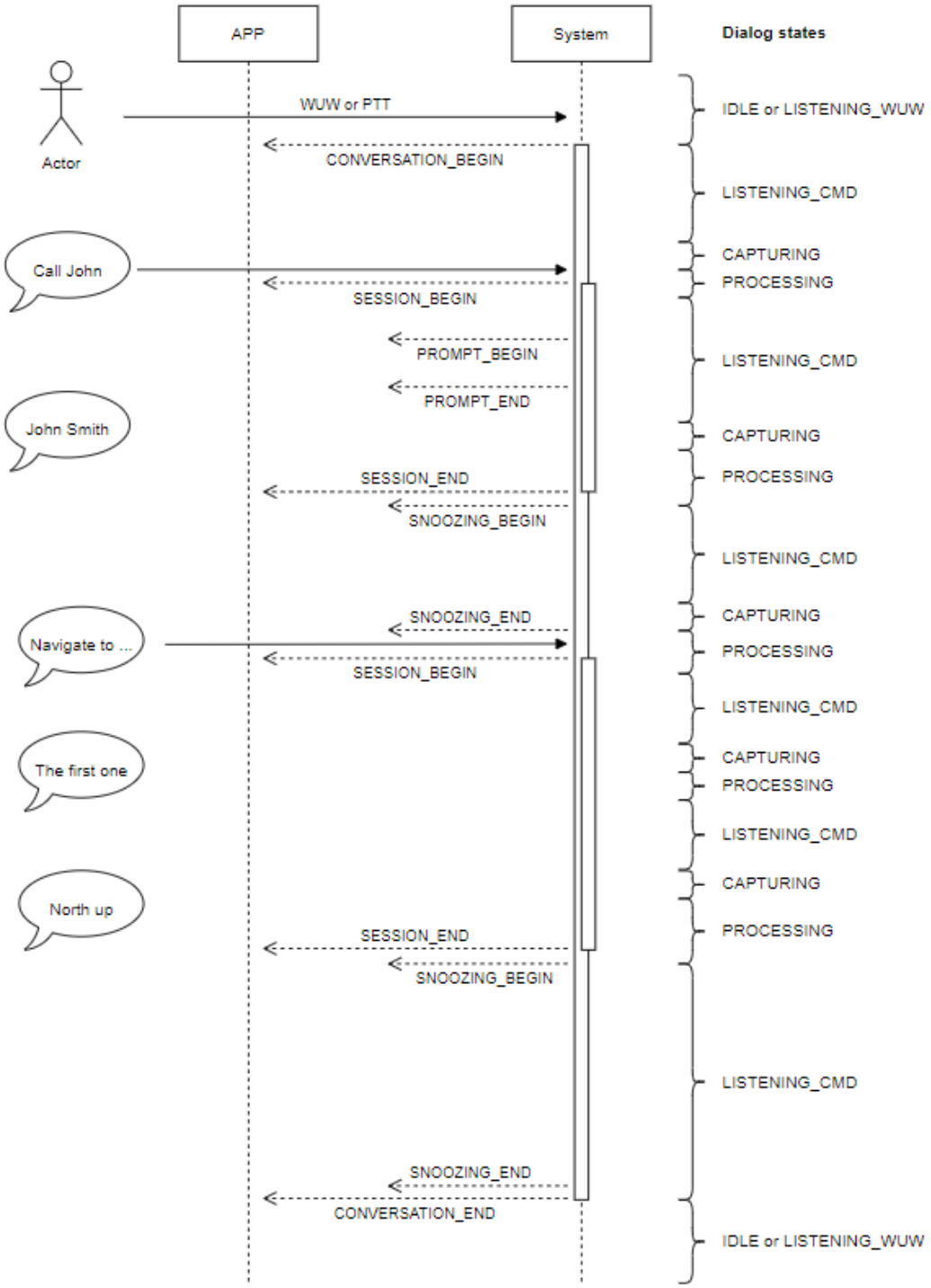
```
{
 "name": "DialogEvent",
 "type": "NOTIFICATION",
 "params": {
  "event": "CONVERSATION_BEGIN",
  "extra": {
    // different event may have different extra information.
  }
 }
}
```

9 FLOWCHART (DIALOG STATE AND DIALOG EVENT)

9.1 Barge-in off



9.2 Barge-in on



10 AUDIORECORDER AND AUDIOPLAYER

Audio capturing and playback is very essential for a dialog system, the SDK has default implementation for audio recorder and audio player. However, application can still implement its own audio recorder and/or audio player to replace the default implementation in the SDK.

