Agent Interaction SDK Java Developer Guide

Chat Interactions

12/13/2025

Contents

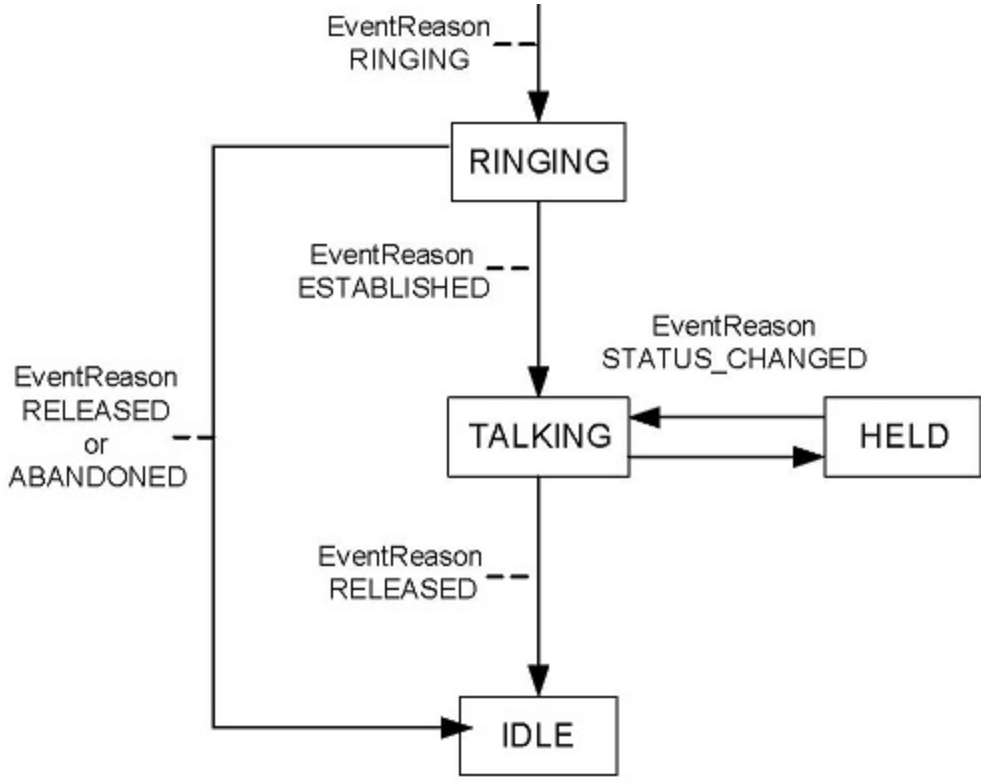
Chat Interactions

Chat interactions are a type of multimedia interactions, that make use of the `InteractionChat` interface, and inherit functionality from `InteractionMultimedia` interface. Other multimedia interactions are discussed in E-Mail Interactions and Open Media Interactions.
This chapter discusses chat interactions. It also presents `SimpleChatInteraction`, a new example that allows user to join chat sessions, send messages, and use the CoBrowse feature.

Chat Interaction Design

Chat State

Because `InteractionChat` inherits the `InteractionMultimedia` interface, each `InteractionChat` object has a status, described in the `Interaction.Status` inner class. The following diagram shows the possible states of a chat interaction. The only chat interactions are incoming.



Chat State Diagram

For a chat interaction, the interaction status can change due to a common`Interaction.Action`, that is, a call to the corresponding method. For example, a successful `Interaction.Action.ANSWER_CALL` action changes the interaction status from `Interaction.Status.RINGING` to `Interaction.Status.TALKING`. The corresponding method is `InteractionChat.answerCall()`.

Not all `Interaction.Action` actions are available on chat interactions. `InteractionChat` inherits `Possible`. Test if an action is possible on a chat interaction by calling`InteractionChat.isPossible(Interaction.Action)`.

