Chat Server Administration Guide

Genesys Engage Chat 8.5.3

1/31/2025

Table of Contents

Chat Server Administration

The following topics provide information for Chat Server administrators:

Topic	Description
Overview	Provides an overview of the chat solution, including a high-level architecture diagram.
Sizing Guide, Setting Load Limits, and Health Monitoring	Describes how much load a solution can hold, how to restrict the load and how to monitor the health per Chat Server instance.
Deploying a Chat Solution	Describes how to deploy a Chat Solution.
Deploying High-Availability Chat Server	Describes how to deploy multiple Chat Server instances in high availability mode.
Multilingual Processing	Describes how to configure a solution to process/work with multiple languages.
Masking Sensitive Data	Describes how to mask out sensitive data in chat session messages/transcripts and in Chat Server logs.
Inactivity Monitoring	Describes how to configure chat session closure upon participants' inactivity.
Matching Contact Attributes	Describes the approach to contact identification and creation.
Sending ESP requests to Chat Session from Workflow	Describes how to send messages, notices, and other requests from workflow (like URS/ORS strategies) to an active chat session.
Chat Server Reporting Data	Describes Chat Server reporting statistics attached to the user data of the interaction in Interaction Server.
File Transfer in Chat Session	Describes how to deploy and configure file transfer between chat session participants.
Chat Server API selected notes and topics	Describes selected cases and topics on how to use Chat Server API for implementation of custom desktop and widget.
Asynchronous Chat	Describes how to work with Asynchronous (async) chat within Genesys Chat Solution, Workspace Desktop Edition (WDE), and Widgets.
Chat Business Process Sample	Provides a sample workflow which demonstrates how to process chat interactions for both regular and asynchronous chat with different channels including web chat, Apple Business Chat, and WhatsApp.
Rich Messaging Support	Provides information on the Chat solution's ability to use Rich Messaging across various chat channels.

Overview

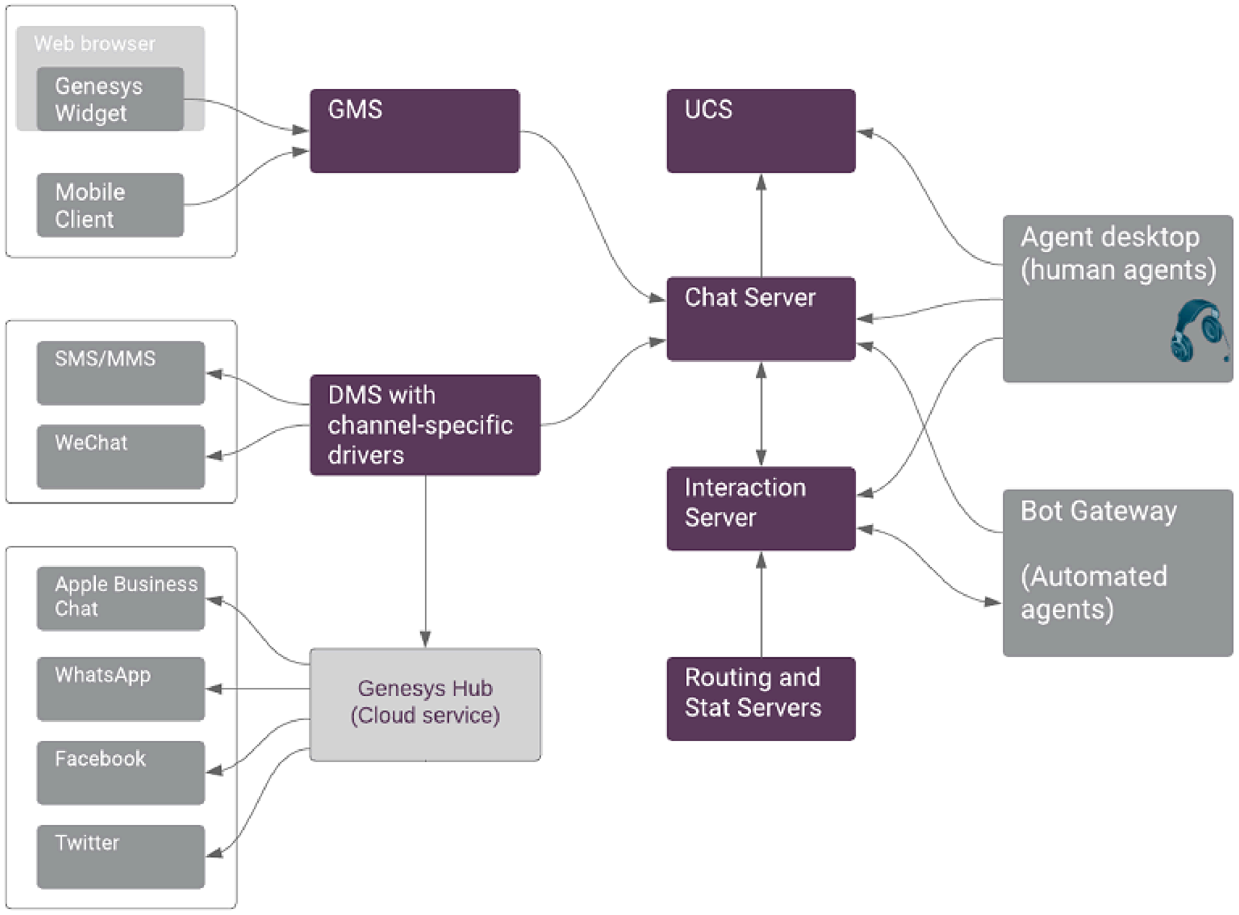
Genesys Chat allows customers to communicate with live and automated (bot) agents in a contact center. It uses a set of different components, each providing a unique piece of functionality.

From a business perspective, we view a chat communication as:

1. A customer starts a chat from a company web page, mobile application, or through a supported messenger application.
2. From the contact center, the chat can first be answered by a bot and then be routed to a live agent, or routed from the agent to a bot as needed.
3. Upon the chat completion, the chat transcript is saved in the historical records and the contact center reporting reflects the processing of this chat conversation.

The diagram below depicts essential components of the chat solution and the most significant communication channels between applications.

Components



From a technical standpoint, this works in the following way with the Genesys Chat Solution:

1. A contact uses either Genesys Chat Widget (or a custom web chat application), or mobile application which communicates with Genesys Mobile Engagement (GMS) public-facing REST (or CometD) API.
2. GMS communicates with a backbone Chat Server. When a new chat session is being requested, Chat Server:
    - Communicates with Universal Contact Server (UCS) to create a record (with known attributes) and identifies the customer with the contact attributes provided.
    - Creates the interaction in Interaction Server which is sent to the workflow and executed by Universal Routing Server (URS) and/or Orchestration Server (ORS) applications. The workflow permits system messages to the chat session to invite bots to route the interaction to the best available agent or the last handling agent.
3. If bots are involved, Bot Gateway Server (BGS) connects the bot to the chat session upon the request from the workflow. As soon as the bot has finished, the workflow continues.

4. The workflow invites an agent (represented by the agent desktop application) and Genesys provides both Workspace Web Edition (WWE) and Workspace Desktop Edition (WDE) agent desktops. **Note:** Customers can implement their own custom desktop by using Platform SDK (PSDK) or Web Services and Applications (GWS) API to Genesys components.
5. If the agent accepts the invitation for chat, the agent desktop connects the agent to the chat session.
6. If the chat session is running in Asynchronous (async) mode, the agent can put the conversation on hold by disconnecting from the chat session and placing the interaction into the agent Workbin. Later, the agent can resume the conversation or, alternatively, the workflow can be configured to alert an agent or to start routing the interaction upon a reply from the customer.
7. Upon the completion of the chat, UCS saves the final version of the chat session transcript in the UCS record. The workflow sends the offline interaction for post-processing (for example, sends an email to the customer with the chat session transcript) and then stops the interaction.
8. For historical reporting, during chat session closure, Chat Server provides a set of internal metrics which describes the chat session behavior and participants. This information is absorbed by Genesys Info Mart (GIM) (the core historical component) and later is used in historical reports provided by Genesys CX Insights (GCXI).
9. Other messaging channels can be added to the chat solution by deploying Digital Messaging Server (DMS) based components, such as Short Message Service (SMS), Apple Business Chat, WhatsApp, WeChat, Facebook, or Twitter. All messaging channels communicate through Chat Server, utilizing the same functionality in workflow, bots, and reporting. For the agent desktop in WDE, special plugins are used which implement channel-specific features such as structured messages, and others.

Entities

	Chat session	Interaction	UCS record
Purpose	Connects chat participants and allows message exchange	Used for routing, other services invocation (from routing strategies), reporting (real time and historical)	Stores information about the chat session, including the chat session transcript (in XML form), and can be associated with the contact record
Controlled by	Chat Server	Interaction Server and Routing (continued in Interaction Server database)	UCS (continued in UCS database)
Created	When customer (or agent in outbound mode) initiates the chat session	When Chat Server submits the interaction upon chat session creation	When Chat Server creates the record upon chat session creation
Updated	For every event in chat session (adding or removing participant, message or notice, idle control)	In async mode, special attributes are updated when a qualified event in chat session occurs	Chat Server stores intermediate versions of the chat transcript in high availability mode
Finished	When the last participant leaves (either voluntarily or forcefully) the chat	Chat Server sets an offline attribute in interaction, and the workflow stops the	Record stays permanently, with a special "closed" status mark

	Chat session	Interaction	UCS record
	session	interaction	

Features

For chat channels, Chat Server provides the following features:

- All chat conversations can be conducted in multiple languages (including the use of emojis) without any special configuration. For the correct processing of chat session interaction attributes, Unicode Transformation Format-8 (UTF-8) mode can be enabled for all components.
- The cleanup of Personally Identifiable Information (PII) can be enabled in configuration settings. For example, forcing Chat Server to replace digits in credit card numbers with the asterisk symbol, preventing it from appearing in the chat session transcript.
- Inactivity control can be enabled to detect a conversation's inactivity, send an alert to the chat participants, and to close the chat session automatically if the activity is not resumed.
- For web chat, the full high-availability support is provided by utilizing either UCS or Cassandra as the storage for the intermediate content of the chat conversation.
- Every chat session can be silently monitored by a supervisor. The agent can invite another agent into the same chat session either in conference mode (when the second agent is visible to a customer), or in consult/coaching mode (when the second agent is invisible to a customer).
- Most of the channels (including web chat) support file transfer, where both a customer and the agent can send files to each other. Some channels support Rich Messages or so-called structured messages (for example, list picker in Apple Business Chat, or a carousel for web chat).

