Agent Interaction SDK Java Developer Guide

Outbound Service

12/14/2025

Contents

- 1 Outbound Service
  - 1.1 Outbound Design
  - 1.2 Steps for Writing an Outbound Application
  - 1.3 Preview Outbound Interactions
  - 1.4 Predictive Outbound Interactions
  - 1.5 Handle Outbound Chains

Outbound Service

Handling outbound information no longer requires handling a particular interaction type. Implementing outbound is now a matter of handling additional outbound information that the Agent Interaction (Java API) provides to interactions.
This chapter shows you how to deal with the outbound service.

Outbound Design

In the 7.6 release, you no longer deal with the `InteractionVoiceOutbound` interface. Instead, you manage interactions as usual, and you get additional outbound information for an interaction to be processed. These changes make it possible to handle both voice and multimedia interactions in outbound campaigns.

Outbound Information

To access outbound information, you deal with the `OutboundService` instance of the current `Place` in use. With `OutboundService` methods, you can register listeners for getting events about current campaigns, and access to `OutboundChain` objects for this `Place` .
Each `OutboundChain` instance contains customer outbound data, provided as a collection of `OutboundRecord` objects. For example, in the context of a voice outbound call, each record of the chain contains a phone number associated with the customer to be called. If the call with a given record fails, the agent can get a chained record to attempt a new call for this customer.
Regardless the campaign mode, the `Place` object receives `PlaceEventOutboundChainInfo` events for new or modified outbound chains that the agent should process.
To determine which outbound chain is associated with an interaction, call the `OutboundService.getOutboundChain(Interaction)` method.

Outbound Actions

`OutboundRecord` and `OutboundChain` interfaces provide you with outbound methods, that the agent calls to perform outbound actions, such as `cancel`, `do not call` , and `reschedule` .
These actions are independent from the outbound interaction used to process the outbound record or the outbound chain. They manage record information on the Outbound Server.

Campaign Dialing Modes

The campaign dialing modes determine how an agent, or a group of agents, participates in an outbound campaign. This affects the outbound event flow, and also agent actions to be performed when participating in the campaign.
The following table introduces these dialing modes.

Campaign Dialing Modes

Campaign Mode	AIL Events	Description
PREVIEW	`PlaceEventOutboundChainInfo`	In preview dialing mode, an agent requests one or several records from the OCS, previews the record(s), and decides to process one of them by creating a new outbound interaction.
PUSH_PREVIEW	`PlaceEventOutboundChainInfo` `InteractionEvent` (for `NEW` `InteractionOpenMedia`)	(Also called proactive) In push preview mode, the agent does not need to request the record to preview the record. He or she gets this and subsequent records through a new open media interaction. To process the record, the agent creates a new outbound interaction.
PROGRESSIVE	`PlaceEventOutboundChainInfo` `InteractionEvent` (for `DIALING` `InteractionVoice`)	The OCS dials a record in the list as soon as an agent is available.
ENGAGED_ PROGRESSIVE	`PlaceEventOutboundChainInfo` `InteractionEvent` (for `DIALING` `InteractionVoice`)	The OCS creates a voice interaction to dial a record in the list when an agent is available and engaged. Your agent application gets a dialing outbound voice interaction and an outbound chain.
PREDICTIVE	`PlaceEventOutboundChainInfo` `InteractionEvent` (for `DIALING` `InteractionVoice`)	The OCS predicts agent availability and dials a record. Your agent application gets a dialing outbound voice interaction and an outbound chain.
ENGAGED_ PREDICTIVE	`PlaceEventOutboundChainInfo` `InteractionEvent` (for `DIALING` `InteractionVoice`)	The OCS predicts when the agent is available and engaged, and creates a dialing outbound voice interaction to dial a record in the list. Your agent application gets a dialing outbound voice interaction and an outbound chain.
UNKNOWN	Unknown	Unknown campaign mode.

> **Important**
>
> According to the campaign mode, you may notice the following:

- In a non-engaged mode, the contact may be online before the agent.
- In an engaged mode, your application can receive a record after the agent accepted a ringing call. Test the `InteractionVoice.isASMCall()` to check whether a ringing call is part of an outbound campaign.

For further information, refer to the *Outbound Contact 7.6 Documentation*.

