Agent Interaction SDK Java Developer Guide

Service Status and Connection

12/16/2025

Contents

Service Status and Connection

This chapter explains how to deal with connection maintenance.

Service Status Design

Basically, a service represents the status of a connection to a server in the Genesys Framework or in Multi-Channel Routing. To handle connection changes, you implement a `ServiceListener` class, that monitors events on `ServiceStatus` objects. Then your application registers this listener for each service to be listened.
To determine which service your application will be able to listen, refer to the `ServiceStatus.Type` enumeration:

- `CONFIG` —Connection to the Configuration Layer.
- `TELEPHONY` —Connection to a T-Server.
- `IS` —Connection to an Interaction Server.
- `CHAT` —Connection to a Chat Server.
- `AIL` —for AIL itself.
- `DATABASE` —Connection to the Contact Server database.

The possible `ServiceStatus.Status` values are:

- `ON` —Server is running and connection is established.
- `OFF` —Server is down or connection is broken.
- `ABSENT` —Connection has never been established with this server.
- `LICENSE` —Connection could not be established for license reasons.
- `RESTORING` —Connection is being restored.
- `ALREADY_RUNNING` —Server is already running.
- `UNKNOWN` —Server status is unknown.
- `UNKNOWN_HOST` —Server host is unknown.
- `APPLICATION_NAME_INCORRECT` —Connection could not be established due to an incorrect application name.
- `APPLICATION_TYPE_INCORRECT` —Connection could not be established due to an incorrect application type.
- `LOGIN_INCORRECT` —Connection could not be established due to incorrect credentials.
- `CONNECTION_FAILED` —Connection could not be established.

Connection Loss

If AIL loses its connection to a server, your application gets an event related to the associate service, turning its status to `OFF`. See Steps for Listening to Service Status for further details about receiving these events.
However, the connection loss has additional repercussions for the objects that depend on the disconnected server.

T-Server Use Case

If AIL loses its connection to T-Server (for instance, the network link breaks), the associated `TELEPHONY` service turns its status to `OFF`. As a consequence, from your application's point of view, the DN and its voice interactions are no longer available:

- if your application implements the `handleDnEvent()`method of your `PlaceListener`, it catches a `DnEvent`which notifies the `OUT_OF_SERVICE` status (`DnEvent`.EventReason.STATUS_CHANGED).
- if your application implements the `handleInteractionEvent()` method of your `PlaceListener` , it catches an `InteractionEvent` which notifies the IDLE status (`InteractionEvent`.EventReason.ABANDONED).

Actually, if only the network link breaks, voice interactions still exist (the agent and the customer are still talking); the strategy determines whether interactions are re-routed or not.

> Important
>
> Do not assume that if your interaction status turns to `IDLE` (reason `ABANDONED`), or if your DN status is `OUT_OF_SERVICE`, AIL has lost its connection to T-Server. Rely on the service status to diagnose the lost connection.

Multimedia Use Case

In case your application loses its connection to the Interaction Server (for instance, the network link breaks), the associated `IS` service turns its status to `OFF`. As a consequence, from your application's point of view, the media are no longer available.

- if your application implements the `handlePlaceEvent()`method of your `Place` , or `Agent`instance, it catches a `PlaceEventMediaStatusChanged`which notifies the `Media.Status.OUT_OF_SERVICE` status (`Media.Reason.STATUS_REASON_CHANGED)` .
- if your application implements the `handleInteractionEvent()` method of your `PlaceListener` , it catches an `InteractionEvent` which notifies the IDLE status (`InteractionEvent`.EventReason.ABANDONED) for each multimedia interaction.

As with voice interactions, media interactions are `IDLE` only from the API point of view. On the Interaction Server side, the place's media are logged out. The interactions are no longer associated with the disconnected place and are pushed back in queues (according the deployed strategy).

Reconnection

When AIL loses its connection to a server, it tries to reconnect till the connection attempt succeeds. Then, your application gets an event for the associated service, turning its status to `ON` . See Steps for Listening to Service Status for further details about receiving these events. As for the disconnection, the reconnection generates several events to update object statuses.

T-Server Use Case

For example, in the case that the link to a T-Server is restored, the switch is able to provide AIL with information. So, the `Agent, Place,` and `Dn` instances update, and your application gets status notifications according to the implemented event handlers. Additionally, if the agent is still talking to the contact, your application will get a new `InteractionVoice` instance in `TALKING` status,`InteractionEvent.EventReason.ESTABLISHED` .

Multimedia Use Case

At the reconnection, AIL logs into the media of the place. If your application implements the `handlePlaceEvent()`method of your `PlaceListener` , it catches a `PlaceEventMediaStatusChanged`which notifies the new `Media.Status.READY` status (`Media.Reason.BACK_IN_SERVICE`) . Then, getting back interactions is a job for the strategy or the agent application.

Restart all the Connections

At runtime, your application deals with a unique instance of the `AilFactory` . If you need to restart the AIL library and all its connections to Genesys servers, first kill your instance of `AilFactory` by calling the `AilLoader.killFactory()` method, as shown in the `release()` method of the `Connector` application block.

```
mAilLoader.killFactory();
```

If this method call succeeds, you can get a new reference on the `AilFactory` singleton (see Five Rules to Build an AIL Server Application).

Important

Genesys recommends the use of `ailLoader.getFactory()` in your AIL client application (instead of having a reference to the singleton throughout the code). This decreases the risk of reference issues associated with `killFactory()` usage.

Steps for Listening to Service Status

Now that you have been introduced to connection maintenance, it is time to outline the steps you will need to work with its events and objects.
As specified in the previous section, you need to register a listener per `ServiceStatus` object that your application should listen to. In the following code snippets, a single listener class is implemented to listen to the whole set of `ServiceStatus` objects.
There are five basic things you will need to do in your AIL applications to monitor service connections:

- **Implement a ServiceListener class**. Here is how a SimpleService class would do this:

```
public class SimpleService implements ServiceListener {
```

- **Implement the serviceStatusChanged() method**, that notifies service status changes. In the following code snippet, the code implemented displays those statuses in real time.

```
// This method must be implemented because this class
 // implements ServiceListener

 public void serviceStatusChanged(
                ServiceStatus.Type service_type,
                String service_name,
                ServiceStatus.Status service_status) {

        System.out.println(“Connection maintenance - “ + service_type.toString()+ ”: ” +
 service_name+ “ in status “ + service_status.toString());
 }
```

- **Register the listener for each service to be monitored** using this listener. In the following code snippet, the constructor of `SimpleService` registers for all services.

```
//This constructor registers for all service available in AIL
public SimpleService(AilFactory ailFactory)
{
        // For each service...
        Iterator it = services.entrySet().iterator();
        while (it.hasNext()
        {
                Map.Entry entry = (Map.Entry) it.next();
                // Get the service name
                String name = (String) entry.getKey();
                ServiceStatus service = (ServiceStatus)        entry.getValue();
                ailFactory.addServiceListener(service.m_type, this) ;
        }
}
}
```