> **Important**
> Workspace Web Edition does not support file transfer. Make sure to disable file transfer in Chat Server and CX Widget when using WWE.

Additional functionality provided by other components:

- Agent Desktop provides:
  - Agent access to the full contact history (all previous chat and non-chat interactions with a customer).
  - Standard responses with pre-populated content. Such responses are especially important when the agent needs to send structured messages.
- A URS/ORS workflow allows the interaction to be routed to the "last called agent" (in other words, the agent who handled the previous interaction from this customer), which is utilized in async chat when a customer's message awakens the interaction in the workbin.

Sizing Guide, Setting Load Limits, and Health Monitoring

Sizing Guide

The following guidelines are recommended for a Genesys Chat solution running on a single host with two Intel Xeon 3.0 GHz processors. Observe these recommendations for an optimum (without significant delay) performance.

Item	Maximum
Message size	4 KB (Genesys Desktop limitation; Chat Server does not have this restriction.)
Transcript size	54 KB (Genesys Desktop limitation; Chat Server does not have this restriction.)
Concurrent sessions (in a realistic simple scenario)	1000 per Chat Server
Messages per second	50 (rare temporary peaks up to 150)
Sessions opened and closed per second	10 (rare temporary peaks up to 30)

Chat Server limits the maximum number of concurrent opened connections (for all protocol and participants) with:

- 32768 on Linux.
- 4096 on Windows.

When opening and closing the connection, Chat Server prints in the log the current number of opened connections (as "conns="). Connections, with the exception of a few persistent system ones, are used in a couple of different ways:

- Each agent or bot connected to a chat session consumes one connection and then releases it when they leave the chat session.
- Each time a customer chat application sends a request (either via REST API or CometD), GMS opens the connection and then, after receiving a reply from Chat Server, closes the connection.

Note: You can configure a timeout in Chat Server that prevents keeping unused connections open. Use the `user-register-timeout` option to set the maximum time between establishing the connection and receiving:

- Either a basic protocol `Register` request
- Or any flex protocol request

Connection Delay with Antivirus

It may take some time (up to several minutes on some UNIX Platforms) for Chat Server to connect to an unopened port on a Windows host that is running an antivirus program. For example, if Chat Server is running on Linux and is trying to connect to an inactive UCS instance, it could take up to three minutes for Chat Server to detect that the listening port is not open.

Setting Load Limits

Starting in the 8.5.0 release of Chat Server, you can impose load limits on Chat Server: when Chat Server reaches the specified limit, it no longer creates new sessions or restores existing sessions.

Set load limits using the following configuration options (full descriptions are in the eServices Options Reference):

- Enable or disable the general functionality of load limitation using limits-control-enabled.
- Set specific limits:
  - limit-for-flex-users — Maximum number of currently logged-in flex users.
  - limit-for-reply-delay — Configures the maximum average delay (in milliseconds) for processing requests. The average value is calculated on the limit-average-interval interval. This delay increases if the Chat Server instance is overloaded with a large number of incoming requests. **Note:** This delay directly corresponds to CPU consumption by the Chat server process; it starts to grow when the Chat Server CPU is running closer to, or higher than, 100% of one CPU core.
  - limit-for-sessions — Maximum number of concurrent chat sessions (the value of which must not be larger than the value for limit-for-flex-users).

  If any of these limits is reached, Chat Server stops creating and restoring sessions.
- If Chat Server is configured in primary/backup mode (not recommended; see Deploying High-Availability Chat Server), you may want to stop it from reporting `service unavailable` to SCS when a limit is reached. You can do this using limits-reached-report-scs. Blocking the reports avoids a scenario in which Chat Server in primary/backup mode closes a chat session because (1) Chat Server reaches any of its configured load limits, (2) Chat Server sends a `service unavailable` notification to Solution Control Server, (3) SCS switches Chat Server to backup state, which closes the chat session. (This scenario does not apply if Chat Server is in N+1 mode: multiple Chat Servers with no backup configured).
- Set the point at which Chat Server returns to full functionality using limits-restore-threshold. This value is a percentage of the limit set by the three **limit-for-X** options.

Example

If **limit-for-flex-users** is set to 400 and **limits-restore-threshold** is set to 80, then:

1. When the number of flex users reaches 400, Chat Server stops creating and restoring sessions, and rejects login attempts by flex users.
2. When the number of flex users falls to 320, Chat Server returns to full functionality.

KPI (Key Performance Indicator) counters

Starting with release 8.5.103, Chat Server includes KPI (Key Performance Indicator) counters that monitor activity within the server.

Accessing KPI counters

Access KPI counters in one of two ways:

- The Chat Server log. by configuring options log-output-content, log-output-proviso, log-output-timeout in the **[health-service]** section.
- The web REST interface which you can configure by:
  - Adding a port with the ID=`health` to the ports of Chat Server.
  - Adjusting the **soap-*** options found in the **[health-service]** section.

Web interface

Access the web interface through the following URL format: `http://ServerName:ServerPort/Counters?<list of parameters>` where:

- *ServerName* is the host name where Chat Server is running.
- *ServerPort* is the web service port, also specified as the `health` port of Chat Server.

Web interface parameters:

Name	Description	Valid Value	
metadata	Returns a list (in JSON format) of all supported counters, with descriptions.	`true`, `yes`, `false`, `no`	`false`
content	Returns a list (in JSON format) of counters according to the provided parameter value.	`all`—returns all available counters new—returns only recently updated counters set—returns all initialized (non-zero) counters	`all`

Name	Description	Valid Value	
reset	Resets all counters to zero.	`true`, `yes`, `false`, `no`	`false`
prometheus	Returns metrics in Prometheus format. (Introduced in Chat Server version 8.5.312.10)	`true`, `yes`, `false`, `no`	`false`
gms_info	Returns metrics about GMS nodes and the associated chat session (only the Prometheus format is supported). When used, all other parameters are ignored. (Introduced in Chat Server version 8.5.312.10)	`true`, `yes`, `false`, `no`	`false`

If a parameter is omitted or has an invalid value, then the default value is used for that parameter. Parameters are processed according to the following rules:

- If **reset** is `true`, then all the counters will be reset and then the rest of the parameters will be processed.
- If **metadata** is `true`, then the `content` parameter will be ignored and the metadata will be sent.

Example URLs:

- `http://`*`hostname`*`:7000/Counters?content=set`
- `http://`*`hostname`*`:7000/Counters?metadata=true&reset=true`

How counter values are calculated

All counters, except the **process memory** counter, are cumulative. The value begins to accumulate the moment the application is launched or the counters are reset. To calculate the difference, the user must use two sample counters from different times and subtract the earlier sample from the later one. To find the counter's rate value, divide the difference by the number of elapsed seconds between the two samples.

Important

On some platforms, the time for processing internal activities may be reported as zero. This does not indicate an issue with the counter. On the contrary, a rapidly growing value on the counter for internal activity indicates that the server is overloaded.

Deploying a Chat Solution

This page outlines the essential steps of deploying a Chat Solution.

Overview

To deploy Chat Server, perform the following steps:

- Create a Chat Server application in configuration. Make sure that the following configuration is specified correctly:
    - **Ports**. At least 3 ports must be configured with the following IDs: `default`, `webapi` and ESP. Additionally, if KPI must be exposed via REST API, port with ID `health` must be added.
    - **Connections**. Chat Server must be connected to Interaction Server and UCS, and optionally Chat Server could be connected to the Configuration Server application (usually confserv) and Message Server. Setting addp is recommended for all these connections.
    - **Endpoints**. The `endpoints:*tenant_dbid*` section must be renamed to contain an appropriate tenant ID value (for example `endpoints:1`) and the `default` option must be initialized with a queue name (to which Chat Server will submit interactions).
    - **Logs**. By default logs are configured to hide all possible sensitive data - which however might not be convenient if troubleshooting is required. Decide on how you want to set the hide-attached-data and message-log-print-size options in the `settings` section and options in the `log-filter-data` section.
    - Set up Chat Server High Availability configuration if needed.
    - Connect other applications to the Chat Server application:
        - **Interaction Server** must be connected to the ESP port (addp is recommended for this connection).
        - **GMS** must be connected to the `webapi` port (addp is not recommended for this connection due to short living nature of these connections).
- Install the Chat Server installation package (IP).
- Configure other applications to work with chat channels:
    - Create/modify **capacity** rule to include the `chat` media type.
    - Create/modify **workflow** (using either IR Designer or Composer) to route a chat interaction. For information about sending messages to a chat session refer to How to Send Message or Notice to Chat Session from Workflow.
    - Enable **GMS** to provide chat API (by adding the `chat.customer-support` section).
    - Configure **agents** (persons in configuration) with a chat channel access.
    - Adjust configuration of **Workspace** (agent desktop) if needed. For WDE review the `chat.*` and `chatserver.*` options and for the `openmedia.workitem-channels` option add the `chat` value.