Steps for Writing an Outbound Application

Now that you have been introduced to the outbound 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, outbound data do not interfere with interaction management. To handle campaign information and outbound records, modifications in your agent application consist in a few adds-in.
There are five basic things you will need to do in your AIL application:

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

```
public class SimpleExample implements CampaignListener {
        //...
        public void handleCampaignEvent(CampaignEvent event)
        {

                // Testing whether it is a new active campaign
                if(event.getEventType()==CampaignEvent.Type.CAMPAIGN_ADDED)
                {
                        //The agent takes part in a new outbound campaign
                        //...
                }
        }
}
```

- **Get an outbound service** to register your campaign listener. For instance, you could modify one standalone code example by declaring a private `outboundService` variable, then by adding the following code snippet in the constructor method:

```
Class SimpleExample implements CampaignListener
{
        OutboundService outboundService;
        public SimpleExample()
        {
                //Retrieve the service
                outboundService = samplePlace.getOutboundService();
                //Add your campaign listener
                outboundService.addListener(this) ;
                //...
        }
```

```
}
```

- **Set up button actions** (or actions on other GUI components) tied to outbound features, according to the `OutboundService`, `OutboundChain`, and `OutboundRecord` objects' methods. For instance, to cancel a record, your agent needs an `Cancel` button to cancel the processing of the record.

```
// Add a cancel button for joining the chat session
JButton cancelOutboundRecordButton = new JButton(“Cancel”);
cancelOutboundRecordButton.setAction(new AbstractAction("Cancel") {
        public void actionPerformed(ActionEvent actionEvent) {
                try {
                        outboundRecord.cancel(null);
                } catch (Exception exception) {
System.out.println(exception.getMessage(),"ErrorEvent");
                }
        }
});
```

- **Modify the `PlaceListener.handlePlaceEvent()` method** to handle `PlaceEventOutboundChainInfo` events. Create a thread that manages `PlaceEventOutboundChainInfo` events to update your application with outbound information.

```
// For instance, display the event reason</tt>
System.out.println("Outbound chain event- reason is:
"+PlaceEventOutboundChainInfo.getReason().toString());
```

- **Check if interactions own outbound information** in the implemented `handleInteractionEvent()` methods. If a campaign is started, according to the campaign mode, your application may have to check if interactions in `NEW`, `DIALING`, and `RINGING` status are associated with outbound information.

See further sections for details.

Preview Outbound Interactions

In a Preview Outbound Campaign, the agent is active and requests new outbound records, then chooses either to process or not process the call.

Active Campaigns

As soon as the agent is logged in, his or her place receives events for active campaigns. To determine whether your agent participates in a Preview Outbound Campaign, you can implement the `CampaignListener` as described in the previous section. On `CampaignEvent` events of type CAMPAIGN_ADDED, retrieve the corresponding CampaignInfo object and test its type, as follows:

```
if(event.getEventType()==CampaignEvent.Type.CAMPAIGN_ADDED)
{
        OutboundCampaignInfo campInfo = event.getCampaignInfo();
        if(campInfo.getCampaignMode() == OutboundCampaignInfo.Mode.PREVIEW)
        {
                //...
        }
}
```

Another way to determine whether a preview outbound campaign is active is to retrieve outbound campaigns available for your place, by calling the `OutboundService.getCampaignInfos()` method, or, if you know the campaign ID, by calling the `OutboundService.getCampaignInfo()` method.

Start and Stop Preview

Your application should start and stop preview mode according to the optional setting `agent_preview_mode_start` defined in the OCS application object in Configuration Manager. If this option is set to `true`, your agent application must start preview mode method after an agent logs in, prior to any preview record request, in order to receive scheduled call records from OCS.

This setting is most often used to ensure that no prescheduled call records are sent to the place directly after the agent logs in.

> **Important**
>
> The Agent Interaction (Java API) does not include a method to test this option. This is your responsibility to determine whether you develop an agent application working with an OCS whose `agent_preview_mode_start` option is `true`.

To start preview mode, retrieve the `OutboundCampaignInfo` instance that corresponds to your campaign. Then, call the `startPreviewMode()` method.

```
OutboundCampaignInfo campaign = OutboundService.getCampaignInfo(campaignID);

campaign.startPreviewMode();
```

To stop preview mode (when the agent stops working in outbound campaigns), call the `stopPreviewMode()` method.

```
campaign.stopPreviewMode();
```

If the agent wants to participate in a preview campaign, preview mode must be started before requesting any preview record, else OCS ignores calls to get preview records.
When the option `agent_preview_mode_start` is set to `false` , OCS assumes that the agent is ready to receive any prescheduled call records. If a preview campaign is running when the agent logs in, he or she can request preview records at anytime.