If the chat interaction has a `TALKING` status, the chat interaction is active in the chat session and you can take chat-specific actions on the interaction, by using methods such as:

- `sendMessage()`
- `conferencePlace(), conferenceAgent()`
- `clearCall()`

Details about these and other methods are provided in the Handling a Chat Interaction section.

CoBrowse Interactions

The Agent Interaction (Java API) provides a CoBrowse feature through the `InteractionCoBrowse`

interface used as a container to store URLs that your application can share with a customer at runtime. The stored URLs are then intended to be used in contact histories. For further details about the History feature, see Contact History.
The CoBrowse feature does not include any web management and is fairly simple to use. You create an `InteractionCoBrowse` instance by calling a standard `createInteraction()` method; most of the time, you will need a CoBrowse interaction when you are handling another interaction (of any type, that is e-mail, chat, voice, or open media.) To link your `InteractionCoBrowse` instance to an existing interaction you specific a parent interaction in parameters at its creation.

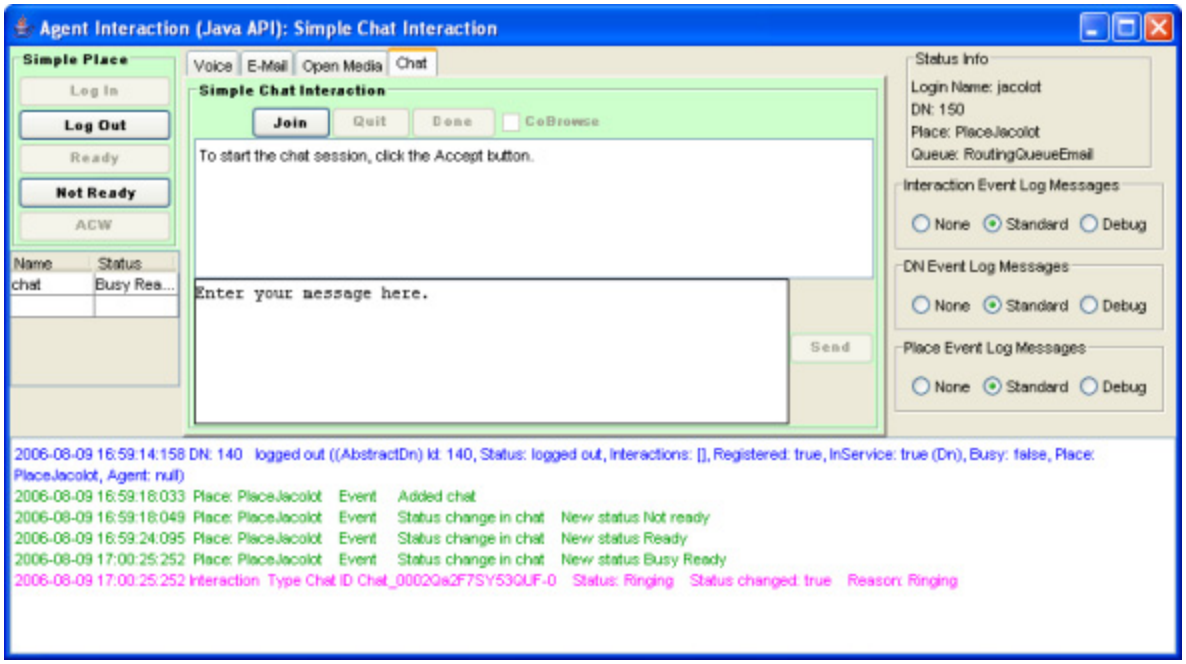
As this interaction is a used as a container, this `InteractionCoBrowse` instance has no status to monitor by handling events. Then, all your application is responsible is saving the `InteractionCoBrowse` instance.

The `SimpleChatInteraction` example includes a CoBrowse feature. For further details, refer to Add CoBrowse-Handling Code.

SimpleChatInteraction

This example is similar to `SimpleVoiceInteraction`, which was introduced earlier. It uses the same graphical user interface and the same internal structure, inheriting from `SimplePlace`. When you launch this example, which is in the `StandAloneExamples` directory, you will see the user interface presented in the screen shot below.