Interaction Server Cluster Support

Please refer to Interaction Server Cluster documentation (in particular, review Special Considerations for Media Servers in Suggested Deployment Configuration) in order to configure Chat Server to work with Interaction Server Cluster.

Important

The ESP Configuration Must Be Symmetrical section means that in order to be able to send ESP messages from workflow to Chat Server using Interaction Server Cluster, each Interaction Server node in the cluster **must** have its own connection to ESP port of Chat Server.

UCS 9.1 support

In order to configure Chat Server 8.5.x with the cluster of UCS 9.1 nodes (all of which are running in active mode):

1. Configure all UCS instances as primary and backup pairs:
   a. UCS 9.1 ignores this configuration.
   b. Chat Server instances will be using this configuration.
2. Divide all instances of Chat Server into the same number of groups as the number of UCS pairs.
3. For each group, connect each instance of Chat Server only to the primary UCS of a corresponding UCS instances pair.

For maintenance/upgrade:

1. Ugrade all instances of UCS configured as backup (stop instances, upgrade the IP, and then start instances again).
2. Stop all UCS instances configured as primary. At this point Chat Server instances will switch to UCS instances which are configured as backup.
3. Upgrade all UCS instances configured as primary (stop instances, upgrade the IP, and then start instances again).

Important

It is possible that in some situations—for example, after performing the upgrade procedure—Chat Server instances could be connected to UCS instances configured as backup (you can verify this from the Chat Server logs). In a case such as this, there will simply be an additional reconnect in Chat Server instances to UCS during the upgrade procedure.

Deployment guidelines for async and regular chat

The page provides some important guidelines regarding regular (traditional) and async (asynchronous) chat deployment.

- Comparison of regular vs. async chat mode
- Comparison of short polling (REST-API-based) vs. CometD-based chat
- Sizing guidelines based on comprehensive stress testing
- Async chat workflow recommendations
- How disconnects and idle timeouts work

Regular vs. async chat mode

In general, both async and regular chats are processed the same way by all components. However, async chat provides additional capabilities that require a bit more planning and workflow implementation.

	Regular Chat	**Async Chat**
Duration of single conversation	Lasts only minutes or dozens of minutes.	Could potentially last for days or even weeks.
Agent handling	An agent can: • Accept the chat session • Transfer the chat session • Stop the chat session	In addition to the agent handling found in regular chat, an agent can also: • Place the chat session on-hold • Resume the chat conversation at a later time by re-activating the interaction from the workbin
Mobile oriented	Can be implemented, but not suited for lengthy conversations.	Suitable for mobile applications as it permits lengthy conversations with periods of inactivity.
Workflow	Mostly used for routing purposes (in other words, selecting the best available agent).	Additionally, must handle re-activation of interactions placed on hold after a qualified event occurs (for instance, a new incoming message from a customer, or an async idle timeout expiration).

	Regular Chat	Async Chat
Performance implications	Must be sized to conduct a certain number of active chat sessions.	Must take into the account the presence of a large number of chat sessions, most of which are expected to be in a dormant state. Please see below about sizing guidelines.

Short polling vs. CometD-based chat

End-user (customer-oriented) web or mobile chat applications must communicate with Genesys Mobile Engagement (GMS) through two alternative APIs:

- Short polling (REST API) - For this API, the chat application is required to send frequent (usually every other 3 seconds) polling requests to keep the chat session transcript updated.
- CometD-based - This API can utilize either WebSockets or long-polling. This provides chat session transcript updates more promptly, on the customer side.

While the CometD approach naturally appears more efficient (as it also reduces the overall load onto the system by eliminating unproductive API calls), the table below provides a comparison of different aspects which should be taken into account when selecting the best approach for your deployment and implementation:

	Short polling (REST API)	CometD-based
Performance	Consumes more CPU and traffic resources as it produce a lot of unproductive API calls (according to research, on average 98.5% of polls are wasted). So, if a message is expected to be posted into the chat session every 30 seconds, an extra 10 unproductive API requests must be processed during this time by the GMS and Chat Server components. Basically, this means that the load on these components are measured on how many concurrent chat sessions an instance can hold, independently of the scenario density (see Sizing Guide, Setting Load Limits, and Health Monitoring for more information).	CPU and traffic resources are used mostly for the productive load.
Connections	Each API call is executed on a separate connection, which is closed immediately after receiving an HTTP response. The number of concurrent connections (between GMS and	Number of concurrent connections is similar to the amount of concurrent chat sessions. Also, GMS imposes a limitation of only one CometD connection every chat session.

	Short polling (REST API)	CometD-based
	Chat Server) depend on the number of concurrent chat sessions, divided by the short polling interval (usually 3 seconds).	
Complexity	Simple to implement and troubleshoot (as it is based on pure REST API).	Troubleshooting requires the knowledge of CometD protocol functionality.
Client library	There are numerous stable versions of HTTP REST libraries.	CometD client library is required, which increases the complexity of the chat web application.
Timeouts (in Chat Server)	The flex-disconnect-timeout configuration option is used to disconnect a chat participant who has not sent any API requests after a specified amount of time.	The flex-push-timeout configuration option is used to disconnect a chat participant who is not confirmed by GMS as alive after a specified amount of time.

Sizing recommendations

On the high level, sizing guidelines depend on various factors:

- Short polling (GMS Chat API Version 2) vs. CometD (GMS Chat API Version 2 with CometD) mode:
  - Short polling produces a constant "background" (or, "noise") load onto GMS and Chat Server, thereby consuming much more CPU and network resources. Also, it is important to appropriately tune the operational system Transmission Control Protocol (TCP) parameters to minimize the `TIME_WAIT` state duration, as each short polling request leads to the opening and closing of the TCP connection.
  - The CometD approach requires you to keep a long-living connection to GMS for each chat session. It should be noted that some load balancing solutions do not handle long-living connections properly and might result in the closure of an inactive connection.
- In chat async mode, dormant vs. active sessions:
  - Active chat sessions usually constitute only a fraction of all ongoing async chat sessions. The number of such chat sessions should be around the number of active chat agents, multiplied by the capacity of agents (or, how many parallel chat session an agent can work on). These chat sessions consume almost all assigned resources (first of all CPU).
  - Dormant chat sessions are those which do not have an active agent (or bot) in the chat session. So, for example, in short polling mode the customer-facing application must minimize the resource consumption by reducing (or completely eliminating) the periodic short polling requests.
- Scenario density: Using a CometD approach, the load scales linearly with the number of messages sent from chat participants during a certain time period. Using the short polling approach, the load is mostly dependent on the “noise“ load, since polling requests take up most of the packets being processed.
- In High Availability (HA), UCS vs. Cassandra:
  - UCS-based HA option requires less deployment and maintenance efforts, and guarantees the presence of the latest transcript version for ongoing chat sessions in the UCS database (DB).

However, with large deployments, UCS and UCS DB might be overloaded with intermediate transcript updates which are generated by Chat Server after each chat session message.

- Cassandra allows you to off-load UCS from an unnecessary load. However, in the case of an unplanned Chat Server switch-over during the ongoing chat session, the chat transcript can never be propagated into the UCS record if the chat session cannot be restored on another Chat Server instance (in other words, when it coincides with a customer-facing chat application failure or closure).

Important

For async chat, especially in short polling mode, a customer-facing web or mobile chat application must noticeably reduce the frequency of short polling requests when it detects that a session was placed on hold (in other words, when the agent leaves chat session.

Performance benchmarks

The following benchmarks were produced on:

- Hardware with "Intel Xeon E7-8880L 2 GHz" with a single instance of Chat Server which consumed in average one CPU core. At the maximum possible load, Chat Server can consume no more than 2 CPU cores since all chat-processing operations are executed in a single thread (similar to `Node.JS`) and only auxiliary activity operations are executed by a few other working threads.
- Two instances of GMS, each consuming approximately one CPU core.

The average length of a chat session was around 35 seconds (with 3 messages from a customer and 3 messages from an agent), which is a very dense scenario. Each cell in the table contains the total number of concurrent chat sessions (and active vs. dormant ratio).

Mode	Active to dormant sessions ratio		
All active	**1:10**	**1:50**	
Short polling mode	1000	8000 (800 / 7200)	35000 (700 / 34300)
CometD-based mode	1500	11000 (1000 / 10000)	39000 (900 / 38100)

Important

These are performance load test benchmarks; these numbers are not expected in a real-life scenario.

Async chat workflow recommendations

For async chat, the workflow (or the set of Universal Routing Server [URS] or Orchestration Server [ORS] strategies) must additionally provide the handling of chat sessions being placed on hold by an agent to the regular chat workflow. The on-hold session can be processed in the follow ways:

- Upon the qualified event (a message or a configured notice) in the chat session from a customer, Chat Server updates a special key-value pair (KVP) in the corresponding interaction, which is handled by Interaction Server. As you see it implemented in the Chat Business Process Sample, the workflow can force the interaction for routing, which routes the interaction to any other agent after several attempts of trying to route it to the last handling agent.
- Alternatively, the workflow can place the interaction back to the last handling agent's workbin, if that last handling agent is not available at the moment. However, in this case, the workflow must implement the "escape" to avoid this interaction being stuck forever, if that last handling agent never comes back to the interaction.
- With a custom desktop, the workflow might not force the interaction to routing at all upon the qualified event. In this case, the agent desktop can directly subscribe to notifications from Interaction Server when the interaction properties are changed in the agent workbin.
- The workflow must ensure that interactions are not stuck when placed on hold. In the Chat Business Process Sample, this is implemented in `async-chat-main-check-view` view of `async-chat-main-queue` with the condition `GCTI_Chat_AsyncCheckAt < _current_time()`, where `GCTI_Chat_AsyncCheckAt` is set by Chat Server to the sum of async-idle-alert and async-idle-close configuration options of Chat Server application.

