Agent Interaction SDK Services Developer Guide

E-Mail Interactions

12/16/2025

Contents

E-Mail Interactions

The e-mail interaction service is defined by the `IInteractionMailService` interface of the `com.genesyslab.ail.ws.interaction.mail` namespace. To integrate this service, your application deals with classes and enumerations of this namespace, and also with the classes and interface of the `com.genesyslab.ail.ws.interaction` namespace.

Introduction

The e-mail interaction service performs actions on e-mail interactions and enables your application to benefit from collaboration features. E-mail interactions are specific objects representing e-mails—that is, interactions using the `MediaType.EMAIL` media.
The following subsections provide overviews of e-mail handling and collaboration handling, and identify the services upon which the e-mail service depends.

Common E-Mail Features

The e-mail service is designed to allow your application to provide standard e-mail handling features, such as:

- Creating an e-mail.
- Sending an e-mail.
- Replying to an e-mail.
- Transferring an e-mail.

Collaboration Features

The collaboration features allow your agent desktop application to send invitations to other agents, requesting their assistance.
For example, the agent using your desktop application might be writing an outgoing e-mail replying to a customer’s questions. But, this agent needs more information to answer a specific point, and therefore, requires collaboration with other agents.
The agent sends invitations to a set of other agents who might be able to provide assistance. Those invitations contain the agent’s question and the problem e-mail. These agents receive invitations; if they have information to help, they reply to the invitation. The e-mail service collects the replies so that the initiating agent can complete the e-mail and send it.
The e-mail interaction service provides the following collaboration features:

- For the agent initiating the collaboration:
  - Creating and sending invitations.
  - Recalling, or reminding other agents about, invitations.
  - Reading collaborative answers.

- For the agent receiving an invitation:
  - Accepting, declining, replying to an invitation.
  - Sending a collaborative reply.
  - Saving, closing, or deleting a collaborative reply.

E-Mail Service Dependencies

The e-mail interaction service only deals with actions on e-mail interactions. When integrating this service, your application has to take into account the events that can occur due to this service. The e-mail interaction service depends on the following other services:

- The event service, to subscribe to and receive events.
- The agent service:
  - To deal with e-mail interactions, your application must log an agent into an `EMAIL` media type with the agent service.
  - While the agent is logged into an `EMAIL` media type, your application can use the `IInteractionMailService` to perform actions on e-mail interactions.

> **Important**
> See The Agent Service.

- The interaction service:
  - To access to `IInteractionMailService` attributes, your application uses the `IInteractionService` DTO methods.
  - Event attributes of the`interaction.mail`

domain are published in `InteractionEvent` defined in the `IInteractionService` interface.

> **Important**
> See also The Interaction Service.

E-Mail Essentials

The `IInteractionMailService` requests a single action on a single e-mail interaction at a time. Your application receives `InteractionEvents` when events occur on e-mail interactions.

E-Mail Attributes

The `IInteractionMailService` has no methods to read attributes. Your application must call one of the `IInteractionService.getInteractionDTO*()` methods. See also Handling Interaction DTOs.

Common Interaction Attributes

For each e-mail interaction, the attributes of the `IInteractionService` interface are also available. Your application should often use the following attributes:

- `interaction:interactionId`—The system interaction identifier, required in calls to the methods of the e-mail interaction service.
- `interaction:status`—The interaction status, defined with the `InteractionStatus` enumeration.
- `interaction:eventReason`—The `InteractionEventReason` value published when an `InteractionEvent` event occurs for a voice interaction.

Common E-Mail Attributes

Attributes common to all types of e-mail interactions are defined in the `IInteractonMailService` interface, and belong to `interaction.mail` domain. The following list is representative (but not exhaustive):

- `interaction.mail:toAddress` is a string containing the e-mail address field for the e-mail's receiver.
- `interaction.mail:fromAddress` is a string containing the sender's e-mail address field.
- `interaction.mail:messageText` is a string containing the text of the e-mail.
- `interaction.mail:ccAddresses` is a string containing the e-mail addresses that are to receive a copy of the e-mail.

The subject of an e-mail is defined in the `interaction:subject` attribute of the `IInteractionService` interface.

Other E-Mail Attributes

The `IInteractionMailService` interface includes two additional subdomains dedicated to common e-mail interactions:

- `interaction.mail.in`—Subdomain dedicated to incoming e-mails.
- `interaction.mail.out`—Subdomain dedicated to outgoing e-mails.

These attributes are available according to the type of the e-mail interaction being processed. Those types are detailed in the E-Mail Types subsection below.
See the *Agent Interaction SDK 7.6 Services API Reference* for further details about the attributes of the `interaction.mail` domain and its subdomains.

E-Mail Types

Your application accesses types for e-mail interactions by reading the

`interaction:interactionType` attribute of the `IInteractionService` interface.
The types of interactions handled by the e-mail service can be divided into two categories:

- `EMAIL_*` for common e-mails interactions.
- `COLLABORATION_*` for collaborative interactions.

The following table presents the types of common e-mail interactions that the e-mail service handles.

Interaction Types for the E-Mail Service

Interactions	InteractionType	Description
Incoming E-mails	`EMAIL_IN`	Interactions for e-mails received by the agent
Outgoing E-mails	`EMAIL_OUT`	Interactions for e-mails sent by the agent
Outgoing Reply E-mails	`EMAIL_OUT_REPLY`	Interactions for e-mail replying to incoming e-mails

The e-mail interaction types are detailed in the following subsections. See Collaboration Interaction Types for further information about collaboration interactions.

Incoming E-Mails

The e-mail interaction type corresponding to incoming e-mails is `InteractionType.EMAIL_IN`. It represents e-mails received by an agent. For this type of e-mail interaction, common e-mail attributes—`interaction.mail:*` such as message, subjects, and addresses—are already filled.

The `interaction.mail.in:currentReplyMailoutId` attribute is specific to incoming e-mails, and is filled with an e-mail interaction identifier if there exists an outgoing e-mail (whether created or sent) replying to this incoming e-mail.

Use the `IInteractionService.getInteractionDTO*()` method to read this attribute value. To deal with incoming e-mails, see Answering an E-Mail and Replying to an E-Mail.