Handle Regular Preview Calls

Your agent is now ready to participate in an active campaign. If preview mode mode is active (see above for details), the first step is to request a preview record.

> **Important**
> If your campaign mode is push preview, your application does not need to request the preview record.

To get a preview record, you can choose between calling the`getPreviewRecord()` method (which returns the record) or calling the `requestPreviewRecord()` method of your `OutboundCampaignInfo` interface.
If your application calls the `requestPreviewRecord()` method, and if a record is available, you get a `PlaceEventOutboundChainInfo` event through the `PlaceListener.handlePlaceEvent()` method.

Use the `PlaceEventOutboundChainInfo` event to inform the user that a new outbound record should be processed, as shown here:

```
public void SimplePreviewExample implements PlaceListener {
        OutboundService outboundService ;

        public void SimplePreviewExample()
        {
                //...
                boolean request = campInfo.requestPreviewRecord();
                if(request == true)
                        System.out.println("Request for preview record succeeded.");
        }

        public void handlePlaceEvent(PlaceEvent event)
        {
                if(event instanceof PlaceEventOutboundChainInfo)
                {
                        PlaceEventOutboundChainInfo eventInfo = (PlaceEventOutboundChainInfo)
event ;
                        OutboundChain outboundChain = eventInfo.getOutboundChain();
                        OutboundRecord outboundRecord = outboundChain
                        .getActiveRecord();
                        System.out.println("Outbound chain event, reason "+
eventInfo.getReason().toString());
                }
        }
```

To process the `OutboundChain`, create an interaction which uses the active record to fill in the `Interaction` data and methods' parameters. In the following code snippet, the interaction processes a voice call and uses record data to dial the call.

```
        // Method called when the agent wish to use the record
        public void processActiveRecord(OutboundChain outboundChainToProcess)
        {
                InteractionVoice outboundIxn =
                        (InteractionVoice) outboundChain.createInteraction
(MediaType.VOICE,null,agentInteractionData.getQueue());
                outboundIxn.makeCall(outboundRecord.getPhone(), null,
InteractionVoice.MakeCallType.REGULAR, null, null, null) ;
        }
}
```

> **Important**
> When your outbound interaction is released, close the outbound chain, as specified in section Close the Chain.

Handle Push Preview Interactions

In push preview mode, your application deals with open media interactions: it does not make preview requests, it receives the open media interaction and the chain which contains the record.
Attached data provided in the open media interaction contains additional preview information that your application uses to create the outbound interaction (for instance, voice or e-mail).
Push preview mode requires that the agent logs into a third party media (refer to the Configuration Layer for further details). Then, the setup is the same: you get an `OutboundService` from your `Agent`, and you register your listeners.

Push Preview Events

When the agent is logged into and ready on the third party media, you get the following events:

- a `PlaceEventOutboundChainInfo` event through the `PlaceListener.handlePlaceEvent()` method.
- an `InteractionEvent` event through the `PlaceListener.handleInteractionEvent()` method for an `InteractionOpenMedia` in `NEW` status, as shown in the following code snippet.

```
public void handleInteractionEvent(InteractionEvent event)
{
        if(event.getSource() instanceof InteractionOpenMedia
                        && event.getStatus() == Interaction.Status.NEW))
                        {
                                InteractionOpenMedia outboundPreviewIxn =
(InteractionOpenMedia) event.getSource();
                                OutboundChain outboundChain =
outboundService.getOutboundChain(ixn);
                                OutboundRecord outboundRecord =
outboundChain.getActiveRecord();
                                // Use the outbound chain to create the outbound interaction
                                // and use the InteractionOpenMedia to get additional data
                                //...
                        }
}
```

Process the Outbound Interaction

To process the `OutboundChain` , create an interaction by calling the OutboundChain.createInteraction() method (for instance, voice or e-mail) which uses the active record and the open media interaction to fill in the `Interaction` data and methods' parameters. In the following code snippet, the interaction processes a voice call and uses record data to dial the call.

```
// Method called when the agent wish to use the record
public void processActiveRecord(OutboundChain outboundChain, InteractionOpenMedia
```

```
outboundPreviewIxn)
{
        InteractionVoice outboundIxn =
                (InteractionVoice) outboundChain.createInteraction
(MediaType.VOICE,null,(String) outboundPreviewIxn.getAttachedData("queue");
                ixnVoice.makeCall(recordToProcess.getPhone(), null,
InteractionVoice.MakeCallType.REGULAR, null, null, null) ;
}
}
}
```