How disconnects and idle timeouts work

Chat Server configuration options allows you to specify timeouts to control two different functional areas:

- The disconnect of a chat session participant, which leads to the removal of a chat participant from a chat session.
- The absence of an activity from participants in a chat session, which leads first to an alert notification and then the closing of a chat session (if no activity is being produced since the alert was sent).

To describe each functional area, the following definitions must be introduced:

- "Protocol inactivity" means the absence of any protocol requests from a client to Chat Server for a certain period of time. It is used to detect the disconnect of a chat participant. For example, if the client application sends short polling refresh requests it still resets the timeout for protocol inactivity even if it does not carry any useful load. So, the client is considered active on the protocol communication level.
- "Session inactivity" means the absence of qualified events (such as messages) from the chat participants with full visibility in the chat session. For example, if a customer and an agent are not sending messages for a certain period of time, then it is considered as a session inactivity. If, at the same time, the agent communicates with another agent invisibly from a customer (or, consultation call), it does not affect this decision (as this conversation is not fully visible for all chat session participants).

Important

A chat session stays alive in Chat Server until at least one participant is present. As soon as the last participant leaves, Chat Server closes the chat session forever and it cannot be resumed again.

Chat session participant disconnect and removal

In terms of connectivity, chat session participants can be processed differently depending on how the application (representing the participant) communicates with GMS and/or Chat Server:

- An agent (or bot) participant communicates with Chat Server via persistent TCP network connection, thus the disconnect leads to the immediate removal of a participant from a chat session.
- A chat participant, represented as a customer in a chat session (or, "client participant") can communicate with GMS in three different modes. Each mode utilizes different configuration options:
    - **Short polling (REST API)**. In this mode, Chat Server uses flex-disconnect-timeout which defines the maximum amount of time of protocol inactivity. As soon as the timeout expires, Chat Server removes the participant from a chat session. If this is the last participant, Chat Server closes the chat session.
    - **CometD only**. If a customer web application communicates with GMS over CometD, GMS subscribes to unsolicited notifications from Chat Server for this chat participant immediately after the chat session is successfully created by Chat Server. This request forces Chat Server to disable flex-disconnect-timeout for the chat participant and instead uses flex-push-timeout for the periodic querying of GMS to confirm that the participant is still connected over CometD. When GMS sends the confirmation, it tells Chat Server to consider the chat participant alive. As soon as GMS detects the disconnect over CometD, it sends an "unsubscribe" request, which forces Chat Server to enable flex-disconnect-timeout until the new subscribe request is sent by GMS to Chat Server (upon client re-connection over CometD to GMS).
    - **CometD and short polling with subscription for either mobile or custom-http push notification**. This mode operates almost exactly the same way as "CometD only" except that GMS never sends an "unsubscribe" request upon a CometD disconnect to Chat Server. This forces Chat Server to use only flex-push-timeout to ping GMS. In this mode, flex-disconnect-timeout is activated only when a client chat participant is removed from the chat session forcibly by another chat participant (such as an agent or bot).

Important

When Genesys agent desktops (WDE and WWE) receive the event indicating that a client left the chat session (for any reason), the agent desktop automatically sends the request to Chat Sever to close the chat session. A custom desktop can implement this differently if needed, as Chat Server keeps the chat session alive until the last participant leaves the chat session.

Inactivity control and chats session closure

We define chat session inactivity as the absence of a qualified event in a chat session for a certain period of time (defined by timeouts in configuration). A qualified event can be a message, a notice (as defined by async-idle-notices or include-notices), and a participant (the agent) joining or leaving the chat session. Only events with full visibility (in other words, visible to all participants) are taken into account here.

There are two complementary inactivity control configurations in Chat Server:

- Generic chat configuration (applicable both for async and regular chat sessions).
    - It is enabled:
        - If the option `enable` in section [inactivity-control] is set to `true`.
        - When both a customer and an agent (bots are not considered agents in idle control configuration) are present in the chat session. Once the last agent leaves the chat session, Chat Server disables this configuration.
    - After a certain period of inactivity (defined by option timeout-alert), Chat Server sends an alert notification (with text defined in message-alert option).
    - If there is still no activity (for the period defined by option timeout-alert2), Chat Server sends the second alert (with message defined in option message-alert2).
    - If there is still no activity (for the period defined by option timeout-close), Chat Server sends a "close" notification and immediately closes the chat session.
- Async only chat configuration (applicable only for async chat session):
    - Is enabled from the very start of an async chat session. It is activated independently of the presence of an agent in a chat session (in other words, it is activated even if you have only a customer in the chat session).
    - After a certain period of inactivity (defined by option async-idle-alert), Chat Server sends an alert notification (with text defined in the message-alert option).
    - If there is still no activity (for the period defined by option async-idle-close), Chat Server sends a "close" notification and immediately closes the chat session.

Important

Chat Server resets the inactivity period after any qualified event occurs in the chat session. Both inactivity configurations could be activated simultaneously.

Deploying High-Availability Chat Server

Overview

Chat Server can run in high availability (HA) mode where in the case of any Chat Server failure, chat sessions can be continued on other running Chat Server instances. Run Chat Server in load-balancing mode (also known as N+1) to enable HA mode. In load-balancing mode, configure Chat Server to run all instances in primary, or active, mode with no backup applications configured. **Note:** Primary and backup mode are still supported, but not recommended.

When running Chat Server in HA mode:

- The web chat application (for instance, Genesys Mobile Services or GMS) selects an active Chat Server instance to begin a new chat session. If the instance becomes unavailable during the course of the chat session, the web chat application selects another active Chat Server instance where the session will be restored and continued. At session restoration, Chat Server updates the interaction properties in Interaction Server with connection parameters that reflect the new location of the chat session.
- Agent Desktop watches for interaction property updates. After receiving the appropriate notification, Agent Desktop reconnects to the chat session at the specified Chat Server instance.

In HA mode, Chat Server stores the intermediate transcript after each submitted chat session message in persistent storage. This allows the chat session to be restored on a different Chat Server instance upon request. You can configure either of the following options to store the intermediate session transcripts:

- UCS – UCS configuration is simpler but creates an additional load on UCS and its database which the system can tolerate for small or medium sized deployments.
- Cassandra – Cassandra requires special configuration but removes the load from UCS which is beneficial to deployments with higher loads.

Note: In both cases, once the session has ended, the final transcript will be stored in UCS.

Configure Chat Server for HA

Configure as follows:

1. Configure and deploy the required number of Chat Server instances based on the expected load.
2. Connect Web API Server (GMS) to the *webapi* port of all Chat Server instances.
3. Connect Interaction Server to *ESP* port of all Chat Server instances.
4. Set the following values for options in the **settings** section for Chat Server applications:

- **session-restoration-mode** = `simple`

  This enables Chat Server's session restoration functionality.

  Note: The **session-restoration-mode** option has no effect unless the **transcript-auto-save** option is set to a valid positive value.

- **transcript-auto-save** = `1`

  This forces Chat Server to update the transcript in persistent storage (UCS or Cassandra) after each submitted message. You may also set this option to 2 (notify clients when the transcript is updated), however that would be effective only if the agent desktop can process special notifications from Chat Server (in particular, the notice **ucs-save-fail/save**). From the standpoint of resources, using the value 2 will slightly increase CPU usage; also Genesys Interaction Workspace does not support this functionality.

- **transcript-save-on-error** = `close`

  This forces Chat Server to close the chat session (without a final update in UCS) if, during the session, Chat Server detects a non-recoverable error or failure message when trying to store the intermediate chat session transcript.

5. Review the values for the following options (see the eServices Options Reference for full descriptions):
   - **transcript-resend-attempts**
   - **transcript-resend-delay**
   - **transcript-save-notices**

   The default values are acceptable for HA functionality; however you may wish to evaluate whether those values produce the behavior that you expect.

6. (Introduced in Chat Server version 8.5.312.10) To purge a chat session from the Chat Server instance that was processing that chat session before the restoration, set the configuration option session-restore-do-purge to `true`. While it should be used cautiously, it can be helpful in chat deployments where a chat session might accidentally be moved to another instance of Chat Server (for example by a malfunction in load balancing, or network issues). When enabled, and after a successful chat session restoration, Chat Server sends a purge request through Interaction Server to the Chat Server instance that was previously processing that chat session. That chat session is then completely removed from the Chat Server.
7. (Introduced in Chat Server version 8.5.316.02) If session-restore-extend-by is enabled (with any non-default value), you must:
   - Set session-restore-push-send to `true` if your solution is using GMS CometD Chat V2 or enables push notifications in Chat V2
   - Set a value for flex-push-on-join (Introduced in Chat Server version 8.5.315.05)
8. (Introduced in Chat Server version 8.5.316.02) If you believe the workflow might stop the interaction without closing the chat session, you may need to provide a non-empty value in ixn-submittedby-name. This forces Interaction Server to notify all instances of Chat Server when the interaction is stopped in the workflow. When this event is received, Chat Server closes the chat session. Otherwise, if the chat session is restored on a different Chat Server, it will not be closed.

Deploying Chat Server with Cassandra (Optional)

When Chat Server uses UCS to save intermediate transcripts it produces an additional load on the UCS database, especially for deployments with a high volume of customer chat interactions. To improve the performance of UCS and its database, Chat Server (starting with release 8.5.104) can use Cassandra for this functionality. With Cassandra, Chat Server requires UCS only to store the final chat transcript upon chat session completion.