> ### Important
> For the sake of simplicity, this example is designed to handle one chat session at a time. Set up a capacity rule limiting the agent to a single chat interaction at a time in your routing strategy. For further details, see *Universal Routing 7.6 Documentation*.



Join a Chat Session

If the user checks the `CoBrowse` checkbox, `SimpleChatInteraction` opens a dialog box which displays the content of the child CoBrowse interaction created for the example. Then, `SampleChatInteraction` parses all the received chat messages, adds any URL found in the chat text to the `CoBrowse` interaction, and finally saves this interaction when the chat interaction terminates.

Set up Button Actions

By inheritance, `SimplePlace` takes care of all agent buttons located in the `Simple Place` panel. `SimpleChatInteraction` is in charge of chat buttons designed to manage the chat session, that is, `Join` a ringing chat session, `Quit` an established session, `Done` to mark the corresponding chat interaction as `Done` . The Send button enables the agent to send a chat message entered in the message panel.
The GUI program `AgentInteractionGui` has created buttons for each of these actions, but at this point they do nothing. `SimpleChatInteraction` rings these buttons to life by overloading the `linkWidgetsToGui()` method.
The following code snippet shows how to implement the Send button. The corresponding button action uses the contents of the chat message text area to send a message, then, it clears this text area.

```
// Add a send button for the chat session
sendButton = agentInteractionGui.sendChatMsgButton;
sendButton.setAction(new AbstractAction("Send") {

public void actionPerformed(ActionEvent actionEvent) {
 try {
   String msg = chatMsgTextArea.getText();
   sampleChatIxn.sendMessage(msg);
```

```
   chatMsgTextArea.setText("");
 } catch (Exception exception) {
   agentInteractionGui.writeLogMessage(exception.getMessage(),"ErrorEvent");
 }
}});
```

Add Event-Handling Code

`SimpleChatInteraction` is designed to handle chat interactions. This means there needs to be interaction-related, event-handling code.
As explained in the Threading section in About Agent Interaction (Java API), the standalone examples use threads to avoid delaying the propagation of events.
In this purpose, the `SimpleChatInteraction` uses two threads:

- `ChatSessionInteractionEventThread` to handle `InteractionEvent` events sent for the `InteractionChat` used to process the chat session.
- `InteractionChatEventThread` to handle `ChatEvent` events sent for processing messages during the chat session.

The event-handling code in `ChatSessionInteractionEventThread` is similar to the event-handling code in `VoiceInteractionEventThread`. It checks several statements to handle status changes in the processed `InteractionChat` instance (that corresponds to the chat session.) For further details, see Add Event-Handling Code.

The event-handling code in `InteractionChatEventThread` is in charge of displaying chat messages and information about users in the chat panel.

```
if(chatEvent.getEventType() == InteractionChatEvent.Type.MESSAGE_RECEIVED)
{
        displayInteractionChatMessage(chatEvent.getParty(), chatEvent.getText());
        if(sampleCoBrowseIxn != null)
        {
                checkURLs(chatEvent.getText());
        }
}
else if(chatEvent.getEventType() == InteractionChatEvent.Type.DISCONNECTED)
{
        agentInteractionGui.writeChatMessage(" ", "You are disconnected !",
AgentInteractionGui.ELSE_STYLE);
}
else if(chatEvent.getEventType() == InteractionChatEvent.Type.USER_JOINED)
{
        agentInteractionGui.writeChatMessage( chatEvent.getDate().toGMTString(),
chatEvent.getParty().getNickName() + " joined", AgentInteractionGui.ELSE_STYLE);
}
else if(chatEvent.getEventType() == InteractionChatEvent.Type.USER_LEFT)
{
        agentInteractionGui.writeChatMessage( chatEvent.getDate().toGMTString(),
chatEvent.getParty().getNickName() + " left", AgentInteractionGui.ELSE_STYLE);

}
else if(chatEvent.getEventType() == InteractionChatEvent.Type.USER_REENTER)
{
        agentInteractionGui.writeChatMessage( chatEvent.getDate().toGMTString(),
chatEvent.getParty().getNickName() + " reentered", AgentInteractionGui.ELSE_STYLE);
}
```

Chat events do not affect the status of the chat interaction. That's why interaction widgets don't update on chat events.