10.1 AudioRecorder

AudioRecorder is used for capturing audio data for voice recognition. An AudioRecorder must implement *IAudioRecorder* interface, the default implementation makes use of Android AudioRecord for audio capturing, while capturing audio data by using native OpenSLES/AAudio is also possible.

If application wants to implement its own AudioRecorder, application needs to:

- Implement *AudioRecorderProvider*
- Return its own implementation of *IAudioRecorder* in method *createAudioRecorder*
- Add a file named cerence.ark.assistant.framework.audio.recorder.AudioRecorderProvider in application's resources/META-INF/services folder.
- The file cerence.ark.assistant.framework.audio.recorder.AudioRecorderProvider must contain the implementing class's fully qualified name (e.g. "com.cerence.audio.MyAudioRecorderProvider"). The implementing class must also have a constructor accepting no arguments, used by this package to load the class. Without META-INF file, the SDK will choose the default *AudioRecorderProvider* implementation.

10.2 AudioPlayer

AudioPlayer is used for playing the TTS audio data generated by dialog system through devices' speaker. An AudioPlayer must implement *IAudioPlayer* interface, the default implementation makes use of Android AudioTrack for audio playback, while playing audio data by using native OpenSLES/AAudio is also possible.

If application wants to implement its own AudioPlayer, the application needs to:

- Implement *AudioPlayerProvider*
- Return its own implementation of *IAudioPlayer* in method *createAudioPlayer*
- Add a file named cerence.ark.assistant.framework.audio.player.AudioPlayerProvider in application's resources/META-INF/services folder.
- The file cerence.ark.assistant.framework.audio.player.AudioPlayerProvider must contain the implementing class's fully qualified name (e.g. "com.cerence.audio.My AudioPlayerProvider"). The implementing class must also have a constructor

accepting no arguments, used by this package to load the class. Without META-INF file, the SDK will choose the default *AudioPlayerProvider* implementation.

11 AUDIO FOCUS MANAGEMENT

ARK Assistant SDK follows the convention of Android audio focus management. The SDK has the default implementation for audio focus request and release. Normally, before dialog system interacts with user, the system will request audio focus first, if failed to do so, the dialog system will enter IDLE or LISTENING_WUW state without any prompt. After the system finishes interaction with user, the dialog system will release the audio focus.

Application can override the default implementation if this is not desired behavior.

12 ANDROID PERMISSION

ARK Assistant SDK needs following permissions for working properly:

```
<!--internet-->
<uses-permission android:name="android.permission.INTERNET" />
<!--audio recoder-->
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<!--contact-->
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<!--call log-->
<uses-permission android:name="android.permission.READ_CALL_LOG" />
<uses-permission android:name="android.permission.WRITE_CALL_LOG" />
<!--dial-->
<uses-permission android:name="android.permission.CALL_PHONE" />
<!--location-->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

13 DEPENDENCIES

Application needs to add following dependencies in its gradle build file:

```
dependencies {
    implementation 'androidx.annotation:annotation:1.2.0'
}
```

14 API OVERVIEW

This chapter describes the core APIs of the SDK, for more detailed description, please refer to Java doc.

14.1 ArkAssistant

ArkAssistant provides API for starting/stoping voice assistant. Implementation of these APIs in this class will handle the time-consuming operations in background thread. It is safe to call these APIs in UI thread.

Method	Description
void initializeVoiceAssistant(Context context, Config config, HmiListener hmiListener, IHostController controller, ICallBack callBack)	Initialize the voice assistant, providing the domains' implementation via IHostController, this method will initialize the underlaying core engines (ASR, TTS, etc.).
void startVoiceAssistant (AssistantConfiguration assistantConfiguration, ICallBack callBack)	Start voice assistant, after calling this method, user can interact with the assistant.
void stopVoiceAssistant (ICallBack callBack)	Stop voice assistant, it will abort ongoing recognition and/or prompting, it will also release the audio resources for recording and playback. After calling this method, user cannot interact with the assistant.
AssistantState getAssistantState()	Get current state of assistant.
void sendAppEvent (AppEvent event, ICallBack callBack)	Send the application event which will be handled by dialog system.
boolean setTtsVoiceId(String ttsVoiceId)	Set TTS voice id for dialog system prompting.
static List<TtsVoice> getAvailableTtsVoices(String dataPath, String language)	Gets available TTS voices. The method can be called without initialization of the assistant.
List<String> getAvailablePrompterNames()	Gets the available prompter instance names for creating prompter. This method can only be called after initialization of assistant.
void notifyContentChanged(String contentType)	Notify the dialog system that content on the device has changed. Dialog system will get the latest data via IDynamicContentProvider. Application is responsible for calling this method after successful initialization of voice assistant, if application failed to do so, the dynamic content will not be recognized.
void registerAssistantListener(AssistantListener listener)	Register AssistantListener to get notified when assistant state changed
void unregisterAssistantListener(AssistantListener listener)	Unregister AssistantListener

IPrompter createPrompter(String prompterName, int ttsStreamType)	Create the prompter instance for prompting by application.
IPrompter createPrompter(String prompterName, AudioAttributes attributes)	An overloaded method for creating the prompter instance.
void destroyPrompter(IPrompter prompter)	Destroy the prompter if application does not need it anymore.
void destroyVoiceAssistant (ICallBack callBack)	Destroy the voice assistant, release all resources associated with the assistant SDK.

14.2 Activated

Defines an interface for objects with a simple binary lifecycle- on/off, started/stopped, activated/de-activated

Method	Description
void activate()	Activates the implementation, performing any required initialization.
void deactivate()	Deactivates the implementation, performing any required cleanup.

14.3 Domain

Domain is an abstract class that provides the interfaces for all domain's implementation. Each domain's implementation should derive from Domain.

Method	Description
void loadRequestHandlers(final Map<String, RequestHandler<? extends BaseResult>> map)	Load request handlers into this domain. Implementation of this method should map each request name to corresponding RequestHandler.
Dhi getDhi()	Get Dhi (Domain Hmi Interface) object for sending Hmi request.

14.4 RequestHandler

RequestHandler is an interface that defines the method a request handler should implement.

Method	Description
void onHandle(IRequest request, IResponseSender<T> responseSender)	This method will get called if a request is received. Implementation of this method should handle the request and send the response back to dialog system through IResponseSender interface.
boolean onAbort(IRequest request)	If handling of the previous request timeouts, this method will get called for requesting aborting the previous request. Return true if application can abort the ongoing request, if returns false, that means application is not able to abort the ongoing request, then dialog system will **continue** waiting the response from application. The default implementation will return true.

14.5 IController

IController defines the interface to load implementation of the domains

Method	Description
void loadDomains(final Map<String, Domain> map, final Dhi dhi)	Load domains' implementation into this Controller. Implementation of this method should build the relationship between domain name and its implementation by using a Map.