Important

- There are no configuration options for Chat Server to enable Cassandra. Instead, the presence of the application connection to Cassandra RAP forces Chat Server to use Cassandra.
- Chat Server supports Cassandra only when Chat Server is deployed either on Windows or Linux.
- Chat Server, deployed with Cassandra, must run in the UTF-8 mode to support non-ASCII characters in chat conversations. **Note:** This is not required starting from Chat Server version 8.5.301.06

To facilitate the process above, the following steps must be completed after configuring Chat Server for HA:

- Deploy Cassandra.
- Initialize Cassandra for Chat Server.
- Connect Chat Server with Cassandra.

Deploy Cassandra

Officially, Chat Server supports Apache Cassandra 3.11, 4, and 5. Download the latest stable release of Cassandra here.

Important

For multi-node (cluster) Cassandra installation, use NTP (Network Time Protocol) to synchronize the clocks on all nodes.

Cassandra installation
For simple one-node Cassandra deployment:

1. Modify the **cassandra.yaml** configuration file:
   1. Set the value of **seeds**, **listen_address**, and **rpc_address** to the host IPv4 address.
   2. Make sure that Cassandra ports specified in the **cassandra.yaml** configuration file do not overlap with ports used by existing applications.
2. For load testing purposes, modify the **cassandra.yaml** configuration file:
   1. Set the value of **auto_snapshot** to `false`.

   2. Set the value of **compaction_throughput_mb_per_sec** to `0`.
   3. Set the value of **write_request_timeout_in_ms** to `10000`.
3. Start Cassandra node.

Initialize Cassandra for Chat Server

The initialization scripts are located in the *cassandra* sub-folder of the Chat Server installation folder. Before running the scripts using the Cassandra CQL Shell to create a new keyspace and the required tables, you must set the following values:

- Replication factor: In production, Genesys recommends a **replication_factor** of at least 3. The replication strategy should be set according to the cluster and datacenter configuration.
- Time-to-live: Occasionally, in case of failures some records in Cassandra are not be deleted by Chat Server. To setup a Cassandra self-cleanup procedure that will delete records after a certain time, select the appropriate script from the "Cassandra" sub-folder and set **default_time_to_live** and **gc_grace_seconds**.

Note: For version 8.5.104 of Chat Server, initialization scripts are not included with installation package. Scripts could be found here.

Connect Chat Server to Cassandra

To Connect Chat Server to Cassandra:

1. Create and configure a Chat Server Cassandra RAP application object based on the *ChatServerCassandraRAP* template provided in the installation package.
   1. Configure **Host** in the **Server Info** tab to point to one of the Cassandra cluster nodes. Configure the *default* port to the Cassandra cluster connection port and set the **Reconnect Timeout**.
      Note:
      - The **Reconnect Attempts** parameter in the **Server Info** tab is not used.
   2. In the **Options** tab, configure the **cassandra** section. Refer to the Options Reference Guide or the contents of ChatServerCassandraRAP.xml.
      Note:
      - In production, the recommended read and write consistency level is **quorum**.
      - If the **contact-points** option is not set, the Cassandra Cluster uses the **Host** defined in the **Server Info** tab as a contact point. If set, this option's value is used instead of the **Host** defined in the **Server Info** tab.
      - In order to configure authenticated access to Cassandra nodes, specify username and password.
        Note: You need to provide required configuration for Cassandra in the **cassandra.yaml** configuration file for **authenticator**.

2. Optionally, see Configure a secure connection to Cassandra for more information.
3. Add a connection from each Chat Server application to the newly created Chat Server Cassandra RAP.
4. Restart Chat Server.

Tip

In order to monitor Cassandra availability for Chat Server, configure alarms in management framework for log messages:

- 59520: Cassandra status changed to "UNAVAILABLE".
- 59521: Cassandra status changed to "AVAILABLE".

Run a Test

A properly configured solution with HA mode must work without any additional configuration for other components. This section describes a simple test.

Requirements:

- GMS and Widget CX
- Interaction Workspace (agent desktop)
- At least two running instances of Chat Server

Conduct the test as follows:

1. Start a chat session using Widget CX.
2. Send a message to verify that the chat session is active.
3. Then kill the Chat Server process with the ongoing chat session using Task Manager on Windows or `kill -9` on UNIX. GMS will then attempt to connect to another Chat Server instance where the chat session will be restored. At this point you will see a message showing that a user was disconnected and connected again.
4. Send a message to verify that the chat session continues.
5. Optionally, examine the Chat Server logs to see what actions were performed by the server to restore the chat session.

How to Upgrade Running Chat Solution

To upgrade Chat Server to a newer version in the N+1 deployment (also called load-balancing mode, when two or more instances are running in primary mode) without any service interruption the following steps are necessary for each Chat Server instance:

1. Make a certain instance of Chat Server unavailable for the creation of a new chat session (from GMS) by disabling Chat Server application in Configuration Server/Layer.
2. Wait until all current chat sessions are finished in this particular instance. This can be validated in Chat Server logs by the message "Int 59245 data: deleting session with sid=... and intx=.. (current sessions=0)". Or by requesting KPI counters via the web (REST) interface and validating that "session_created-sessoin_closed" is equal to zero.
3. Stop Chat Server. Replace Chat Server with a newer version (uninstall existing IP, install a new one). If needed, modify/update the Chat Server application. Enable the application in configuration. Start Chat Server.

If the load allows, several instances of Chat Server could be upgraded at the same time. Make sure to run enough Chat Server instances to handle the current load.

Chat Server could also be shutdown (not recommended) without being disabled and without waiting for current sessions to be finished. In this case, GMS moves chat sessions, running on this instance, to a different instances of Chat Server. This might result in short disruptions for those chat sessions (with additional participant left/added messages in chat session transcript).

Configuring a secure connection to Cassandra

The SSL communication mode between Chat Server and Cassandra nodes is optional and can be configured in the **[encryption]** section of the Chat Server Cassandra RAP object.

Important

If a shared Cassandra ring is used, the impact of your settings on other Cassandra-dependent components should be verified prior to making changes.

The following examples assume that:

- The Cassandra cluster consists of two nodes, node1 and node2, running on hosts with IP addresses 172.21.80.85 and 135.225.58.181.
- All passwords in this example are "genesys".
- The java *keytool* and *openssl* are available for certificate creation and manipulation.
- Only one Chat Server Cassandra RAP is configured for all Chat Servers in the solution, so relative paths to the certificates and keys should be the same on all Chat Server hosts. Should these paths be different, you can configure multiple Chat Server Cassandra RAP objects pointing to the same Cassandra cluster.

Example of certificates creation

Cassandra nodes certificate creation

Create a keystore and generate a node1 certificate.

```
keytool -genkeypair -noprompt -keyalg RSA -keysize 2048 -validity 36500 -alias node1 -keystore keystore1.jks -storepass genesys -keypass
genesys -dname "CN=172.21.80.85, O=Genesys, L=Daly City, ST=California, C=US"
```

Keystore **keystore1.jks** should be accessible by node1 and referred to in section **client_encryption_options** of the **cassandra.yaml** file in node1 configuration.

Create a keystore and generate a node2 certificate.

```
keytool -genkeypair -noprompt -keyalg RSA -keysize 2048 -validity 36500 -alias node2 -keystore keystore2.jks -storepass genesys -keypass
genesys -dname "CN=135.225.58.181, O=Genesys, L=Daly City, ST=California, C=US"
```

Keystore **keystore2.jks** should be accessible by node2 and referred to in section **client_encryption_options** of the **cassandra.yaml** file in node2 configuration.

Creating Client Certificates

Generate a client certificate with a private key.

```
openssl req -x509 -days 365 -subj "/C=US/ST=California/L=Daly City/CN=chatclient" -newkey rsa:2048 -keyout chatclientkey.pem -out
chatclient.pem
```

Copy both output files **chatclientkey.pem** and **chatclient.pem** into each Chat Server host and configure the **client-private-key-file** and **client-certificate-file** accordingly.

Exporting of Cassandra Node Certificates

Export node1 certificate:

```
keytool -exportcert -rfc -noprompt -alias node1 -keystore keystore1.jks -storepass genesys -file cassandra1.pem
```

Export node2 certificate:

```
keytool -exportcert -rfc -noprompt -alias node2 -keystore keystore2.jks -storepass genesys -file cassandra2.pem
```

Copy the exported node certificates, **cassandra1.pem** and **cassandra2.pem**, to each Chat Server host into the directory that is passed through each Chat Server Cassandra RAP object **trusted-cert-dir** option.

Importing Client Certificates

Import the client certificate into the truststore of node1:

```
keytool -import -file chatclient.pem -alias chatclient -keystore truststore1.jks -storepass
genesys
```

Import the client certificate of the truststore of node2:

```
keytool -import -file chatclient.pem -alias chatclient -keystore truststore2.jks -storepass
genesys
```

Cassandra and Java with Cryptography Extension

Cassandra nodes with client encryption enabled may fail to start unless Java is updated with the Java Cryptography Extension.

1. Download the Java Cryptography Extension (JCE) from Oracle's website.
2. Replace **US_export_policy.jar** and **local_policy.jar** in your JRE Java folder (found in: **\jre7\lib\security** for Windows or **/jre/lib/security/** for Linux-like platforms).
3. Restart Cassandra.

Client encryption with different Cassandra node certificates and client authentication

In **cassandra.yaml** of node1:

```
client_encryption_options:
  enabled: true
  keystore: <path-to-keystore>/keystore1.jks
  keystore_password: genesys ## The password you used when generating the keystore.
  truststore: <path-to-truststore>/truststore1.jks
  truststore_password: genesys ## The password you used when generating the truststore.
  require_client_auth: true
```

In **cassandra.yaml** of node2:

```
client_encryption_options:
  enabled: true
  keystore: <path-to-keystore>/keystore2.jks
  keystore_password: genesys ## The password you used when generating the keystore.
  truststore: <path-to-truststore>/truststore2.jks
  truststore_password: genesys ## The password you used when generating the truststore.
  require_client_auth: true
```

Cassandra RAP, section encryption:

```
enbabled=true
trusted-cert-dir=<Path to directory containing cassandra1.pem and cassandra2.pem. The
chatclientkey.pem file should not be placed into this directory.>
client-private-key-file=chatclientkey.pem
```

```
password=genesys ## openssl will prompt for this password to be entered during the
certificate creation
client-certificate-file=chatclient.pem
verify-peer-cert=true
verify-peer-identity=true
```

Using cqlsh with SSL encryption

Use the following directions to configure the **cqlshrc** configuration file.The following examples assume that all relevant .pem files are copied into the local C:\certs\ directory.