Add CoBrowse-Handling Code

In this code example, CoBrowse is started when the user checks the CoBrowse checkbox. At that moment, `SimpleChatInteraction` calls its `startCoBrowse()` method which creates an `InteractionCoBrowse` instance for the current chat session, as shown here:

```
try {
//1. Create a new InteractionCoBrowse
  sampleCoBrowseIxn = (InteractionCoBrowse) sampleAgent.createInteraction(MediaType.COBROWSE,
sampleChatIxn, sampleChatIxn.getQueue());
} catch (RequestFailedException e) {
   agentInteractionGui.writeLogMessage(e.getMessage(), "ErrorEvent on CoBrowse creation");
}
```

Then, when the application gets a chat message, it parses the message content by calling the `checkURLs()` method. If this method finds a URL in the text, it adds the URL to the `InteractionCoBrowse` instance.

```
test[i] = "http:"+test[i];
agentInteractionGui.coBmodel.addElement(test[i]);

try {
  sampleCoBrowseIxn.addURLs(new String[]{test[i]});
} catch (RequestFailedException e) {
  agentInteractionGui.writeLogMessage(e.getMessage(),"ErrorEvent");
}
```

When the chat interaction is terminated, that is, in `IDLE` status, the associated CoBrowse interaction is saved and marked as done.

```
sampleCoBrowseIxn.save();
sampleCoBrowseIxn.markDone();
```

Handling a Chat Interaction

Chat interactions are multimedia interactions that allows an agent to manage, or participate in, a chat session. You need a single chat interaction to let your agent take part in the chat session. The chat interaction receives events for message exchanges.
Chat interactions are available in the `InteractionChat` interface of the `com.genesyslab.ail` package. The following sections detail how to use this interface.

Entering a Chat Session

You can participate in a chat session if your `Agent` object has successfully logged into a `CHAT` media in its place. Through a registered `AgentListener` of your `Agent` object, you are notified of a chat session

request by receiving an `InteractionEvent` event about an `InteractionChat` object, in`Interaction.Status.RINGING`.

```
void handleInteractionEvent( InteractionEvent event ) {
  // ... Check if it is the awaited interaction
  InteractionChat myChatInteraction = (InteractionChat) event.getSource();
  // ...
}
```

To take part in the chat session, invoke the `answerCall()` method. If the action is successful, your application receives an `InteractionEvent` event, showing that the interaction is now in state `TALKING`. See Processing a Chat Interaction.

The agent is now one party to the chat session and the chat interaction is active in the chat session.

> **Important**
>
> Assign a nickname to the agent with the `InteractionChat.setNickName()` method before answering the interaction. If you do not assign a nickname, the nickname is the agent's user name.

Chat Parties

The parties of the chat session are available through the `InteractionChat.getParties()` method. A `ChatParty` object describes the nickname and visibility of each party.

Handling Chat Events

To handle discussion, chat interactions send text messages and receive `InteractionChatEvent` event with a registered `InteractionChatListener`. These events also propagate chat errors and party changes during the chat session. The `InteractionChatEvent`.Type inner class lists the possible `InteractionChatEvent` types.

The following code snippet implements an `InteractionChatListener` class:

```
class ExampleChatListener implements InteractionChatListener {
  public void handleInteractionChatEvent (InteractionChatEvent chatEvent)
  {
      /// Management of the chat event
  }
}
```

To receive `InteractionChatEvent` events, register your `InteractionChatListener` on your `InteractionChat`. When registering, you can get all the events exchanged during the chat session before you are connected, as shown in the following code snippet:

```
InteractionChatEvent[] allPreviousEvents = myChatInteraction.addChatListener(new
ExampleChatListener(), true); // previous events are returned.
```

> **Important**
> You can also get all these events after registration, by calling the `InteractionChat.getEvents()` method.