14.6 IHostController

IHostController is an extension of IController, it defines the interface for getting DynamicContentProvider and it provides the callback which should be implemented by application for handling the unhandled request. Assistant application should provide the implementation of IHostController when initialize the voice assistant.

Method	Description
void loadDomains(final Map<String, Domain> map, final Dhi dhi)	Load domains' implementation into this Controller. Implementation of this method should build the relationship between domain name and its implementation by using a Map.
IDynamicContentProvider loadDynamicContentProvider()	Load the dynamic content provider. Application could provide its own implementation of IDynamicContentProvider through this method.
Response<BaseResult> onUnhandledRequest(IRequest request)	Get called when the system did not find the RequestHandler to handle the request from dialog system. This case will happen if application did not register corresponding RequestHandler by mistake or on purpose. The system requires a default response should be sent back in this case.

14.7 Dhi

Dhi (Domain hmi interface) is the bridge between Hmi and Domain. With Dhi, domain can send the Hmi request to application for GUI rendering.

Method	Description
void sendHmiRequest(IRequest request, IResponseSender<BaseResult> responseSender)	Send the Hmi request to application, the request and its data which is defined by specific domain and it is NOT defined by dialog system.

14.8 IRequest

IRequest provides the interfaces for getting the request name and its parameters.

Method	Description
String getName()	Get the name of the request.
String getParams()	Get the parameters in the request as JSON string.
String getDomainName()	Get the domain name of the request.

14.9 IResponse

IResponse represents the data need to be sent back to dialog system.

Method	Description
void setResultCode(String resultCode)	Set result code of the execution of corresponding request
void setErrorMessage(String errorMessage)	Set the error message for the execution of corresponding request if available.
void setData(T data)	Set the data of the response.

14.10 INotification

INotification represents the dialog system's notification which application might be interested. It's the base class for all dialog notifications, e.g., AsrResultUpdated, DialogEvent, DialogStateChanged etc.

Method	Description
String getName()	Get the name of the notification.
String getParams()	Get the parameters of this notification.

14.11 IResponseSender

IResponseSender defines the interfaces for sending response to dialog system.

Method	Description
void send(IResponse<T> response)	Send the response represented by IResponse object to dialog system.

14.12 HmiListener

HmiListener defines the callbacks for receiving HMI request and notifications from dialog system.

Method	Description
void onRequest(IRequest request, IResponseSender<BaseResult> responseSender)	Get called if there is a Hmi request from dialog system or Domain implementation. The implementation of this callback should show GUI to user according to the request's parameters.
void onNotification(INotification notification)	Get called if there is notification from dialog system. Application could only focus on its interested notifications and ignore other notifications.

14.13 IAudioRecorder

IAudioRecorder defines the interfaces need to be implemented for audio data acquiring.

Method	Description
boolean onInitialize(AudioChannel audioChannel, AudioFormat format, int sampleRateInHz)	Get called to request initializing the audio recorder according to the requirement.
byte[] onGetAudio(int requestedSizeInByte)	Get called to acquire audio data. Application must ensure the data is complete (without data drop) and return the data in real time (without delay). To avoid many GC, application should create one buffer and reuse it for audio data filling, that means the return value is always the same object but with latest audio data for each call.
void onRelease()	Get called when the system does not need any audio anymore (e.g., application called stopVoiceAssistant). Application should release the resources associated with the audio recorder.

14.14 AudioRecorderProvider

Application can provide its own implementation of IAudioRecorder through AudioRecorderProvider.

Method	Description
IAudioRecorder createAudioRecorder()	Application needs to implement this method to return its own implementation of IAudioRecorder.

14.15 IAudioPlayer

IAudioPlayer defines the interfaces need to be implemented for audio playback.

Method	Description
