Chat Server Administration Guide

Async Chat Overview

12/17/2025

Async Chat Overview

Provisioning

In order to enable async chat capabilities, the following must be provisioned:

- During a chat session creation, special key-value pairs must be provided when creating chat session via GMS API:
  - `GCTI_Chat_AsyncMode=true`. This enables a special processing in Chat Server and Agent Desktop, and can also be used in the workflow logic.
  - Provide specific data to enable push notifications (even if you do not need to process them). This forces the Chat Server to use flex-push-timeout instead of flex-disconnect-timeout for the async chat session. If you plan to process push notifications, the corresponding push notification functionality in GMS must also be configured and the following data must be provided:
    - When using Chat API Version 2 with CometD, provide both `push_notification_deviceid` and `push_notification_type`. Note that the CometD client is limited to a single active connection with GMS in order to receive the chat session activity events.
    - When using GMS REST API V2, see Push notifications via GMS to HTTP server. Although The REST API does not impose a single active connection restriction, please carefully review the performance implications described in the deployment guidelines for async and regular chat.
- Chat Server (version 8.5.301.06 and higher required) configuration must be reviewed for the following configuration options:
  - To prevent the chat session from being closed when a mobile application disconnects with GMS, set the value of the **flex-push-timeout** option to a larger value (for example, 86400 seconds). If there is no protocol activity from a mobile application for the duration of this timeout, Chat Server checks with GMS to see if a connection with a mobile application is still alive. If GMS does not confirm the liveness of a connection, Chat Server sets the timer again and, when the timeout expires, removes a customer from the chat session if no protocol activity is detected.
  - Consider adjusting (only if needed) async-idle-alert , async-idle-close , async-idle-notices . Default values of these options enable async inactivity control monitoring for async chat sessions. The value of option **async-idle-notices** also defines the condition when `GCTI_Chat_AsyncStatus` is updated to a value of "2".
- To enable a session restoration by Agent Desktop or Workflow, set the appropriate value for the configuration option session-restore-extend-by (introduced in Chat Server version 8.5.312.10) and session-restore-push-send (available in Chat Server version 8.5.316.02), and also set the value for flex-push-on-join (Introduced in Chat Server version 8.5.315.05) to `true`. This functionality is supported in Workspace Desktop Edition (WDE) version 8.5.145.06 or later. Upon joining a chat session, a custom Agent Desktop can also implement a session restoration by making several attempts to reconnect to that chat session and providing a special KVP “ChatServerWebapiToken” in the `userdata` of the **Join** request (this must be taken from the interaction `userdata`). Performing a session restoration using Agent Desktop is required only for a GMS-based web chat; it is not used in social messaging channels where Digital Messaging Server (DMS) is solely responsible for the chat session restoration.

> **Important**
>
> WDE can only restore the chat session from the same Chat Server instance which was initially processing the chat session prior to being restarted. WDE is not aware of (and so cannot use) any other instance of Chat Server.

- Special workflow which implements the processing of async chat (Chat Business Process Sample provides a demonstration of these capabilities). To extend this sample with a session restoration (introduced by the configuration option **session-restore-extend-by** = `esp`), the following items must be changed in **async-chat-stuck-strategy**:
  - Extend the External Services Protocol (ESP) request **GetSessionInfo** method with the parameter `InduceRestore=true` (alternatively, you can use any other ESP method). This triggers the session restoration in Chat Server, however the ESP request itself will fail if the chat session does not exist in this instance of Chat Server. Otherwise, the block will exit through the green port, and the workflow can continue operations as normal.
  - The red port of this ESP IRD block must be connected with a function to wait for a timeout (for example 2-4 seconds).
  - The function block must then be connected to another ESP block (with the same or different ESP method). At this moment the chat session is restored and the ESP request is successful. If it fails again (due to possible delays with the session restoration, for example), you can repeat the ESP request a few more times with a delay between these subsequent requests. Alternatively, if the Chat Server that had been stopped cannot be re-started immediately, you can omit the Chat Server application name in the ESP request in order to let the chat session restore itself on a different instance of Chat Server.
  - This functionality can be tested by:
    1. Setting smaller values for **async-idle-alert** and **async-idle-close**.
    2. Starting the async chat session.
    3. Closing the Widget.
    4. Restarting the Chat Server instance before `GCTI_Chat_AsyncCheckAt` expires (which is slightly larger than the sum of **async-idle-alert** and **async-idle-close**).
    5. As soon as `GCTI_Chat_AsyncCheckAt` expires, the Workflow sends the interaction into **async-chat-stuck-strategy** and attempts to restore the chat session.