1. Copy **cqlshrc.sample** from the ~/conf directory to another location, for example **C:\certs\ directory**.
2. Rename the file to **cqlshrc.conf**.
3. Modify the following sections to be consistent with the encryption configuration shown above:

```
[authentication]
;username = fred
;password = !!bang!!$
;; We assumed no user name or password is set in the Cassandra example

[cql]
version = 3.2.0
;; it would not connect with lower version

[connection]
hostname = 172.21.80.85
;; this is node1 of our example
port = 9042
;; we assume the port is default
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
certfile = C:\certs\cassandra1.pem
;; the certificate of node 1
validate = true
;; assume that we want to validate the node, optional
userkey = C:\certs\chatclientkey.pem
;;if client auth is required on cassandra
usercert = C:\certs\chatclient.pem
;;if client auth is required on cassandra

[certfiles]
172.21.80.85 = C:\certs\cassandra1.pem
;; the cert for node1
135.225.58.181 = C:\certs\cassandra2.pem
;; the cert for node2
```

Start cqlsh with the following command:

```
cqlsh --ssl --cqlshrc=C:\certs\cqlshrc.conf
```

Cqlsh shell should connect to node1 using the configured SSL.

Client encryption with a single Cassandra node certificate and client authentication

In **cassandra.yaml** of node1:

```
client_encryption_options:
  enabled: true
  keystore: <path-to-keystore>/keystore1.jks
  keystore_password: genesys ## The password you used when generating the keystore.
  truststore: <path-to-truststore>/truststore1.jks
  truststore_password: genesys ## The password you used when generating the truststore.
  require_client_auth: true
```

In **cassandra.yaml** of node2:

```
client_encryption_options:
  enabled: true
  keystore: <path-to-keystore>/keystore1.jks
  keystore_password: genesys ## The password you used when generating the keystore.
  truststore: <path-to-truststore>/truststore2.jks
  truststore_password: genesys ## The password you used when generating the truststore.
  require_client_auth: true
```

Cassandra RAP, section encryption

```
enbabled=true
trusted-cert-dir=<Path to directory containing cassandra1.pem. The chatclientkey.pem file
should not be placed into this directory.>
client-private-key-file=chatclientkey.pem
password=genesys ## openssl will prompt for this password to be entered during the
certificate creation
client-certificate-file=chatclient.pem
verify-peer-cert=true
verify-peer-identity=false
```

Client encryption with a single Cassandra node certificate and no client authentication

In **cassandra.yaml** of node1:

```
client_encryption_options:
  enabled: true
  keystore: <path-to-keystore>/keystore1.jks
  keystore_password: genesys ## The password you used when generating the keystore.
  truststore: <path-to-truststore>/truststore1.jks
  truststore_password: genesys ## The password you used when generating the truststore.
  require_client_auth: false
```

In **cassandra.yaml** of node2:

```
client_encryption_options:
  enabled: true
  keystore: <path-to-keystore>/keystore1.jks
  keystore_password: genesys ## The password you used when generating the keystore.
  truststore: <path-to-truststore>/truststore2.jks
  truststore_password: genesys ## The password you used when generating the truststore.
  require_client_auth: false
```

Cassandra RAP, section encryption

```
enbabled=true
trusted-cert-dir=<Path to directory containing cassandra1.pem>
client-private-key-file=
password= ## empty
client-certificate-file=
verify-peer-cert=true
verify-peer-identity=false
```

ECDHE Cipher Suite Support

When the Java version used does not support ECDHE cipher suite, the cipher_suites option of the **client_encryption_options** section in **cassandra.yaml** file must be modified to exclude cipher suites prefixed with **TLS_ECDHE_**. For example:

```
cipher_suites:
[TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA]
 #,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA]
```

Initialization Cassandra scripts for Chat Server

Note: The content of this page is only applicable for version 8.5.104 of Chat Server. For later versions, starting with 8.5.105, initialization scripts are included in the installation package in the subfolder *cassandra*.

Create keyspace

```
CREATE KEYSPACE genesys_chat_server
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };

USE genesys_chat_server;
```

Create tables without a cleanup procedure

```
CREATE TABLE transcripts (
   id text,
   creatorid text,
   transcript text,
   PRIMARY KEY (id, creatorid) )
WITH CLUSTERING ORDER BY (creatorid ASC);

Table owner
CREATE TABLE owner (
   id text,
   ownership_start_time timeuuid,
   creatorid text,
   PRIMARY KEY (id, ownership_start_time) )
WITH CLUSTERING ORDER BY (ownership_start_time DESC);
```

Create tables with a cleanup procedure

You can specify **default_time_to_live** and **gc_grace_seconds**, as shown in the following example, where 1209600 is the number of seconds in two weeks:

```
CREATE TABLE transcripts (
   id text,
   creatorid text,
   transcript text,
   PRIMARY KEY (id, creatorid) )
WITH CLUSTERING ORDER BY (creatorid ASC)
AND default_time_to_live = 1209600 AND gc_grace_seconds = 1209600;

Table owner
CREATE TABLE owner (
   id text,
   ownership_start_time timeuuid,
   creatorid text,
   PRIMARY KEY (id, ownership_start_time) )
WITH CLUSTERING ORDER BY (ownership_start_time DESC)
```

```
AND default_time_to_live = 1209600 AND gc_grace_seconds = 1209600;
```

For more information, see Deploying Chat Server with Cassandra.

Multilingual Processing in Chat Server

Overview

Genesys Chat can process multiple languages simultaneously, including:

- Chat transcript messages. The data is transferred in UTF-16.
- Attached data (such as first name, last name, subject, and so on) and ESP messages (submitting messages to chat session from the strategy). The data is transferred in UTF-8. There are the following requirements:
    - Chat Server must be running with UTF-8 locale; details are in "Internal Locale of Chat Server" below.
    - If UCS is running on Windows, its startup script (`ContactServerDriver.ini`) must be configured to use `-Dfile.encoding=UTF-8.` If UCS is running on UNIX, no special configuration is required.
    - Routing strategies must send data in UTF-8 encoding.
- Chat Server can be configured to send inactivity system messages in different languages.

Internal Locale of Chat Server

The encoding and locale that Chat Server uses internally are determined by the following, in order of priority:

1. The command line parameter `-codepage.` The value of the parameter must be a valid and enabled encoding name. To use UTF-8 on Windows platform the value must be `UTF-8.`
2. Connection to a Configuration Server that is running in multi-language mode, which sets Chat Server's internal locale to UTF-8.
3. The current system locale.

Sending Chat Messages from Routing Workflows

To send messages in different languages to a chat session from a workflow the following is required:

- Chat Server runs in UTF-8 mode.
- Workflow (strategies) must be created:
    - Either by Interaction Routing Designer of version 8.1.400.10 or later (if URS is used).
    - Or by any version of Composer (if ORS is used).

Masking Sensitive Data

Chat Server logs and chat transcripts might contain sensitive data such as credit card numbers, phone numbers, Social Security numbers, and so on. You can omit this data from logs and mask it in transcripts.

Logs

To omit sensitive data from logs, you must configure both UCS and Chat Server, as follows:

- In the **[settings]** section, set message-log-print-size to `0`. This means that logs do not show the messages sent between chat participants. Where a message occurs, the log shows `[truncated from size=x]`, where *x* is the number of characters in the suppressed message.
- In the **[log-filter-data]** section,
  - Set StructuredText to `hide` so that logs will omit the transcript that UCS sends to Chat Server.
  - Set Transcript to `hide` so that logs will omit the transcript that Chat Server sends to UCS.

Chat Transcripts

Overview

Chat Server can mask sensitive data in messages during chat sessions and in saved transcripts by using a regular expression (regex) to find and substitute the data with a configurable replacement character. Regular expressions, specified for Chat Server, must use the same syntax and semantics as defined for Perl 5 (however, Privacy Manager imposes additional constrain by allowing only java.util.regex compatible expressions). When enabled this functionality will:

- Examine each chat message with an ordered set of regex rules. Use the apply-config option to configure the source/location of regex rules that will be applied. Note: all options are located in the `transcript-cleanup` section.
- Replace any part of the message that matches a regex rule with a replacement character specified by the configuration. The default is specified by the default-repchar option.
- When replacing symbols you can choose to replace all symbols or only digits. When replacing digits, you can also leave the last few digits unmasked —see the default-spec option.

This functionality can be applied for the messages of an ongoing chat session and/or a transcript saved in the contact history (UCS). This is specified by the apply-area option.

Tip

- Starting with release 8.5.103, Chat Server supports reading the regex rules from UCS. To do this,
    - Set the apply-config option to `mix` or `ucs`.
    - Use Privacy Manager, a plugin for Genesys Administrator Extension (GAX), to select and activate these rules.
- Prior to release 8.5.103, Chat Server used different options from the `settings` section for this functionality. Click here to view the previous description.

Deployment steps for Personally Identifiable Information (PII) cleanup

To deploy PII cleanup, set apply-area to `always` (or to a different value based on your needs; see table below) in Chat Server **transcript-cleanup** options and, if needed, adjust any other options in section [transcript-cleanup].