boolean onInitialize(AudioChannel audioChannel, AudioFormat format, int sampleRateInHz, AudioAttributes attributes)	Get called to request initializing the audio player according to the requirement.
void onPlayAudio(byte[] audioData)	Called to request application to play the audio via speaker. After this method returns, the audio data should already be played or written to audio output device.DO NOT buffer the audio data and let the method return immediately, otherwise application will get wrong state notification of the prompt (E.g., the system thought the audio play has finished while the application is still playing the audio).
void onEndOfStream()	Called to notify that there is no more audio chunk for current prompt.
void onStop(boolean isAbort)	Called to request application to stop the playback. If isAbort is true application should stop audio playing immediately, otherwise application should play all received audio data before stopping.
void onRelease()	Get called when the system does not need the audio player anymore (e.g., application called stopVoiceAssistant). Application should release the resources associated with the audio player.

14.16 AudioPlayerProvider

Application can provide its own implementation of IAudioPlayer through AudioPlayerProvider.

Method	Description
IAudioPlayer createAudioPlayer()	Application needs to implement this method to return its own implementation of IAudioPlayer.

14.17 IDynamicContentProvider

IDynamicContentProvider provides a mechanism for getting dynamic data for voice recognition from device.

Method	Description
List<Contact> getContacts(CancellationSignal cancellationSignal)	Get called to get contact information on the device.
List<Music> getMusic(CancellationSignal cancellationSignal)	Get called to get local music information on the device.
List<App> getApps(CancellationSignal cancellationSignal)	Get all apps information installed on the device for voice recognition.
Location getLocation()	Get called to get the location of the device.
List<String> getDynamicContent(String contentType, CancellationSignal cancellationSignal)	Get customized dynamic content for voice recognition. This method will get called after application notifies the SDK that the dynamic content data has changed.

14.18 IPrompter

IPrompter defines the interfaces for generating speech audio data from text and playing the audio data via device's speaker.

Method	Description
void play(String text, boolean isAbort, PrompterCallback prompterCallback)	Generate speech audio data from text and play the audio data via device's speaker.
void pause()	Pause speech audio generation and prompting, this method will result in prompter in paused state.
void resume()	Resume the prompter, this method will let prompter leave the paused state.
void stop()	Stop speech audio generation and prompting.

14.19 PromptPlayerProvider

Application can provide its own implementation of IAudioPlayer through PromptPlayerProvider.

Method	Description
IAudioPlayer createPromptPlayer()	Application needs to implement this method to return its own implementation of IAudioPlayer.

15 DIALOG NOTIFICATION IN DETAILS

This chapter describe the dialog notification in more details (including the json schema and its parameters).

15.1 DialogStateChanged

This notification will be sent to application if dialog state has changed.

JSON Schema	Parameters

{
"name": "DialogStateChanged",
"type": "NOTIFICATION",
"params": {
"current_state": "DialogState_enum"
}
}
	"DialogState_enum":
	[
	"IDLE",
	"LISTENING_WUW",
	"LISTENING_CMD",
	"CAPTURING",
	"PROCESSING"
	]

15.2 AsrResultUpdated

This notification will be sent to application when ASR result is available. If cloud recognizer is enabled, the partial ASR result will be available while user is talking, this is useful for application to implement the word streaming feature.

JSON Schema	Parameters

{
"name": "AsrResultUpdated",
"type": "NOTIFICATION",
"params": {
"utterance": "string",
"is_final": "boolean"
}
}
	"utterance": ASR result string
	"is_final": True if this is the final result, otherwise false.

15.3 DynamicContentUpdated

This notification will be sent to application when dynamic content data has been loaded into ASR engine.

JSON Schema	Parameters

{
"name": "DynamicContentUpdated",
"type": "NOTIFICATION",
"params": {
"dcc_type": "DCCType_enum",
}
}
	"DCCType_enum":
	[
	"CONTACT",
	"MUSIC",
	"APP"
	]

15.4 SpeechLevelUpdated

This notification will be sent to application when speech energy level updated. Application could make use of the speech level value to do GUI animation.

JSON Schema	Parameters

{
"name": "SpeechLevelUpdated",
"type": "NOTIFICATION",
"params": {
"speech_level": "number"
}
}