Outgoing E-Mails

The e-mail interaction type corresponding to outgoing e-mails is `InteractionType.EMAIL_OUT`. It represents e-mails interactions that your application creates and sends with the `IInteractionMailService` interface.
For this type of e-mail interaction, most of the writable attributes—such as message, subject and addresses—are empty when the interaction is created. Your application should fill them with `IInteractionService.setInteractionDTO()` method.
Some specific attributes are defined in the `interaction.mail.out` subdomain. interaction. Use also

the IInteractionService.getInteractionDTO*() method to read those attributes.
To deal with outgoing e-mails, see Sending an E-Mail.

Outgoing E-Mails for a Reply

The corresponding e-mail interaction type corresponding to outgoing reply e-mails is InteractionType.EMAIL_OUT_REPLY. It represents outgoing e-mails replying to an incoming e-mail. These interactions are created due to aInteractionMailAction.REPLY action performed with aIInteractionMailService.reply() call.
The attributes available are those for common outgoing e-mails, that is, interaction.mail:* and interaction.mail.out:*. For this type of e-mail interaction, common e-mail attributes—interaction.mail:* (such as subjects and addresses)—are already filled with information extracted from incoming e-mails.
To deal with reply e-mails, see Replying to an E-Mail.

E-Mail Actions

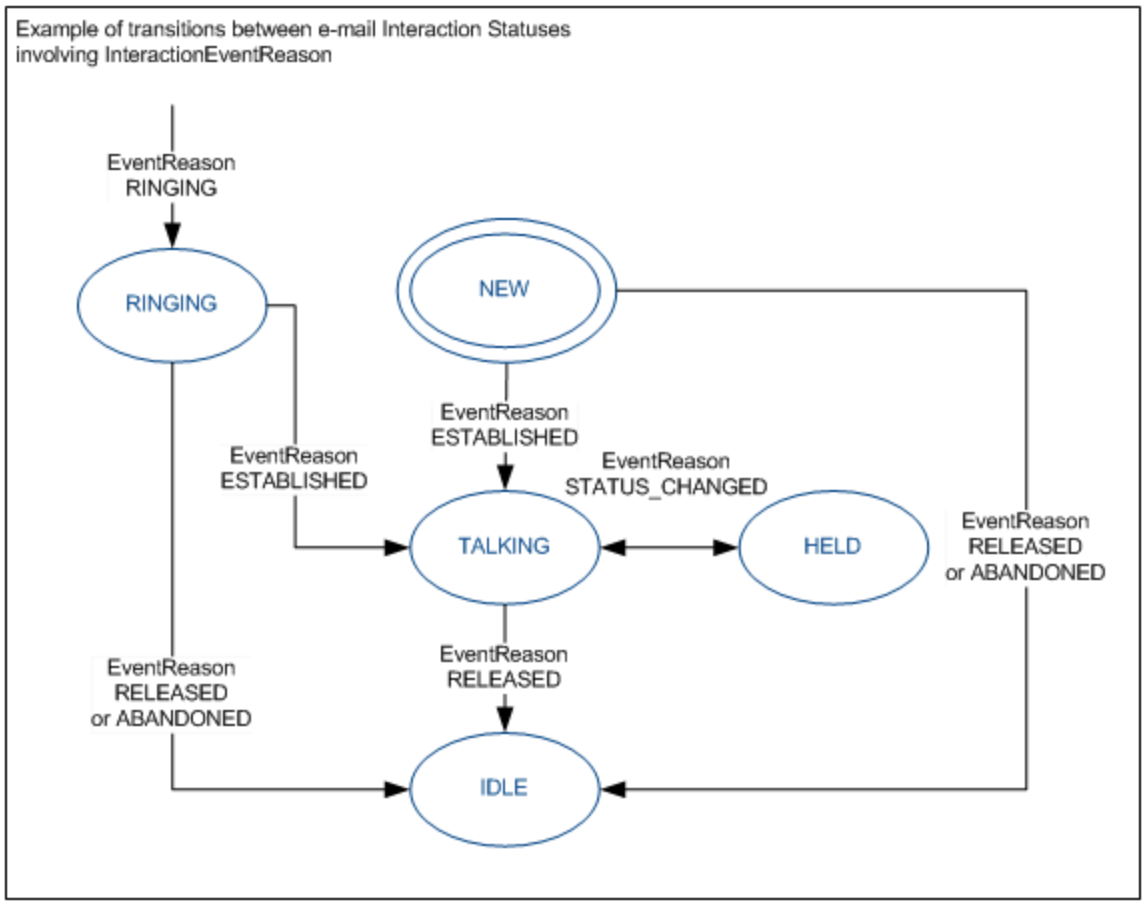
The InteractionMailAction enumeration defines all the available e-mail actions of the IInteractionMailService interface. Constants each correspond to one e-mail interaction service method. For example, the ANSWER constant corresponds to the IInteractionMailService.answer() method.
Your application can access possible actions for e-mail interactions (except for a collaboration interaction) by reading the value of the interaction.mail:actionsPossible attribute of the IInteractionMailService interface.
The IInteractionMailService interface has no methods to read attributes. Your application must use one of the InteractionService.getInteractionsDTO*() methods. See also Handling Interaction DTOs.

E-Mail Statuses

The status of an e-mail interaction is represented by a common interaction attribute defined in the IInteractionService as the interaction:status attribute. Your application cannot read this attribute with an IInteractionMailService method, so you must use one of the IInteractionService.getInteractionDTO*() methods. See also Handling Interaction DTOs.
The possible statuses of an e-mail interaction are parts of the InteractionStatus enumeration of the com.genesyslab.ws.interaction namespace.
The status of an e-mail interaction changes if:

- A successful action is confirmed by an event to the server-side application; for example, the e-mail has been sent and the e-mail interaction status change to InteractionStatus.IDLE.
- A CTI event occurred; for example an error occurred and e-mail interaction status change to InteractionStatus.IDLE.

The following diagram shows non-exhaustive transitions that can occur between e-mail interaction statuses.



Generalized State Diagram for E-Mail Interactions (Incomplete)

> ## Warning
> This figure is provided as an informative example. It is non-exhaustive: it does not include all the possible transitions.

The diagram above shows a sequence of e-mail interaction statuses, with `InteractionEventReason` values as transitions. The `InteractionEventReason` value for a status change is propagated with the `interaction:eventReason` attribute in `InteractionEvent`.

