Agent Interaction SDK Java Developer Guide

Voice Callback

12/14/2025

Contents

Voice Callback

Implementing voice callback is a matter of handling additional callback record information that the Agent Interaction (Java API) provides to interactions.

Callback Design

Scenario

If a customer requests a callback, the Voice Callback server records the request. Interaction SDK (Java) supports Web Callback, that is, the customer can request a callback from a web application.

At the time that the customer requested (As Soon As Possible, or some specific calendar day/time), the Voice Callback server inserts a record of the request into an appropriate queue. From the queue, the Voice Callback request is sent to the place of an available, appropriate agent. When the AIL library receives the request, it creates a `CallbackRecord` for the request, creates a new accompanying `Interaction` (which may be cast as `InteractionVoice`) on the place, and sends appropriate event objects to registered listeners.

As the phone call progresses, the library updates the `Interaction`'s status.
The completion of the phone-call attempt may have various outcomes, including successful interaction with the customer, or busy, or connection to an answering machine or fax machine, and so on.
Upon completion of the attempted call, the application lets the agent signal the AIL library that the `CallbackRecord` is processed, and the `CallbackRecord` status is updated. (When the `Interaction` is closed and marked done, the library also closes the associated `CallbackRecord`.)

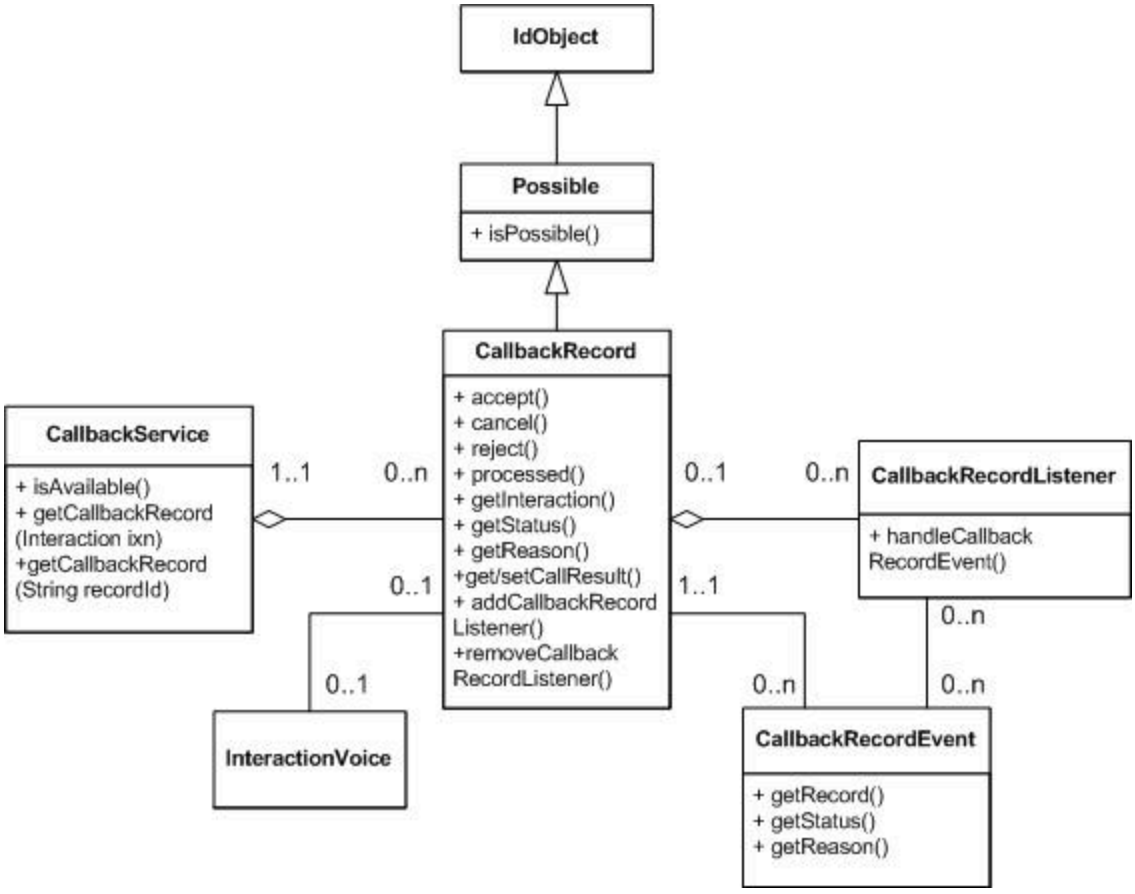
The `CallbackRecord`'s outcome status determines future actions for this Voice Callback request. For example, if the `CallbackRecord` is not successful, the Voice Callback server may re-insert the request in the queue.