	"speech_level": The integer value represents the speech energy level, the range is [0,100].

15.5 SpeechInputTimerBegin

This notification will be sent to application when the speech input count down timer begins. Application could make use of this notification to do GUI animation.

JSON Schema	Parameters

{
"name": "SpeechInputTimerBegin",
"type": "NOTIFICATION",
"params": {
"ls_timeout": number
}
}
	"ls_timeout": The leading silence timeout value in second. If dialog system does not detect any speech within the timeout time, a SpeechInputTimeout notification will sent to application.

15.6 SpeechInputTimeout

This notification will be sent to application when leading silence timeout detected.

JSON Schema	Parameters

{
"name": "SpeechInputTimeout",
"type": "NOTIFICATION",
"params": {}
}
	NA

15.7 DialogEvent

Dialog event will be sent to application for notifying what happened in dialog system.

JSON Schema	Parameters

{
"name": "DialogEvent",
"type": "NOTIFICATION",
"params": {
"event": "DialogEvent_enum",
"extra": {}
}
}
	"DialogEvent_enum":
	[
	"CONVERSATION_BEGIN",
	"CONVERSATION_END",
	"SESSION_BEGIN",
	"SESSION_END",
	"PROMPT_BEGIN",
	"PROMPT_END",
	"SNOOZING_BEGIN",
	"SNOOZING_END"
	]
	
	"extra": different dialog event has difference extra information. Please see the table below for more information.

Dialog Event Name	Extra JSON Schema	Parameters

CONVERSATION_BEGIN	{
	"reason": "string",
	"area": "Area_enum",
	"user_id": "string"
	}
		"reason": To indicate why the conversion is triggered, e.g., “WUW”, “PTT”, etc. If the conversation is triggered by an app event, the reason will be set to the event name.
		"Area_enum":
		[
		"DRIVER",
		"PASSENGER",
		]
		
		“user_id”: The id of the user who triggered the conversation.

CONVERSATION_END	NA	NA

SESSION_BEGIN	{
	"domain": "string"
	}
		"domain" The domain for the started session.

SESSION_END	{
	"domain": "string"
	}
		"domain" The domain for the ended session.

PROMPT_BEGIN	{
	"id": "number",
	"text": "string",
	"hints": "string",
	"ls": "string"
	}
		"id": The unique number of the prompt.
		"text": The text of the prompt,
		"hints": The hints of the prompt, whether displaying hints information on the screen depends on application’s implementation. NOT every prompt has hints associated with it.
		"ls": The leading question of the prompt, whether displaying leading question on the screen depends on application’s implementation. NOT every prompt has leading question associated with it.

PROMPT_END	NA	NA
SNOOZING_BEGIN	NA	NA
SNOOZING_END	NA	NA

16 LOGGING

The SDK will log text and audio data at runtime to file system. The log files will be saved in writable path which is passed in by application when initializes the assistant. Here are the folders for the log files:

- ${writable_path}/assistant_log: The SDK text log files will be saved in this folder.
- ${writable_path}/log: The dialog system and underlying core engines' log files will be saved here.
- ${writable_path}/assistant_audio: This folder is used for storing the audio data log for audio recorder and audio player.

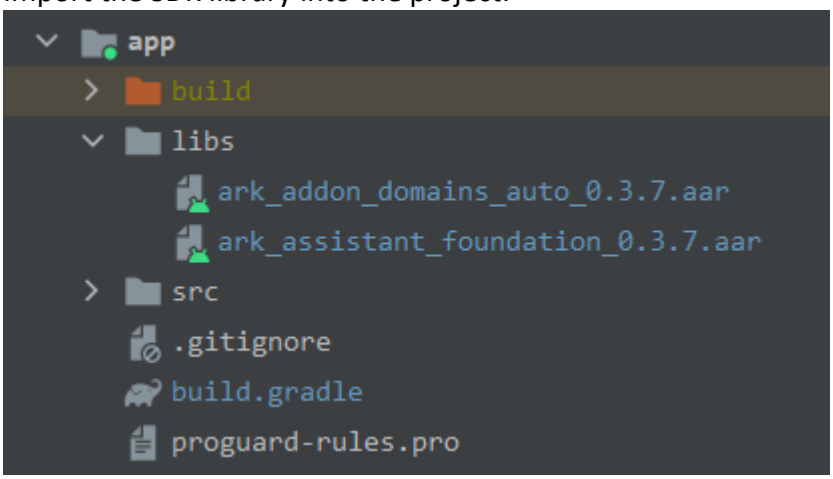
Application can turn on/off these loggings by configuration.

17 INTEGRATE THE SDK

This chapter will guide the developer build up their assistant application step by step in Android Studio.

17.1 Configure the dependency

1. Import the SDK library into the project.



- ark_assistant_foundation.aar: The foundation aar file contains native core libs, Java classes and implementation for foundation features, it defines the core interfaces for ARK assistant, like Domain, IController, RequestHandler etc. Developer can develop the assistant application based on the foundation aar file. If application only uses this aar file, it needs to parse the request which is in raw json format and send the response in raw json format as well. Developer must define the domain classes which extends “Domain” by themselves, load the request handler in the defined Domain.
- ark_addon_domains_auto.aar: This aar file is an addon implementation which defines all the supported domains and translates the Request in Json format into corresponding method call. Some domains have default implementation (e.g., PhoneDomain), some domains only define the abstract methods which needs application to implement. By using this aar file together with ark_assistant_foundation.aar, developer only needs focus on the domains’ implementation, do not need to parse the raw json data. From the naming of this aar file, it indicates that the addon aar is related to automotive dialog implementation. Cerence could provide the addon aar file for other platforms (e.g., two-wheeler, etc.)

2. Add the dependency declaration into Android build.gradle like flowing:

```
repositories {
   flatDir {
      dirs 'libs'
   }
}

dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   // implementation ark assistant sdk
   implementation(name: 'ark_addon_domains_auto_0.3.7', ext: 'aar')
   implementation(name: 'ark_assistant_foundation_0.3.7', ext: 'aar')
   implementation 'androidx.annotation:annotation:1.2.0'
}
```

17.2 Configure Android Manifest

1. Add permission declaration into Manifest

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_CALL_LOG" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />
<uses-permission android:name="android.permission.ACCESS_NOTIFICATION_POLICY" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
```

Please note that not all the permissions are required if application does not implement some features, e.g., if application does not need PhoneDomain, the permission declaration related to PhoneDomain can be removed.

2. Configure SpeechService

   - Configure the SpeechService to be running in the same process with application.

```
<service android:name="cerence.ark.assistant.service.SpeechService" />
```

   - Configure the SpeechService to be running in a separate process.

```
<service
   android:name="cerence.ark.assistant.service.SpeechService"
   android:process=":voice" />
```

17.3 Install the data

Unzip the data.zip to a readable location on the device.

17.4 Implement IHostController

Implement IHostController's method *loadDomains* for loading the desired domains into a Map, the key is the domain name in string, the value is the domain object. Application can also pass pattern as string for the key.

```
public class DemoController implements IHostController {
 @Override
 public void loadDomains(Map<String, Domain> map, Dhi dhi) {
  map.put("Phone", new PhoneDomain(dhi));
  map.put("System", new DemoSystemDomain(dhi));
 }
 @Override
 public IDynamicContentProvider loadDynamicContentProvider() {
  return new DefaultDynamicContentProvider();
 }
 @Override
 public Response<BaseResult> onUnhandledRequest(IRequest request) {
  return new Response<>(ResultCodeEnum.UNSUPPORTED);
 }
}
```

17.5 Implement HmiListener

Implement HmiListener for receiving GUI rendering request and notifications

```
public class DemoHmi implements HmiListener {
  @Override
  public void onRequest(IRequest request, IResponseSender<BaseResult>
responseSender) {
    // Do GUI rendering according to the request.
    responseSender.send(new Response<>(ResultCodeEnum.OK));
  }
  @Override
  public void onNotification(INotification notification) {
    // Handle notification from dialog system.
  }
}
```

17.6 Assistant life cycle

- Initialize the voice assistant

```
// Create the Config object.
Config config = new Config.Builder()
        .language(Language.ENU)
        .dataPath(dataPath)
        .writablePath(writePath)
        .isSaveLog(true)
        .isSaveInputAudio(true)
        .isSaveOutputAudio(false)
        .build();

// Create the Callback object.
ICallBack callBack = new ICallBack() {
      @Override
      public void onSuccess() {
        Log.d(TAG, "Initialize voice assistant success");
      }

      @Override
      public void onFailure(String errorMessage) {
        Log.e(TAG, String.format("Initialize fail, error message = %s",
errorMessage));
      }
};
// Initialize the voice assistant.
ArkAssistant.get().initializeVoiceAssistant(getApplicationContext(), config, new
DemoHmi(), new DemoController(), callBack);
```

- Start voice assistant

```
// Create AssistantConfiguration object.
AssistantConfiguration.Builder builder = new AssistantConfiguration.Builder();
builder.enableBargeIn(false)
      .audioRecordingMode(AudioRecordingMode.SDK)
      .audioPlaybackMode(AudioPlaybackMode.SDK)
      .interactionMode(InteractionMode.WUW)
      .ttsStreamType(AudioManager.STREAM_MUSIC);
// Start voice assistant.
ArkAssistant.get().startVoiceAssistant(builder.build(), null);
```

- Stop voice assistant

```
ArkAssistant.get().stopVoiceAssistant(null);
```

- Destroy voice assistant

```
ArkAssistant.get().destroyVoiceAssistant(null);
```

17.7 Customize domain

- Override the method implementation in an existing domain, given that we have PhoneDomain which has method *makeCall* that uses Android standard API for calling. If the device has a specific API for phone call, that means the standard implementation does not meet the requirement of the application. Application could override the default behavior (e.g., create a CustomPhoneDomain inherited from existing PhoneDomain and load the CustomPhoneDomain into the Controller).

```
public class CustomPhoneDomain extends PhoneDomain {
  public PhoneDomain(Dhi dhi) {
    super(dhi);
  }

  @Override
  public void makeCall(String number, IResponseSender responseSender) {
    //Specific implementation goes here.
  }

  @Override
  public boolean abortMakeCall() {
    return true.
  }
}
```