> ## Warning
> Do not assume any status sequence in your application design. Design your application always to update with the possible agent actions provided and the current interaction status.

E-Mail Interactions Events

The `IInteractionMailService` works with the `InteractionEvent` of the `IInteractionService`. Most events that occur on e-mail interactions are `InteractionEvents`. Therefore for these events, published attributes are `IInteractionService` and `IInteractionMailService` attributes that have the event property.
You have to subscribe to a `TopicsService` defined for the `IInteractionService`. This `TopicsService` must specify (in its `TopicsEvents`) the e-mail interaction DTO to retrieve.

> **Important**
>
> For further details on the `InteractionEvent` mechanism, see Using IInteractionService.

The following code snippet shows how to receive `InteractionEvent` occurring on any interaction belonging to `agent0`, and how to ensure the propagation of e-mail interaction DTOs for an e-mail interaction event.

```
/// Defining a Topic Services for interaction service
TopicsService[] myTopicsServices = new TopicsService[1] ;
myTopicsServices[0] = new TopicsService() ;
myTopicsServices[0].serviceName = "InteractionService" ;

TopicsEvent[] myTopicsEvents = new TopicsEvent[1] ;
myTopicsEvents[0] = new TopicsEvent() ;

/// the targeted events are InteractionEvent
myTopicsEvents[0].eventName = "InteractionEvent" ;
myTopicsEvents[0].attributes = new String[]{ "interaction:*", "interaction.mail:*"};

/// The InteractionEvent concern agent0 interactions
myTopicsEvents[0].triggers = new Topic[1];
myTopicsEvents[0].triggers[0] = new Topic();
myTopicsEvents[0].triggers[0].key = "AGENT";
myTopicsEvents[0].triggers[0].value = "agent0";

/// The InteractionEvent must concern some EMAIL types interactions
myTopicsEvents[0].filters = new Topic[3];

myTopicsEvents[0].filters[0] = new Topic();
myTopicsEvents[0].filters[0].key = "INTERACTION_TYPE";
myTopicsEvents[0].filters[0].value = "EMAIL_OUT";

myTopicsEvents[0].filters[1] = new Topic();
myTopicsEvents[0].filters[1].key = "INTERACTION_TYPE";
myTopicsEvents[0].filters[1].value = "EMAIL_OUT_REPLY";

myTopicsEvents[0].filters[2] = new Topic();
myTopicsEvents[0].filters[2].key = "INTERACTION_TYPE";
myTopicsEvents[0].filters[2].value = "EMAIL_IN";

/// ... Subscribe to the event service
```

See also The Event Service

Common E-Mail Management

The common actions of the `IInteractionMailService` on e-mail interactions are presented in Common Features for an E-Mail Interaction.

Common Features for an E-Mail Interaction

Actions	InteractionMail Action	IInteractionMail Service	Relevant InteractionTypes
Send an e-mail	`SEND`	`send()`	`EMAIL_OUT EMAIL_OUT_REPLY`
Answer an e-mail	`ANSWER_CALL`	`answer()`	`EMAIL_IN`
Reply to an e-mail	`REPLY`	`replyDTO()`	`EMAIL_IN`
Delete an e-mail	`DELETE`	`delete()`	`EMAIL_OUT EMAIL_OUT_REPLY EMAIL_IN`
Transfer an e-mail	`TRANSFER`	`transfer()`	`EMAIL_IN`
Release an e-mail	`RELEASE_CALL`	`release()`	`EMAIL_OUT EMAIL_OUT_REPLY EMAIL_IN`
Mark done an e-mail	`MARK_DONE`	`markDone()`	`EMAIL_OUT EMAIL_OUT_REPLY`

Your application uses the `IInteractionMailService` features to send requests to the server-side application. Once the Genesys Solution—that is, the server-side application and the Genesys Framework—has performed the corresponding action, the event service receives `InteractionEvent`s if the application has subscribed to the correct topics.
The `InteractionEvent` propagates the interaction status changes and the new possible actions corresponding to the e-mail interaction identifier (`interaction:interactionId`).
The following sections detail the common e-mail interaction actions and provide you with sequence diagrams to help you to understand the action sequences.

> ### Warning
> You should not use the sequence diagrams to make assumptions about actions' availability. Instead, use the `InteractionMailAction` possible actions provided in DTOs.

Sending an E-Mail

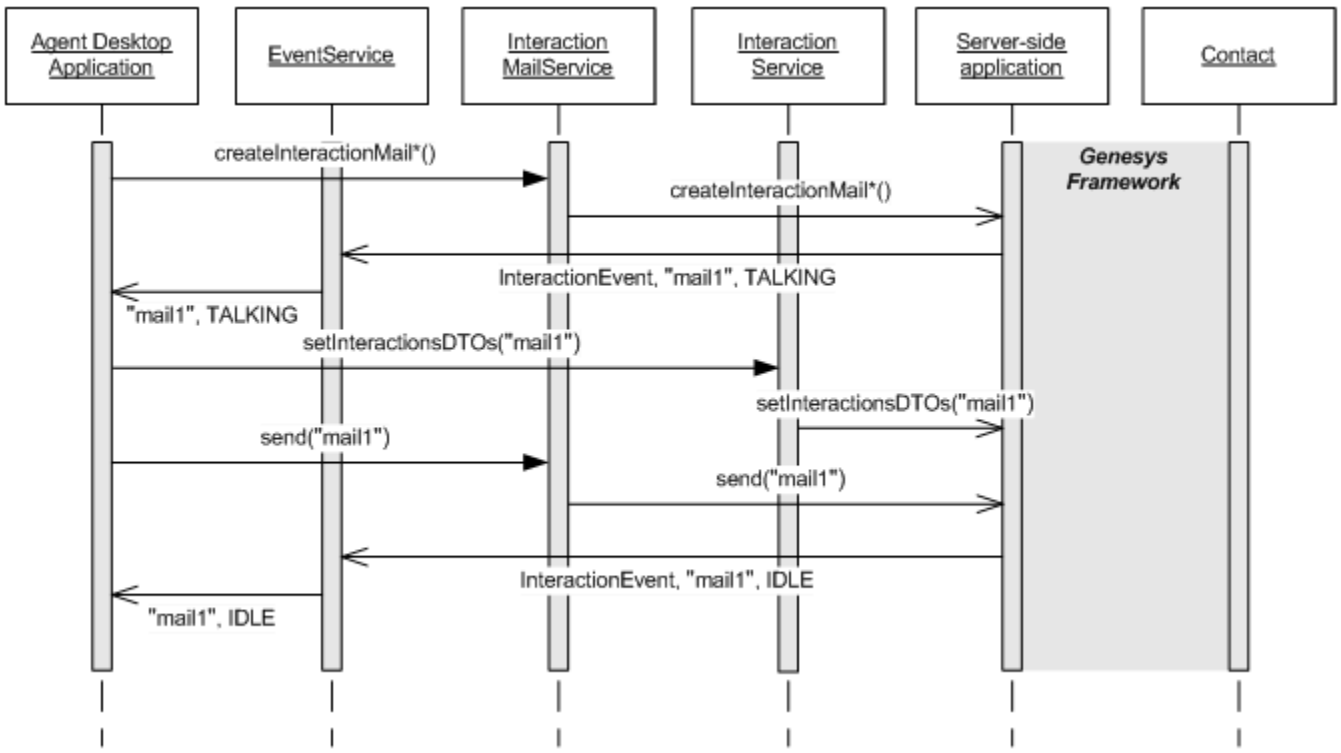
The `IInteractionMailService` interface lets your application send e-mails. There are two scenarios for sending an e-mail:

- Your application has replied to an incoming e-mail and needs to send the reply. See Replying to an E-Mail.
- Your application sends a brand new e-mail as detailed in this section.

This section and its subsections detail how to create and send common outgoing e-mails. The corresponding e-mail interaction type is `InteractionType.EMAIL_OUT`.
When your application needs to send an e-mail, it requires a logged-in agent on an `EMAIL` media type. Your application can test this condition with the `agent:loggedMedias` attribute of the `IAgentService` interface.
Sending the e-mail requires four steps, as follows:

1. Create an outgoing e-mail with one of the IInteractionMailService.createMailInteraction*() methods. See Creating an Outgoing E-Mail Interaction.
2. Fill the outgoing e-mail interaction fields with the `IInteractionService.setInteractionDTO()` method. See Filling an E-Mail Interaction.
3. Send the outgoing e-mail interaction with the `IInteractionMailService.send()` method. See Sending an E-Mail.

The following sequence diagram shows the sequence of actions, requests, and `InteractionEvents` received when sending an outgoing e-mail.



Sequence Diagram for Sending an E-Mail

In Sequence Diagram for Sending an E-Mail, once the outgoing e-mail interaction is created, this interaction status is `InteractionStatus.TALKING` as notified in the received `InteractionEvent`. This status lets your application modify the e-mail data—that is, addresses, text, and so on—with `IInteractionMailService` attributes that have the `write` property. Because the e-mail has just been created, these types of attributes have `null` values.
Then, the `IInteractionMailService.send()` method performs itself a release of the interaction once the outgoing e-mail is sent.

> **Important**
> To use `IInteractionMailService` methods, update the possible actions propagated in the received `InteractionEvent`.

The following subsections detail the method calls for creating and sending an e-mail.

Creating an Outgoing E-Mail Interaction

`IInteractionMailService` has two available methods for creating an e-mail interaction:

- `createInteractionFromPlaceDTO()` creates an e-mail interaction with a place identifier if:
  - An agent is logged on the place.
  - An agent is logged on the place's e-mail media type.
- `createInteractionFromAgentDTO()` creates an e-mail interaction with an agent identifier if the agent is logged into an e-mail media type.

> **Important**
> Both methods create an outgoing e-mail interaction.

The following code snippet creates an e-mail interaction using the `IInteractionMailService.createInteractionFromAgentDTO()` method:

```
/// Creating the e-mail interaction with the e-mail service
InteractionDTO myEMailDTO = myInteractionMailService.createInteractionMailAgentDTO(
myAgentId, // identifier of the logged agent
  myAgentQueue, // identifier of the agent queue
  null); //an array of key attributes to retrieve in the DTO

/// if the creation succeeded, retrieving the interaction id
if(myEMailDTO!=null)
{
   String myNewEMailId=myEMailDTO.interactionId;
}
```

The above code snippet shows that if the interaction is successfully created, its identifier is available in the `InteractionDTO` object returned by the method.

> **Important**
>
> Upon the outgoing e-mail creation, its writable attributes have `null` values.

Sending the E-Mail Interaction

The following code snippet shows how to send the previously created outgoing e-mail:

```
myInteractionMailService.send(myNewEMailId, // Id of the e-mail interaction to send
  myQueue); // The Queue (should not be null)
```

If the send action is performed, the outgoing e-mail interaction status changes to `IDLE` because the send feature has released the e-mail interaction. Your application receives an `InteractionEvent` with this new status and with the updated possible e-mail actions.

Filling an E-Mail Interaction

The e-mail interactions fields that your application might have to fill are `interaction:*` and `interaction.mail:*` attributes with the `write` property. See the *Agent Interaction SDK 7.6 Services API Reference* for further details.
Filling the e-mail interaction's fields requires the interaction identifier, an `InteractionDTO` object, and an `IInteractionService` instance. The following code snippet fills an empty outgoing e-mail identified with `myNewEMailId`.

```
/// Creating the DTO to fill with attributes key-values
InteractionDTO myEmailDTO = new InteractionDTO();

// Setting the id of the outgoing e-mail to fill
myEmailDTO.interactionId = myNewEMailId;

// Creating a Key-value array
myEmailDTO.data = new KeyValue[3];

// Setting the message text
myEmailDTO.data[0] = new KeyValue();
myEmailDTO.data[0].key= "interaction.mail:messageText";
myEmailDTO.data[0].value= “Text of the e-mail to send”;

// Setting the e-mail addresses
myEmailDTO.data[1] = new KeyValue();
myEmailDTO.data[1].key= "interaction.mail:toAddresses";
myEmailDTO.data[1].value= myContact@company.com;

//Setting a subject for the e-mail
myEmailDTO.data[2] = new KeyValue();
myEmailDTO.data[2].key= "interaction:subject";
myEmailDTO.data[2].value= "Subject of the e-mail";
```

```
// Writing the DTO with the interaction service
myInteractionService.setInteractionsDTO( new InteractionDTO[]{ myEmailDTO });
```