Async inactivity control

- Async inactivity control is different from regular inactivity control (specified in configuration of section **[inactivity-control]**) in a way that it does not require the presence of an agent in the chat session.
- *Activity* in inactivity control means any activity in the chat session that is visible to all participants and that is not generated by bot participants.
- It is not recommended you enable regular inactivity control (section **[inactivity-control]**) for Chat Server which handles async chat sessions, as it contradicts with the nature of long chat sessions.
- `async-idle-alert/close` reuses the `message-alert/close` options from the section **[inactivity-control]** for the notification.

Minimum release requirements

Workspace Desktop Edition

Workspace Desktop Edition (version 8.5.122.08 and higher) provides:

- Additional **Place Chat On Hold** button (configuration option **chat.on-hold-queue**) which allows an agent to return the chat session into the workflow (in other words, put into a dormant state) where it can await customer activity.
- The possibility to open the active chat from the **workbin** and **contact history** windows.

Genesys Mobile Services (GMS)

The mobile or web (in other words, Widget) application, which implements a chat client for a customer, must operate with GMS using Chat API V2 with CometD and enable push notification functionality (either mobile or custom http). This keeps the chat session running for as long as it is needed. GMS version 8.5.114.09 or higher is required.

Historical reporting

There are minimum release requirements for multiple additional components to enable historical reporting on async chat sessions. For full details, see the Prerequisites table at Integrating Chat Server with Genesys Historical Reporting.

Async chat states

`GCTI_Chat_AsyncStatus` is an integer and any positive value may be used to trigger the workflow to route an interaction to an agent. The following values are possible:

Value	Description
-2	Set when a chat session is placed on hold or into a dormant state (in other words, placed into the workbin or queue by an agent).
-1	Set when an agent is processing a chat session.
0	Undefined value (value is not used).
1	Signifies a newly created chat session.
2	Set when a customer posted a reply (sent a message) into a chat session while there were no agents in that chat session.
3	Set when async inactivity control timeout expires (meaning Chat Server will soon close the chat session).
4	Set when an agent desktop abnormally disconnects from the Chat Server (in other words, in case of

Value	Description
	accidental exits).
5	Set when the agent transfers the chat interaction by placing it into a queue.

Additional information

- Workflow with a special processing must be deployed (see the sample workflow provided in Deployment). It must evaluate the following interaction properties:
  - `GCTI_Chat_AsyncCheckAt` contains a **timestamp** when an interaction must be checked. This property must be used to detect "stuck" interactions in workflow which happens under abnormal situations (for example, in a case when Chat Server has stopped, and a chat session was never restored on another instance of Chat Server).
- Async chat sessions can be processed in the same way as non-async chat sessions. Async chat simply extends the functionality for chat session processing.
- In Async mode (in other words when CometD is used with GMS, and the rate of messages in a single chat session is low), a single instance of Chat Server is capable to support a greater number (up to 5000) of concurrent chat sessions. However, the Chat Server limitation of concurrent connections (4K on Windows and 32K on Linux) must be taken into account when planning your deployment. When calculating the number of total expected connections for web or mobile chat, consider the following:
  - Every agent desktop and every chat bot require a separate persistent connection to chat session. For agents, multiply it by the maximum possible capacity of agents.
  - Additionally, supervisor monitoring requires a separate persistent connection.
  - Client (in other words, consumer-facing) operations do not keep a persistent connection with Chat Server. The connection is only established when the consumer sends the request with a message or notice, or when Chat Widget requests periodic pull transcript updates (which is applicable only in non-CometD mode in GMS).Once the request is complete, the connection is immediately closed.