Handling Chat Messages

To send a message, call the `sendMessage()` method of the `InteractionChat` interface. The message is sent to all parties of the chat session.

```
myChatInteraction.sendMessage( "This is a chat message" );
```

Incoming `InteractionChatEvent` events of type `InteractionChatEvent.Type.MESSAGE_RECEIVED` correspond to chat messages. To read the message, use the `getText()` method of the `InteractionChatEvent` that is sent to your `InteractionChatListener`.

```
void handleInteractionChatEvent(InteractionChatEvent chatEvent ) {
        // Testing if the event is a chat message
        if(chatEvent.getEventType() == InteractionChatEvent.Type.MESSAGE_RECEIVED)
        {
                // Displaying the message
                String message = chatEvent.getText();
                String sender = chatEvent.getParty().getNickName();
                System.out.println("From: "+sender+"\n>"+message+"\n");
        }
}
```

> **Important**
> You can also access all received messages of the session by calling the `InteractionChat.getMessages()` method.

Handling Typing

To notify the parties that the user is typing a message, call the `InteractionChat.typingStarted()` method of the `InteractionChat` interface. The `InteractionChatEvent.START_TYPING` event is sent to all parties of the chat session.

```
myChatInteraction.typingStarted();
```

Incoming `InteractionChatEvent` events of type `InteractionChatEvent.Type.TYPING_STARTED` correspond to that typing notification. To get the name of the party who is typing, use the

getParty() method of the InteractionChatEvent.

```
void handleInteractionChatEvent(InteractionChatEvent chatEvent ) {
        // Testing if the event is a chat message
        if(chatEvent.getEventType() == InteractionChatEvent.Type.TYPING_STARTED        )
        {
                // Displaying the message
                String sender = chatEvent.getParty().getNickName();
                System.out.println(sender+" is typing...");
        }
}
```

If the user stops (without submitting the message), invoke the InteractionChat.typingStopped() method to notify other parties. Parties will receive the InteractionChatEvent.Type.TYPING_STOPPED event.

Push URL

Your application can now push a URL to the chat applications of other parties by calling the InteractionChat.pushURL() method.

```
myChatInteraction.pushURL("http://genesyslab.com");
```

The chat application for each participant then receives the InteractionChatEvent.Type.PUSH_URL event, which contains the pushed URL, which can be retrieved from the InteractionChatEvent.getText() method.

```
void handleInteractionChatEvent(
                InteractionChatEvent chatEvent ) {
        // Testing if the event is a chat message
        if(chatEvent.getEventType() == InteractionChatEvent.Type.PUSH_URL)
        {
                // Getting the URL
                String message = chatEvent.getText();
                // ... Display the URL ...
        }
}
```

Conferencing

The conference feature allows an agent to invite another agent to join the chat session by using the InteractionChat.conferenceAgent() or InteractionChat.conferencePlace() method.
The following code snippet creates a chat conference with the agent agent1.