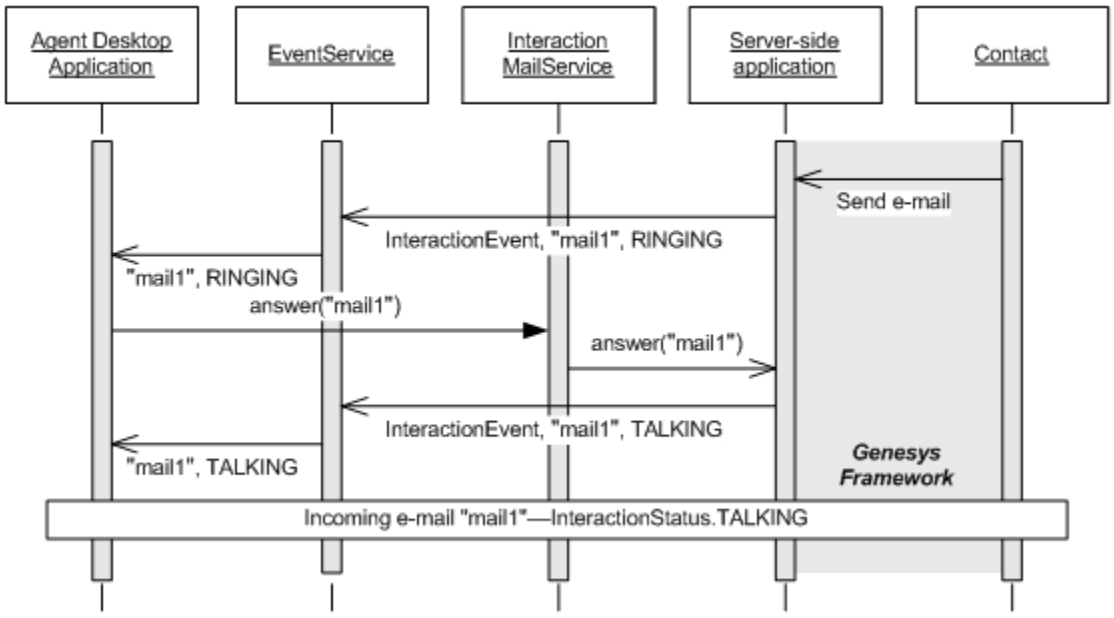
For further detail on `InteractionDTO` and `IInteractionService`, see The Interaction Service.

Answering an E-Mail

The answer feature of the e-mail interaction service is equivalent to the answer feature of the voice interaction service. Your application uses it on an incoming e-mail interaction so as to “accept” the interaction. Once your application has answered the e-mail interaction, the e-mail interaction is assigned to the agent’s place.

For example, your application receives an `InteractionEvent` for an incoming e-mail interaction with the `InteractionStatus.RINGING` status. Your application might then display a dialog box to inform the agent of the new incoming e-mail. If the agent chooses to answer the e-mail, your application can add the incoming e-mail to the agent desktop’s mailbox.

The `IInteractionMailService.answer()` feature works for incoming e-mail interactions. The corresponding e-mail interaction type is `InteractionType.EMAIL_IN`.
The following diagram shows the sequence of actions on incoming e-mail interactions, and the received `InteractionEvents`.



Sequence Diagram for Answering an E-Mail

This diagram shows that an `InteractionEvent` occurs for an incoming interaction e-mail with `InteractionStatus.RINGING` status. Once the `IInteractionMailService` has answered, the new interaction status is `Interactionstatus.TALKING`, which means that the incoming e-mail belongs to the logged-in agent.

The following code snippet illustrates the `IInteractionMailService.answer()` method call:

```
myInteractionMailService.answer( myIncomingMailId,// Id of the incoming e-mail to answer
   null); // KeyValue[] reasons
```

Unlike with a newly created outgoing e-mail, the incoming e-mail interaction attributes do not have `null` values. While the incoming e-mail interaction status is `InteractionStatus.TALKING`, your application might display pertinent information about the incoming e-mail—such as, the sender identity, the subject, and the message itself identified in `IInteractionService` and `IInteractionMailService` attributes.

For further detail about readable attributes of these services, see the *Agent Interaction SDK 7.6 Services API Reference.*