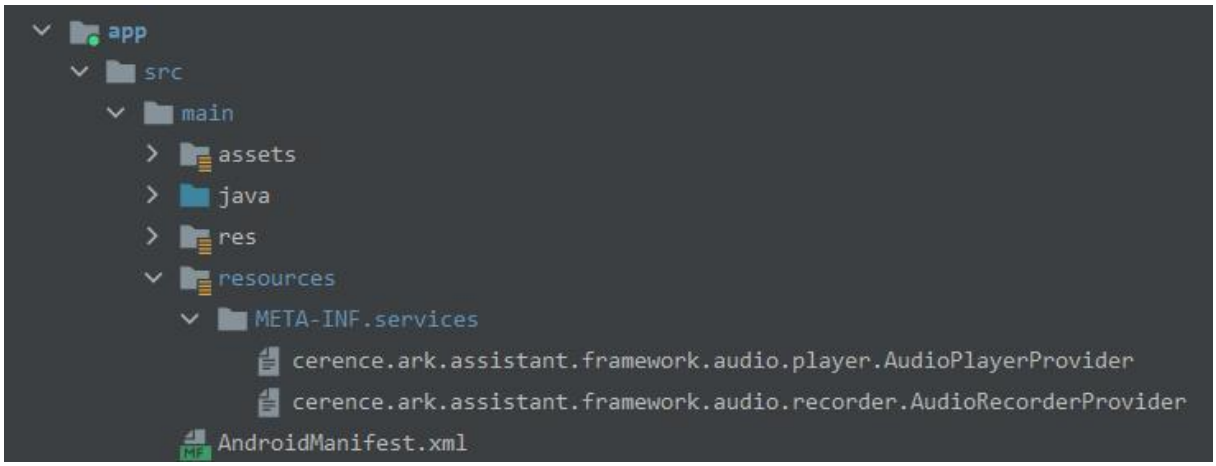
- Override the RequestHandler, it usually happens when dialog interface (Request and Response) has been changed. Developer can either implement a completely new Domain which inherited from Domain class, or implement a custom domain based on the existing domain in the SDK.

```
public class CustomDomain extends Domain {
   public CustomDomain(Dhi dhi) {
      super(dhi);
   }

   @Override
   public void loadRequestHandlers(Map<String, RequestHandler> map) {
      map.put("openMap", new RequestHandler() {
         @Override
         public void onHandle(IRequest request, IResponseSender responseSender)
{
            // Parse the request parameter and call the method
            // for the task completion.
            openMap();
            // send response back to the dialog system.
            responseSender.sendJson(new JSONObject().toString());
         }
      });
   }
}
```

17.8 Customize AudioRecorder/AudioPlayer

The SDK can load the customized audio recorder or audio player via Java Service Provider Interface, developer needs to implement AudioRecorderProvider/AudioPlayerProvider to return the customized implementation of IAudioRecorder/IAudioPlayer, then create the file(s) in folder src/main/resources/META-INF.services and put the fully qualified name of the implementing class into it.



18 TROUBLE SHOOTING

18.1 Initialize voice assistant failed

Check if the resource data has been put into the readable folder on the device and check if the data path has correctly set for the SDK.

18.2 Start voice assistant failed

Check whether the application has got the required permissions.

18.3 Voice recognition does not work

Check whether the audio recording works on the device, and if the application has got the audio record permission. If audio recording works, check the recording data and see if the audio signal is abnormal.

18.4 No ASR partial result

Check if the device has network connection, check whether the system time is current.

18.5 Application becomes non-responsive

If the UI thread is not blocked by any time-consuming operation, please check if application forgot sending the response back to dialog system.

18.6 One-shot command does not work

One-shot feature is only supported when barge-in is on.

19 APPENDIX A: GLOSSARY

Abbreviation	Name
ASR	Automatic Speech Recognition
TTS	Text To Speech
APK	Android Application Package
JAR	Java Archive
NLU	Natural Language Understanding
AIDL	Android Interface Definition Language
AAR	Android Archive
VUI	Voice User Interface
PTT	Push To Talk
HMI	Human Machine Interface
SSE	Speech Signal Enhancement
SPR	Smart Proactive Recommendation

20 APPENDIX B: DOCUMENT HISTORY

DATE	VERSION	AUTHOR	DESCRIPTION
2021-04-20	0.1	Kevin Kang	Initial draft
2021-04-22	0.2	Kevin Kang	Add API Overview
2021-04-23	0.3	Kevin Kang	Add dialog notification in details
2021-04-24	0.4	Kevin Kang	Add flowchart for dialog states and events.
2021-04-28	0.5	Kevin Kang	Add integration guide and trouble shooting.
2021-05-06	0.6	Kevin Kang	Add description for Prompter and PromptPlayer
2021-05-14	0.7	Kevin Kang	Update API description
2021-05-25	0.8	Kevin Kang	Update API description, update required permissions, update dependency.
2021-06-24	0.9	Kevin Kang	Update API description.
2021-07-12	1.0	Kevin Kang	Update the architecture diagram and description.
2021-07-21	1.1	Kevin Kang	Update API description and integration guide.
2021-08-16	1.2	Kevin Kang	Remove PromptPlayer, update API description. Add footprint description.
2021-09-17	1.3	Kevin Kang	Delete deprecated API, update API description, update architecture diagram, update flow chart for barge-in off mode.
2021-10-19	1.4	Kevin Kang	Update API description for IDynamicContentProvider