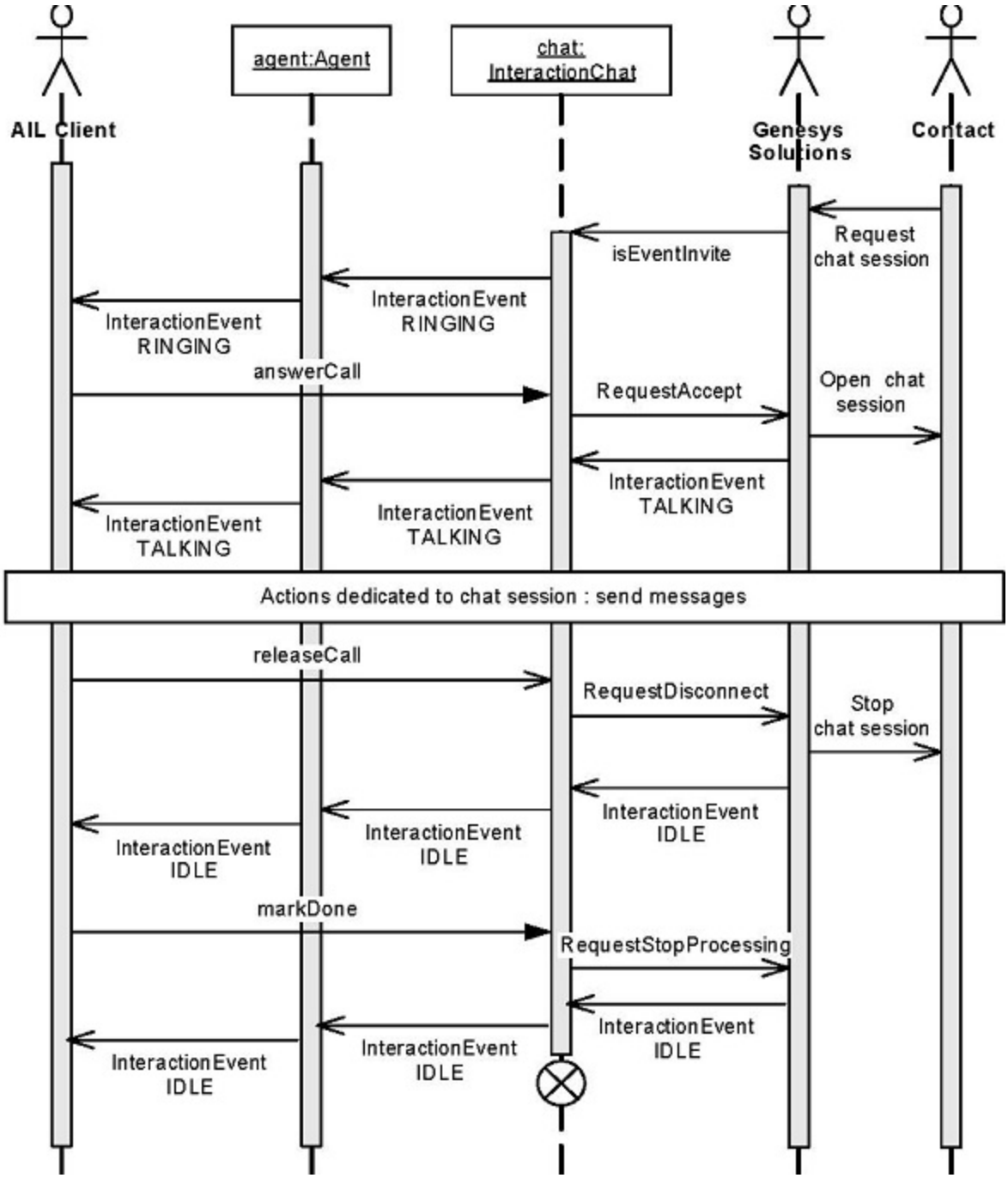
```
myChatInteraction.conferenceAgent("agent1", //agent who should join.
ChatParty.Visibility.INT,        // only agents can see him or her.
"reason for joining"); //a string reason.

The agent agent1 receives an InteractionEvent for a InteractionChat in
Interaction.Status.RINGING. If this agent answers the chat interaction, he joins the chat
session.
```

Terminating the Chat Session

To disconnect the chat session, invoke the `InteractionChat.releaseCall()` method. After receiving the `InteractionEvent` for `Interaction.Status.IDLE`, use the `markDone()` method to properly save and clean the interaction.

The e-mail server can have a strategy to send the transcript of the chat interaction to the contact. In this case, to disconnect, use `clearCall()` or `transferToQueue()` instead of `releaseCall()`. The transcript is automatically sent to the contact.
Chat event flow is shown in Processing a Chat Interaction.



Processing a Chat Interaction