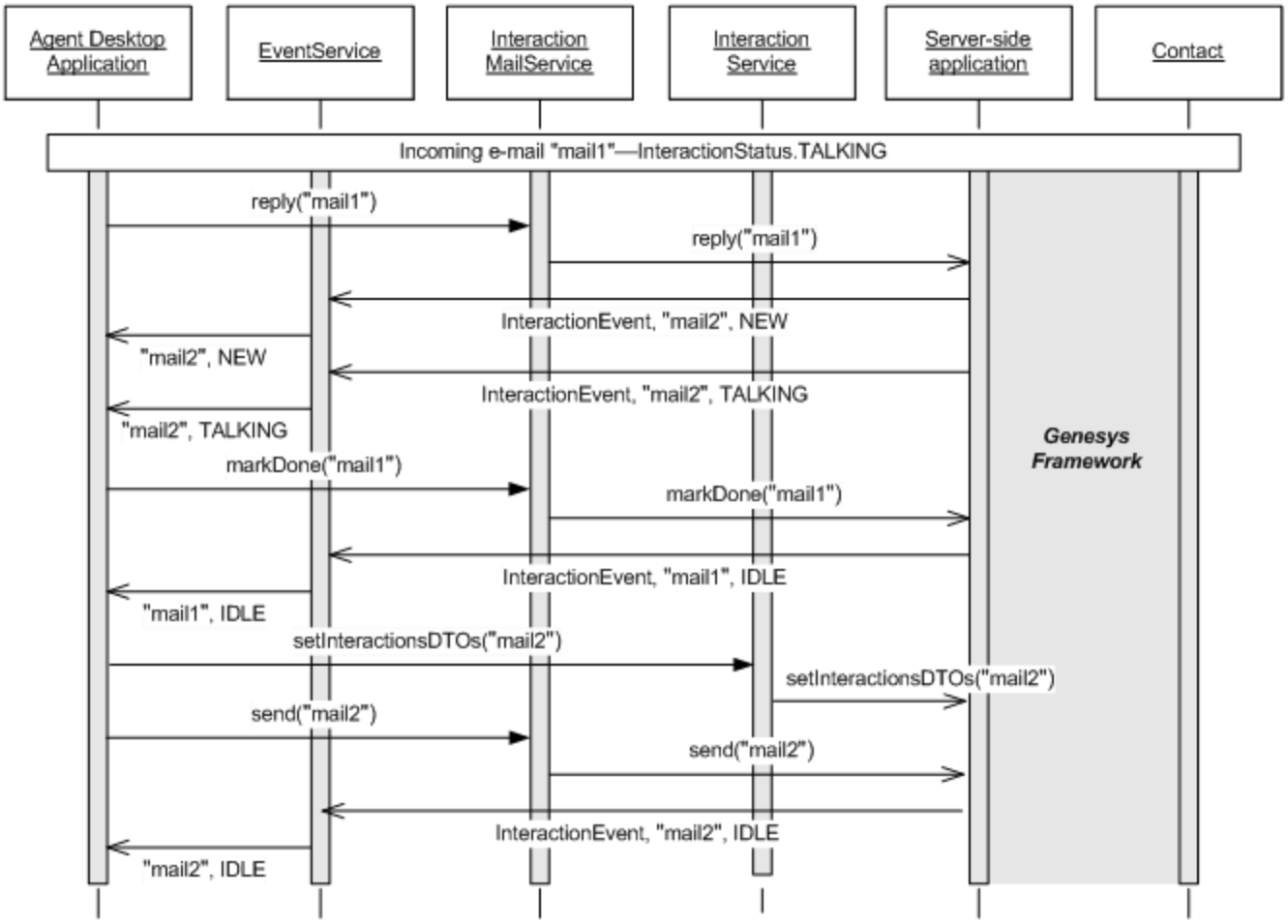
Replying to an E-Mail

Your application can use the `IInteractionMailService` reply feature only with e-mail interactions of type `InteractionType.MAIL.IN`. Moreover, this feature must be available in the possible actions of an incoming e-mail.
When the `IInteractionMailService` interface requests a reply, it creates an outgoing e-mail interaction of type `InteractionType.EMAIL_OUT_REPLY`, which your application must fill and send.
If you use the `IInteractionMailService.replyDTO()` method, replying to an e-mail involves the following steps:

1. Create an outgoing reply e-mail with the IInteractionMailService.replyDTO() method.
2. Mark the incoming e-mail interaction as done when you no longer need it. See Marking Done an E-Mail Interaction.
3. Fill the outgoing reply e-mail interaction fields with the `IInteractionService.setInteractionDTO()` method. See Filling an E-Mail Interaction.
4. Send the outgoing reply e-mail interaction with the `IInteractionMailService.send()` method. See Sending an E-Mail.

> **Important**
>
> If you call the IInteractionMailService.replyExDTO() method and set the auto-mark-done parameter to true, you do not have to mark the incoming e-mail interaction as done.

The following sequence diagram shows the sequence of actions, requests, and `InteractionEvents` sequence for a typical reply to an incoming e-mail, that is, by calling the `IInteractionMailService.replyDTO()` method.



Sequence for Replying to an Incoming E-Mail (no auto-mark done)

In the above diagram, `mail1` is an incoming e-mail interaction identifier. When the `IInteractionMailService` interface has made the reply request, the server-side application creates an outgoing reply e-mail, `mail2`, filled with some information of `mail1`.
The event service receives an `InteractionEvent` for the `mail2` interaction, notifying its `InteractionStatus.TALKING` status. The agent application can mark `mail1` as done and the event service receives the corresponding `InteractionEvent` event for the status change.
The agent desktop application can fill the remaining fields of`mail2` with the `IInteractionService` interface. Once the replied `mail2` interaction is sent, the server-side application releases the `mail2` interaction. The event service receives the corresponding `InteractionEvent` for the status change.
The following code code snippet shows how to implement the `IInteractionMailService.replyDTO()` method to reply to `mail1` and retrieve the corresponding interaction ID.

```
InteractionDTO myReplyDTO= myInteractionMailService.replyDTO("mail1",
   myQueue, // the queue ID
   true,    // Reply to all
   null);   // string keys of the attributes to retrieve
```

```
// Displaying the ID of the created interaction
System.Console.WriteLine("Replying Interaction Id: "+ myReplyDTO.interactionId);
```

> **Important**
> The `InteractionDTO` object can retrieve attributes that have the `read` property.

Marking Done an E-Mail Interaction

Your application should mark done an e-mail interaction when the agent no longer needs it. For instance, if the agent sent several replies to an e-mail and decides that this e-mail no longer requests agent processing, he or she marks it as done.
The following code snippet shows how to mark the previously created outgoing e-mail as done:

```
myInteractionMailService.markDone(myNewEMailId);
```

When the interaction is marked as done, your `IInteractionMailService` and `IInteractionMailService` interfaces can no longer access this interaction or perform your application's requests using its identifier.

Collaboration Essentials

A collaboration session involves several types of interactions. A collaboration interaction is an e-mail interaction that manages additional collaboration data. The e-mail interaction service includes collaboration attributes to access that data, which includes collaboration status.
During a collaboration session, your application can use the e-mail service to:

- Manage the collaboration, if the agent is the initiator.
- Participate in a collaboration session.

When an agent initiates the collaboration, he or she sends invitations to the participants. After a refresh of their applications, the agent can monitor the collaboration activity, and all the participants can access the corresponding invitations. When a participant has replied, the corresponding invitation is fulfilled.
If the agent is a participant, your application only manages interaction events and uses the e-mail service to perform collaboration actions on the collaborative interactions.
The following sections present the details behind this general description.

Collaboration Attributes

Your application can call one of the `IInteractionService.getInteractionDTO*()` methods to read the collaboration attributes. Opening a Workbin Interaction.

Collaborative Interaction Attributes

As the collaborative interactions are e-mail interactions, the following attributes are available for a collaborative interaction:

- `interaction:*`—Common interaction attributes.
- `interaction.mail.*`—Common e-mail attributes.
- `interaction.mail.in.collaboration:*`—Additional collaborative attributes for parent invitations.