By default only hardcoded PII rules are used (as described in Default Rules if No Configuration is Provided). However, if you need to specify your own set of rules, you need to install Privacy Manager , a plugin for Genesys Administrator Extension (GAX), which brings a set of the same default PII rules into UCS, which you can then modify and extend with your own rules.

PII cleanup is applied for:

Option apply-area value	Messages exchanged between a customer and an agent	Chat session transcript stored in UCS (contact history)
never	Unmasked	Unmasked
always	Masked	Masked
history-all or history-final	Unmasked	Masked

Unmasking Data for Active Agents

Starting with release 8.5.106, Chat Server allows to unmask (i.e. suppress masking) for sensitive data in messages from a customer. It is controlled by the settings of the unmask-live-dialog configuration option. Unmasking is applicable only in the presence of active (visible to customer) agents. Coaching and monitoring agents will not see unmasked data. For active agents, only the data sent after the agent joined the session is unmasked.

Example

Consider the following scenario (assuming that the rule for masking credit cards is enabled):

1. The customer initiates a chat session. Without waiting for an agent, the customer sends the credit card number in a message. The credit card number is masked out.

2. An agent joins the session. This agent sees the chat session transcript from the very beginning where the credit card number is masked out in the message from the customer.
3. The customer sends the credit card number again. Both, the agent and the customer, see it.
4. A second agent joins the chat session for a conference conversation. The second agent sees the chat session transcript from the very beginning where the credit card number is masked out in all messages.
5. The customer sends the credit card number again. Now both agents and the customer see it.

After this chat session is finished, the transcript, saved in the contact history (UCS), has all credit card numbers masked out.

Default Rules if No Configuration is Provided

If the apply-config option has a value of `cfg` or is set to `mix` and no UCS PII configuration has been provided for the given chat session, Chat Server uses the following default rules to find sensitive data:

Order	Name	Regular Expression
1. Credit card	GCTI_CreditCards	(?>^|(?<=[\s[:alpha:](),.:;?!"'`]))(?>4\d{3}|5[1-5]\d{2}|6011|622[1-9]|64[4-9]\d|65\d{2})[ -.]?\d{4}[ -.]?\d{4}[ -.]?\d{4}(?>$|(?=[\s[:alpha:](),.:;?!"'`]))
2. Social Security number	GCTI_SSN	(?>^|(?<=[\s[:alpha:](),.:;?!"'`]))(?!000|666|9)\d ]?(?!00)\d{2}[- ]?(?!0000)\d{4}(?>$|(?=[\s[:alpha:](),.:;?!"'`]))
3. Phone number using the North American Numbering Plan	GCTI_PhoneNANPA	(?>^|(?<=[\s[:alpha:](),.:;?!"'`]))(?:\+?1[- . ]?)?(?:\(?[2-9][0-9]{2}\)?[-. ]?)?[2-9][0-9]{2}[-. ]?[0-9]{4}(?>$|(?=[\s[:alpha:](),.:;?!"'`]))

Typing Preview

Typing preview allows an agent to see text that a customer types before the text is submitted to the chat session. You can have Chat Server mask all digits in the typing preview by setting the typing-preview (called transcript-cleanup-typing before release 8.5.103) option to a value other than none. Chat Server then replaces all digits in the typing preview with the character specified by default-repchar (called transcript-cleanup-mask before release 8.5.103).

Inactivity Monitoring

Overview

Inactivity monitoring is a Chat Server functionality that allows closing a chat session if there is no activity by chat participants after a certain period of time.

Inactivity monitoring is enabled:

- For all chat sessions – by setting the enabled option in `inactivity-control` to `true`.
- For async only chat sessions – see Inactivity control and chats session closure for more information.

If inactivity monitoring is enabled in the `inactivity-control` section, it works as following:

- Chat Server activates inactivity monitoring only if at least one customer and one agent (bots are not considered agents in idle control configuration) are participating in the chat session. If the inactivity monitoring is activated in a chat session, then:
- If there is no activity during the time specified by the timeout-alert option, Chat Server issues a warning comprising the text specified by the message-alert option.
- If there is no activity for another timeout-alert2 seconds, Chat Server issues a warning comprising the text specified by the message-alert2 option. Note, that timeout-alert2 is activated only if the option value is greater then zero, otherwise the timeout-close is activated next.
- If there is no activity for another timeout-close seconds, Chat Server issues a notification consisting of the text specified by the message-close option and closes the chat session (and removes all participants from it).
- To suppress sending a message when any of the timeouts expire, set the corresponding `message-xxx` to the empty value. The empty message value does not disable the timeout itself.
- If any activity occurs, Chat Server resets the current timer and reactivates the `timeout-alert` timer. *Activity* means any activity in the chat session that is visible to all participants—so, for example, coaching messages between agents do not count as chat activity.

Inactivity monitoring control is supported for the following components:

Component	Minimum Supported Version	Configuration
Chat Server	8.5.104.08	Disabled by default. Configured in the **[inactivity-control]** section.
Genesys Mobile Services	8.5.106.14	No special configuration needed.
Chat Widget	9.0.000.08	No client-side configuration needed.
Workspace Desktop Edition	8.5.109.25	No special configuration needed
Workspace Web Edition	not supported	n/a

If a component that does not support this feature is deployed in solution, inactivity monitoring control must be disabled in the Chat Server options to avoid chat session closure without notifying all current participants.

Configuration per Session from Workflow

There is a possibility to set a different inactivity control configuration for different chat sessions. In order to facilitate it, the workflow (i.e. ORS/URS strategy) must send the IdleControlConfigure ESP request. Upon receiving such request for an ongoing chat session Chat Server:

- Modifies inactivity control parameters for a given chat session.
- Resets current inactivity control timers if any are currently enabled.

Localization of System Inactivity Messages

Chat Server can be configured to send inactivity system messages in different languages.

Important
This functionality is supported only when Chat Server is configured with a single tenant.

How to Configure Languages

A language must be configured as `Attribute Values` of the Language in `Business Attributes`. An arbitrary number of languages with arbitrary names can be created.

Tip
Each attribute (language) has a name and a display name which can be different. Chat Server uses the attribute name and not the display name for this functionality.

Each attribute (language) can contain the following options in the Annex:

Section	Option	Mandatory	Possible values	Notes
code	`language`	optional	ISO 639 code	The value is converted to lowercase when read from configuration.

Section	Option	Mandatory	Possible values	Notes
code	`country`	optional	ISO 3166 code	Only used if the `language` option is specified. The value is converted to uppercase when read from configuration.
code	`use-language-as-default`	optional	true / false	See How Chat Server Associates Sessions with Languages.
chat-server	`message-alert`	optional	any string (can be empty)	If specified, overrides the message-alert option's value for this language.
chat-server	`message-alert2`	optional	any string (can be empty)	If specified, overrides the message-alert2 option's value for this language.
chat-server	`message-close`	optional	any string (can be empty)	If specified, overrides the message-close option's value for this language.

How Chat Server Associates Sessions with Languages

Each chat session in Chat Server can be associated with a language, configured as a business attribute. For each chat session Chat Server is looking for two special key-value pairs in the initial `UserData`:

- `GCTI_LanguageName`. If it is present in the `UserData`, only this parameter is used. It must contain the name of the language business attribute. If such language business attribute does not exist, the configuration from Chat Server options is used for this session.
- `GCTI_LanguageCode`. Only if `GCTI_LanguageName` is not present in the `UserData`, then `GCTI_LanguageCode` is checked. It must contain code in the `language-country` format (or `language_country` for backward compatibility). Chat Server parses this code into ISO language (value is converted to lowercase) and ISO country (value is converted to uppercase). Then Chat Server is trying to find the appropriate business attribute for this session as follows:
    1. The business attribute with exactly the same language and country, specified in the code section. If not found,
    2. The attribute with the same language and empty (or not specified) country. If not found,
    3. The attribute with the same language and any country code specified, however only among attributes which contain the `use-language-as-default=true` option.

How to Change Chat Session Language

A session language can be changed during the course of a chat session by:

- Sending a request from the workflow.
- Sending a system notice from the chat wigdet or an agent desktop with the `configure-session` action and user data with either `GCTI_LanguageName` or `GCTI_LanguageCode`.

Contact Identification for Regular and Anonymous Chat

Whenever a new chat session is being created, Chat Server communicates with UCS to create a chat interaction record where the chat transcript of a completed chat session is stored as a part of the contact history. Each interaction record can be associated with a contact record that identifies and describes a customer profile. Such an association is done automatically by UCS during the interaction record creation, based on the presence of "contact attributes" in the attached data.

Contact attributes can be comprised of a default set of attributes, which are defined for each interaction type (chat, email, voice). Additionally, or alternatively, custom contact attributes can be configured. You can find more information on this on the Contact Identification page, in the eServices Administrator's Guide.

Anonymous chat recommendations

If a large volume of anonymous chats is expected, a solution must be configured to avoid an association of all anonymous interactions with a single contact record (which often leads to performance implications). There are two alternative approaches to achieve this:

- Recommended: no default or custom contact attributes are provided when starting the chat session. See below on how to achieve this with the Genesys Chat Widget or through the GMS API.
- Alternative: If for some reason, there is a need to associate every chat record with a contact record, then a special custom attribute must be defined (for example: `ExternalCustomerId`), and it must be configured either as the only contact attribute for the media type, or it must be set with a highest priority if several contact attributes are defined (see Customize Contact Identification per Media Type for directions on how to configure this). The value for this attribute can be assigned either arbitrarily (for example, a new UUID each time) or with values coming from a customer CRM solution.

Important

Genesys strongly recommends not using First and Last name with arbitrary values for this purpose as it can lead to performance implications due to the specifics of the contact identification algorithm. `EmailAddress` should not be used either with random values as the email address can potentially be used later by other components (in other words, when sending the transcript email).