If the agent rejects the task, the Voice Callback request remains at the top of the queue to be sent to some other agent's place.

Callback Information

To access callback information, you deal with the `CallbackService` instance of the current `Place` in use. With `CallbackService` methods, you can access to `CallbackRecord` objects for this `Place`.

Each `CallbackRecord` instance enables you to register listeners for getting callback events and contains record data that you use to callback the customer.
To determine which callback record is associated with an interaction, you call the `CallbackService.getCallbackRecord(Interaction)` method.



Interfaces for Callback Features

Callback Campaign Modes

To access callback features, your application gets the `CallbackService` interface associated with the place. Your application receives `InteractionEvent` events for voice interactions in different statuses, depending on the mode of the callback server.

For each voice interaction, use the `CallbackService` interface to get the associated callback record (if any). Then, use the `CallbackRecord` interface to manage the callback activity and to display information.

In preview mode, your application gets the callback record in `PREVIEW` status: the application can accept or reject the callback record by calling the corresponding `CallbackRecord` methods. Then, if the agent accepts the callback, your application makes the call using the `InteractionVoice` object associated with the `CallbackRecord` object.

In predictive mode, the application gets the `CallbackRecord` in `OPEN` status and the `InteractionVoice` object is already in a `DIALING` status. For further details about callback servers' modes, refer to the Voice Callback 7 documentation.

Your application can process the `InteractionVoice` interaction as usual. When the call is released,

assign a call result to the CallbackRecord object, mark it as processed by calling the CallbackRecord.processed() method, then mark the interaction as done.

Steps for Writing a Callback Application

Now that you have been introduced to the callback feature's design, it is time to outline the steps you will need to work with its events and objects.

As specified in the previous section, callback record data does not interfere with interaction management. You should implement a PlaceListener class that manages voice interactions, as explained in previous chapters. Then, modifications in your agent application to handle callback record data consist of a few add-ins.

There are five basic things you will need to do in your AIL applications:

- **Implement a CallbackRecordListener listener** to get notified of changes in active outbound campaigns. Here is how a SimpleExample class would do this:

```
public class SimpleCbRecordListener implements CallbackRecordListener {
        //...
        public void handleCallbackRecordEvent(CallbackRecordEvent event)
        {
                CallbackRecord record = event.getCallbackRecord();
                // update your application with callback information
                // for instance, buttons for callback actions
                //...
        }
}
```

- **Get a callback service** to test whether your PlaceListener should handle callback record on InteractionEvent events, and keep a reference to be able to retrieve callback records. For instance, you could modify one stand-alone code example by declaring a private callbackService variable, then by adding the following code snippet in the constructor method:

```
Class SimpleCallbackExample implements PlaceListener
{
        CallbackService callbackService;

        //...
        public SimpleCallbackExample(Place samplePlace)
        {
                callbackService = samplePlace.getCallbackService();
                if(callbackService.isAvailable())
                {
                        //...
                }
        }
        //...
}
```

- **Set up button actions** (or actions on other GUI components) tied to callback features, according to the CallbackRecord objects' methods, such as accept(), reject(), reschedule(), processed(), and so on.
- **Check if interactions own callback information** in the implemented handleInteractionEvent() methods. To determine whether

```
public handleInteractionEvent(InteractionEvent event)
{
        Interaction sampleInteraction = event.getInteraction();

        if(sampleInteraction.getType() == Interaction.Type CALLBACKREQUEST)
        {
                CallbackRecord callbackRecord =
callbackService.getCallbackRecord(sampleInteraction);
                if(callbackRecord != null)
                {
                        //update your application with callbackRecord info
                        //...
                        //add your listener to get callback record changes
                        callbackRecord.addCallbackRecordListener(new
SimpleCbRecordListener()) ;

                }
        }
}
```

Note that `Interaction` status changes are not automatically coordinated with `CallbackRecord` status. Coordination depends on correct agent behavior.