Outgoing E-Mail Attributes

An agent initiates a collaboration session when writing an outgoing (reply) e-mail. Your application can monitor this session with some attributes dedicated to the collaboration management and defined in the `interaction.mail.out` domain:

- `interaction.mail.out:invitations`—All the invitations sent by the agent who initiated the collaboration session.
- `interaction.mail.out.invitationSentId`—The system identifier of an outgoing e-mail whose invitations were successfully sent to participants.

Your application can use the following `IInteractionMailService` DTO methods to get interaction data:

- `getSentInvitationsDTO()`—Retrieves the interaction data of each sent invitation.
- `getCollaborativeReplyDTO()` —Retrieves the interaction data of a received reply. See Retrieving a Collaborative Reply.

Collaboration Interaction Types

Your application accesses types for collaborative e-mail interactions using the `interaction:interactionType` attribute of the `IInteractionService` interface. The following diagram presents the types of collaborative e-mail interactions that the e-mail service handles.

Types of Collaborative E-Mail Interactions

Interactions	InteractionType	Description
Inbound invitation	`COLLABORATION_INVIT_IN`	Interactions for inbound invitations that the participant (or child) receives or that the agent (or parent) sent in a collaboration..
Reply to invitation	`COLLABORATION_REPLY_OUT`	Interactions for a collaborative reply sent

Interactions	InteractionType	Description
		by a participant (or child).

Incoming Invitation

The parent can see and manage the invitation interactions of the participants:

- Child invitation (invitation from the child point of view):
  - Each participant in the collaboration session receives an incoming child invitation.
  - This interaction informs the participant of the collaboration request.
  - For information about managing child invitation, see Participating in a Collaboration Session.
- Parent invitation (invitation from the parent point of view):
  - For each invitation sent to a participant, the agent who initiates the collaboration can access the corresponding interaction.
  - The agent uses parent invitations to monitor the collaboration and the participants'replies.
  - For information about managing parent invitations, see Managing a Collaboration Session.

Collaborative Reply

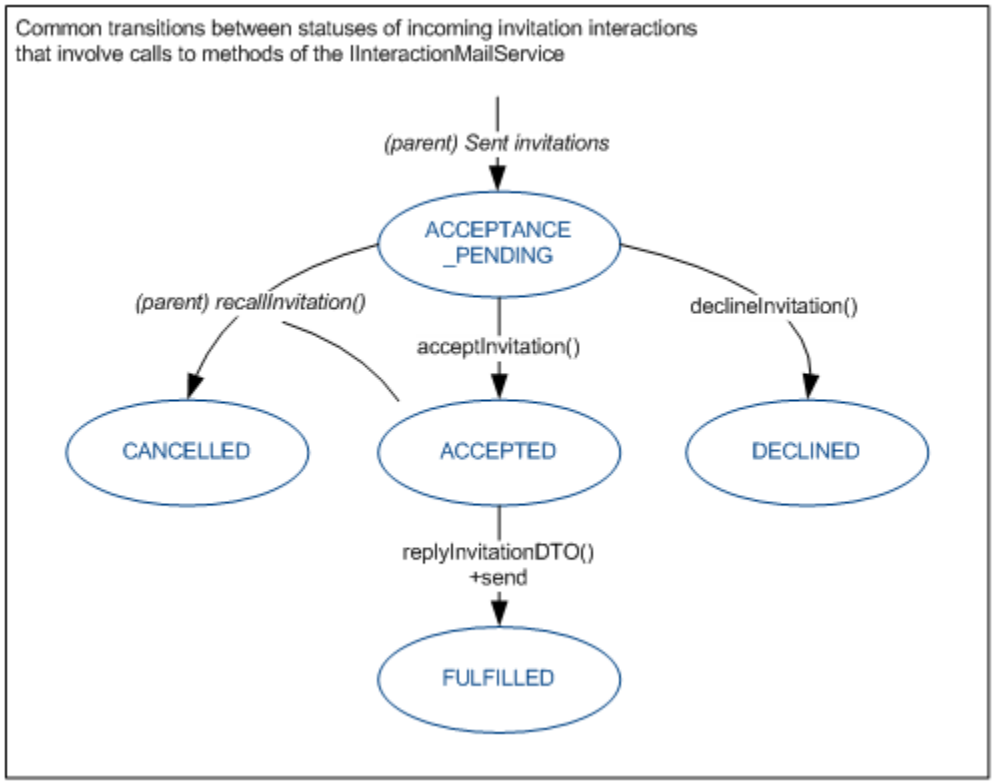
The collaborative reply is an outgoing e-mail replying to a child incoming invitation. As with an outgoing reply e-mail, some fields are filled at the interaction's creation—for instance, `interaction.mail:to` and`interaction.mail:from`.
Use a collaborative reply in the same way as an outgoing e-mail interaction. For further details, see Participating in a Collaboration Session.

Collaboration Status

The `CollaborationStatus` enumeration lists the possible collaboration statuses. Only collaborative interactions can have a collaboration status, available in the `interaction.mail:collaborationStatus` attribute. This status is an additional data. The `interaction:status` attribute is available for any collaborative interactions.
An application that initiates a collaboration has a specific interest in the collaboration status of its parent invitations.When a parent invitation takes on a `FULFILLED` status, the application can get the system identifier in the `interaction.mail.in.collaboration:collaborativeReply` attribute of this interaction to access the corresponding reply.

> **Important**
> Collaboration status changes do not launch additional `InteractionEvent` events. There is no notification.

To refresh the collaboration status of a collaborative interaction, your application must periodically read that status.
The following state diagram shows common transitions existing between collaboration statuses.



Generalized State Diagram for an Incoming Invitation (Incomplete)

Collaboration Handling

The typical `IInteractionMailService` actions upon e-mail invitations are presented in the following table.