To end the outbound interaction, the agent releases it then marks it done. Finally, before your application closes the outbound chain (as explained in section Close the Chain), your application should also mark done the preview open media interaction.

```
outboundIxn.releaseCall();
outboundIxn.markDone();

outboundChain.markProcessed();
outboundPreviewIxn.markDone();
outboundChain.close();
```

> **Important**
>
> The agent can be in charge of performing the mark done of the open media interaction.

Predictive Outbound Interactions

Handling a predictive outbound is simpler than handling preview outbound interactions. The setup is the same: you get an `OutboundService` from your `Agent`, and you register yours listeners, but you do not have to request records. Outbound Server is in charge of distributing records and dialing interactions. Refer to the *Outbound Documentation* for further details.

For a predictive outbound campaign, your application just waits for `RINGING` interactions and outbound chains.

Active Campaigns

To determine whether your agent participates in a Predictive Outbound Campaign, test whether the campaign mode is OutboundCampaignInfo.Mode.PREDICTIVE. To do so, you can implement the `CampaignListener` as described in section Active Campaigns.

Handling a Predictive Outbound Interaction

If your agent participates in a predictive campaign, your application gets interactions in `DIALING` or `TALKING` status, to be processed as usual. For further details, see previous interaction chapters. When your application gets those interactions through interaction events, display outbound data. Each received outbound interaction is associated with an outbound chain, that contains the record

used to fill in interaction data. You get this record by calling the `OutboundChain.getActiveRecord()` method, as shown in the following code snippet:

```
public void handleInteractionEvent(InteractionEvent event)
{
        if(event.getStatus() == Interaction.Status.DIALING
        )
        {
                Interaction outboundIxn = event.getInteraction();
                OutboundChain outboundChain = outboundService.getOutboundChain(ixn);
                OutboundRecord outboundRecord = outboundChain.getActiveRecord();
                //...
        }
}
```

When the agent has processed the outbound record and released the outbound interaction, he or she must specify the processing result by calling the `OutboundRecord.setCallResult()` method. The `OutboundRecord.CallResult` enumeration lists the possible result your application can provide the agent with.

```
// Successfully processed the interaction
outboundRecord.setCallResult(OutboundRecord.CallResult.ANSWER);
```

If the interaction is processed, you mark the corresponding outbound chain as processed, else for instance, you can reschedule the record.
In any case, after you mark done the interaction, call the `OutboundChain.close()` method to terminate properly the outbound processing of the chain.

```
outboundChain.markProcessed();
outboundIxn.markDone();
outboundChain.close();
```

Handle Outbound Chains

This section covers additional information about outbound chains, regardless campaign modes.

Mark Processed

After the agent releases the outbound interaction, your application can set a call result to the active record and mark the chain processed, as shown in the following code snippet:

```
outboundRecord.setCallResult( OutboundRecord.CallResult.FromString(result));
outboundChain.markProcessed();
```

Reschedule the Record

After releasing the call, the agent can request a callback. In this case, your application needs to create a `Calendar` object to set up the callback at the required date, then reschedule the record.

```
java.util.Calendar myCalendar = java.util.Calendar.getInstance();
myCalendar.setTimeZone(java.util.TimeZone.getTimeZone("GMT-8:00"));

// Reschedule the call: 12/01/2008 at 9:00
myCalendar.set(2008, 01, 12, 9, 0);

// Anybody logged in the campaign is authorized to make the call
selectedRecord.reschedule(myCalendar, OutboundRecord.CallbackType.CAMPAIGN);
```

> **Important**
>
> The time zone of the `Calendar` instance must be set to GMT.

Next step is to close the outbound chain. See Close the Chain.
As a result of the `reschedule` command, a new chain is sent at the required date and time. To determine whether the active record of the outbound chain is a callback, call the `OutboundChain.isScheduled()` method (which returns `true` if the active record is a callback).

Close the Chain

In any case, after your application released and marked done the outbound interaction, call the `OutboundChain.close()` method to terminate properly the outbound processing of the chain, as shown in the following code snippet.

```
ixn.releaseCall(null);
outboundChain.markProcessed();
ixn.markDone();
outboundChain.close();
```