Features For Collaboration

Agent	Actions	IInteractionMail Service	InteractionType
PARENT	Send an invitation	`sendInvitationTo*()`	`EMAIL_OUT EMAIL_OUT_REPLY`

Agent	Actions	IInteractionMail Service	InteractionType
	Remind an invitation	`remindInvitation()`	`EMAIL_OUT EMAIL_OUT_REPLY`
	Recall an invitation	`recallInvitation()`	`EMAIL_OUT EMAIL_OUT_REPLY`
	Retrieve DTOs for sent invitations.	`getSentInvitationsDTO()`	`EMAIL_OUT EMAIL_OUT_REPLY`
	Retrieve the DTO for a collaborative reply interaction.	`getCollaborativeReplyDTO()`	`COLLABORATION_INVIT_IN`
CHILD	Accept an invitation	`acceptInvitation()`	`COLLABORATION_INVIT_IN`
	Refuse an invitation	`declineInvitation()`	`COLLABORATION_INVIT_IN`
	Reply to an invitation	`replyInvitationDTO()`	`COLLABORATION_INVIT_IN`
	Send a collaborative reply	send()	`COLLABORATION_REPLY_OUT`

This table separately shows actions related to parent versus child invitation interaction. The `InteractionType` specified in the table's last column is the interaction type of the identifier parameter in the `IInteractionMailService` method call.

Managing a Collaboration Session

To request the collaboration of other agents, your application must be working on an outgoing e-mail interaction, as shown in Features For Collaboration. As your application initiates the collaboration, it becomes the parent of all sent invitations.
The following steps detail the general sequence of actions that your application is likely to follow:

1. Sending invitations to the participants.
2. Recalling, or reminding about, invitations if required.
3. Retrieving a DTO for each collaborative reply sent by a participant.
4. Sending the outgoing e-mail. For details, see Sending the E-Mail Interaction.

Sending Invitations

Depending on the method called to send invitations, your application activates a specific mode:

- `sendInvitations()`—Sends the invitations to the participants in the pull mode. The child invitations are available in workbins.
- `transferInvitations()`—Transfers the invitations to the participants in the push mode. Each participant receives the invitation as an incoming interaction.

A single call to these methods send all invitations, as shown in the following code snippet:

```
Participant[] myParticipants=new Participant[2];
myParticipants[0] = new Participant();
myParticipants[0].name = "agent0";
myParticipants[0].type = ParticipantType.AGENT;
myParticipants[1].name = "agent1";
myParticipants[1].type = ParticipantType.AGENT;
InteractionDTO[] mySentInvitations= myInteractionMailService.sendInvitations(myInteractionId,
myParticipants, "Do you have info about that?", //the trouble "Troubleshooting", //the
subject of the collaboration new string[]{"interaction.*:*"}); //the invitation attributes
//to return in DTOs.
```

The method returns an array of interaction DTOs. Each interaction DTO contains the data of a sent invitation.

Reminding About Invitations

Sometimes, agents who received invitations might forget to reply. Your application can use the `IInteractionMailService.remindInvitation()` method to remind a participant that a collaboration session is still in progress. This method does not inform all participants—only a single one. Call this method if the invitation's collaboration status is: `ACCEPTED` or `ACCEPTANCE_PENDING`. It takes as a parameter the interaction identifier of the parent invitation that corresponds to one participant.
The following code snippet shows a call to this method:

```
myInteractionMailService.remindInvitation( "parentInvitationIdForAgent0", "myPlaceId"); //
place ID of the agent reminding the invitation
```

Retrieving a Collaborative Reply

When a collaboration participant sends a reply to the inviting agent, your application propagates the identifier of the collaborative reply in the `interaction.mail.in.collaboration:collaborativeReply` attribute.
The following code snippet retrieves a collaborative reply interaction with the getCollaborativeReplyDTO() method:

```
InteractionDTO myReplyDTO = myInteractionMailService.getCollaborativeReplyDTO( "myReplyID",
// interaction ID of the reply
new string[]{"interaction.*:*"}); // attributes to get in DTO
```

> **Important**
> Replying to a collaborative reply is not possible.

Recalling an Invitation

If the initiating agent no longer needs the collaboration session—for example, he has found the necessary information—he or she can decide to recall some pending invitations.
In this case, your application uses the `IInteractionMailService.recallInvitation()` method to cancel the invitations. This method does not recall all invitations—only a single one. Call this method if the invitation's collaboration status is: `ACCEPTED` or `ACCEPTANCE_PENDING`. It takes as a parameter the interaction identifier of the parent invitation that corresponds to one participant. If the recall is successful, the collaborative status of the invitation changes to `CANCELLED`.
The following code snippet shows a call to this method:

```
myInteractionMailService.recallInvitation( "parentInvitationIdForAgent0",
"myPlaceId"); // place ID of the agent recalling the invitation
```

Participating in a Collaboration Session

In push mode, when an agent receives an incoming invitation, he or she receives an e-mail interaction of type `COLLABORATION_INVIT_IN` which has both `InteractionStatus.RINGING` and `CollaborationStatus.ACCEPTANCE_PENDING` statuses.
If the agent accepts the invitation, that agent participates in the collaboration session. To end his or her participation, the agent must reply to the invitation, as described in the following subsections.

Accepting an Invitation

Once the application has answered the interaction (see Common E-Mail Management), your application can accept the incoming invitation to enter the collaboration session, by calling the `IInteractionMailService.acceptInvitation()` method, as shown in the following code snippet.

```
myInteractionMailService.acceptInvitation( "myChildInvitationID", //Interaction ID
  "myPlaceID");
```

If the `ACCEPT_INVITATION` action is successful, the collaboration status of the the invitation interaction should become `CollaborationStatus.ACCEPTED`.

For further details about collaboration statuses, see Collaboration Status.

Replying to an Invitation

Your application can use the `IInteractionMailService.replyInvitationDTO()` method to create a collaborative reply interaction of type `COLLABORATION_REPLY_OUT`, as shown in the following code snippet.

```
InteractionDTO myCollaborativeReplyDTO = myInteractionMailService.replyInvitationDTO(
"myChildInvitationID",// Interaction ID
"myPlaceID",
new string[]{"interaction.**:*"}); /// attributes to retrieve

String myCollaborativeReplyID = myCollaborativeReplyDTO.interactionId;
```

The returned `InteractionDTO` contains the identifier of the created interaction. Use this identifier to

fill the e-mail using the `IInteractionService.setInteractionDTO()` method, as detailed in Filling an E-Mail Interaction.
Once the collaborative reply is filled, your application can send it as an outgoing e-mail using the `IInteractionMailService.send()` method, as shown in the following code snippet:

```
myInteractionMailService.send( myCollaborativeReplyID, myQueue); // The Queue (should not be
null)
```