The list of contact attributes for a new chat session can come through the GMS API in Request Chat from:

- Genesys Chat Widget. In this case, the Widget must be launched without asking for contact attributes, but instead by only sending a nickname (which is not being used for contact identification, but is required by GMS API if first/last name is absent). The nickname can either be hard-coded into the Widget or requested on the initial Widget screen. For example, from "launcher.html" this can be done by adding the following to "webchat" under the "Additional Configuration Options" section:

```
{
 "form":{
   "wrapper":"<table></table>",
     "inputs":[
      {
       "id":"cx_webchat_form_nickame",
       "name":"nickname",
       "maxlength":"100",
       "placeholder":"mandatory",
       "label":"Nick Name"
      },
      {
       "id":"cx_webchat_form_subject",
       "name":"subject",
       "maxlength":"100",
       "placeholder":"@i18n:webchat.ChatFormPlaceholderSubject",
       "label":"@i18n:webchat.ChatFormSubject"
     }
   ]
 }
}
```

This defines the nickname instead of First and Last name. Alternatively, the subject can also be removed completely from the initial form. For more information about Widget configuration, see Customizable Chat Registration Form.

- A custom web chat application. In this case, only the "nickname" must be provided in "Request Chat", while `firstName`, `lastName`, and `emailAddress` must not be sent.

How contact identification works for chat

If the user data includes an attribute called `EmailAddress`, UCS looks for a contact in its database whose `EmailAddress` attribute has the same value as the user data attribute. The name of the user data attribute must be exactly `EmailAddress` —if it is `email_address` or anything else, UCS will not try to match its value with the stored value of `EmailAddress`. Or, if UCS finds no matching contact, it creates a new one using the user data.

For either a matching contact or a new one, UCS sends the following, as data about the contact for this interaction, to Chat Server:

- The matched attribute (if not email address, then phone number, and so on).
- The attribute `ContactID`.
- All other attributes of this contact that UCS has stored in its database, except:
- If any user data has an attribute name that matches an attribute name in the UCS `Contacts` table, UCS returns the value of the attribute from the user data, not the value from the `Contacts` table. It does not modify the value in the `Contacts` table.

The last point can cause a problem, as in the following example:

1. Home user Steve Jones wants to open a chat session. In the web interface, he types in his correct email address sjones@here, then erroneously types his first name as Speve.
2. UCS finds a contact record for sjones@here.

3. UCS returns to Chat Server data about an existing contact whose email address is sjones@here and whose first name is Speve. UCS still has the correct first name Steve in its database, but the user data, with the erroneous Speve, preempts the correct data for the purposes of this chat interaction.
4. The system uses the user data to generate the message prompt that marks the home user in the chat display. As a result, the chat session displays something like the following:
   ```
   14:52:20 SpeveJ has joined the session
   14:52:30 SpeveJ > Hi.
   ```
5. The Agent Desktop displays the incorrect first name (in the user data on the lower left pane) and the correct first name (on the Customer Records pane on the right). The agent sees the incorrect first name and opens the chat session by typing, “Hello Speve, how can I help you?”
6. The interaction passes through a strategy that generates an automatic response, which opens, “It was good chatting with you, Speve.”

To avoid this type of problem, be sure that the system (including strategies and desktop) as well as its users refer to the UCS database, rather than user data, for contact attributes. In the example just cited, the agent must be sure to look at the Customer Records (right-hand) pane of the Desktop for the name of the contact.

It is also advisable to closely monitor the inventory of contact attributes that can become user data.

How to send ESP requests to Chat Session from Workflow

Introduction

Genesys can send messages, notices (types are limited), and other requests to a chat session from a workflow (an URS [Universal Routing Server]/ORS [Orchestration Server] strategy).

For example, when a customer starts a chat session from the web page, the chat session is created in Chat Server and corresponding interaction is submitted in Interaction Server. At some point, the interaction is processed by the workflow, which can send a message like "agent will be with you shortly... " and then the routing starts (to find an agent to serve this chat communication).

Prerequisites

Interaction Server application (in configuration) must be connected to Chat Server application's "ESP" port.

How to Implement

The following steps are necessary in order to send a message or notice from the URS strategy:

1. Verify that the interaction is still online by checking that `UData['IsOnline'] != '0'`. If the interaction is offline, which means that the chat session is closed, there is no sense to send messages into it.
2. Extract from the interaction properties the name of the Chat Server application which is processing/handling the ongoing chat session. This can be achieved by assigning `UData['ChatServerAppName']` to a local variable.
3. Use the `External Service` block in the `Data and Services` palette in IR Designer (or the `External Service` block in the `Server Side` palette in Composer) to send a request. The following general parameters must be specified:
   - The Application type must be set to `ChatServer`.
   - The Application name must be set to a value obtained from the user data in step 2.
   - The Service name is set to `Chat`.
   - The `Don't send user data` must be unchecked.
4. Set the corresponding Method name to send one of the ESP requests, described below.
5. Provide mandatory parameters.

Tip

Each ESP request listed below requires the interaction ID to process the request. Chat Server receives the value for the interaction ID from the userdata. This userdata is supplied with each ESP request of the KVP with the **InteractionId** key. However, this behavior can be overwritten if an explicit parameter in each ESP request with the same **InteractionId** name is provided.

Example: In order for the workflow to process internal communication interactions used to invite another agent for a conference or a consult through the queue, the skills, or the agent group, the **InteractionId** parameter must be initialized with the value of the parent chat interaction `UData['ParentId']`.

Available in Chat Server since 8.5.314.02.

Important

Chat Server processes values for all incoming parameters as a string value, even if it represents a number or a Boolean value. For Boolean parameters, only `true` is recognized as a value for **true**. All other values are recognized as **false**.

Method **Message**

`Message` – Submits a text message to a chat session. Provide the following parameters:

Parameter	Mandatory	Value Description
MessageText	yes	Message text to submit to a chat session
MessageType	optional	Specifies any arbitrary text as message type (transparent for Chat Server).
Nickname	optional	Specifies the nick name of a participant on behalf of whom the message will be shown in a chat session.
Visibility	optional	Possible values: • ALL – message is visible to all chat participants (default value) • INT – message is visible to agents and supervisors only • VIP – message ise visible to supervisors only Use `visibility` wisely as not all components (including Genesys Workspace) may show it correctly.

Parameter	Mandatory	Value Description
EventAttributes	optional	Specifies a nested list of attributes which are associated with the message provided. For more information, see Key-value collection format specification below with its Example. Available in Chat Server since 8.5.201.05.

Event3rdServerResponse

Parameter	Value type	Mandatory	Value Description
OccuredAt	String	Yes	Timestamp for when the method was processed.
ScriptPos	Integer	Yes	Position of the message submitted in the chat transcript. Positioning starts from `0`.

Method **Notice**

`Notice` – Sends a notification of the specified type to a chat session. Provide the following parameters:

Parameter	Mandatory	Value Description
NoticeType	yes	Possible values: • `USER_PUSHED_URL` – to implement the "push URL" functionality (`NoticeText` must contain valid URL). • `USER_CUSTOM` – can be used for any custom purpose (completely transparent for Chat Server).
NoticeText	optional	Any arbitrary text.
Nickname	optional	Specifies the nick name of a participant on behalf of whom the message will be shown in a chat session.
Visibility	optional	Possible values: • `ALL` – message is visible to all chat participants (default value) • `INT` – message ise visible to agents and supervisors only

Parameter	Mandatory	Value Description
		• `VIP` – message is visible to supervisors only Use `visibility` wisely as not all components (including Genesys Workspace) may show it correctly.
EventAttributes	optional	Specifies a nested list of attributes which are associated with the notice provided. For more information, see Key-value collection format specification below. Available in Chat Server since 8.5.201.05.

Event3rdServerResponse

Parameter	Value type	Mandatory	Value Description
OccuredAt	String	Yes	Timestamp for when the method was processed.
ScriptPos	Integer	Yes	Position of the message submitted in the chat transcript. Positioning starts from `0`.

Method **PlaceOnHold**

`PlaceOnHold` – Places the chat session on hold which sets `GCTI_Chat_AsyncStatus` to `-2`.

Important

This is only applicable for async chat sessions.

After calling this ESP method, the workflow must place the interaction into the **waiting** queue (in Chat Business Process Sample this is done through `async-chat-return-queue`). `GCTI_Chat_AsyncStatus` must not be set by the workflow itself since the Chat Server does not send a `GCTI_Chat_AsyncStatus` update upon the arrival of a new message from a customer, if Chat Server was not explicitly notified through this ESP method.

Request3rdServer

No parameters required.

Event3rdServerResponse

Parameter	Value type	Mandatory	Value Description
OccuredAt	String	Yes	Timestamp for when the method was processed.

Method **CloseSession**

`CloseSession` – Closes an alive session; Chat Server:

- Notifies active participants
- Updates the UCS record with the final transcript and chat closed date
- Updates `IsOnline=0` and stat metrics (if configured) in Interaction Server
- Marks the chat session as Closed internally (but does not immediately remove it from memory, allowing a customer time to request the last transcript fetch).

Request3rdServer

Parameter	Value type	Mandatory	Value Description
CloseIfNoAgents	String	Optional	Possible values: • `true` – Session closes only when there are no agents. • `false` (default) – Session closes even if participants are present in the session.
Purge	String	Optional	When this value is `true`, Chat Server removes the chat session from its memory without executing any closing actions (such as notifying participants, or updating the UCS and Interaction Server). This request is used internally when `session-restore-do-purge=true`. Available in Chat Server since 8.5.312.10.

Event3rdServerResponse

Parameter	Value type	Mandatory	Value Description
OccuredAt	String	Yes	Timestamp for when the method was processed.
IsClosed	Integer	Yes	Possible Values: • 1 (Session closed) • `0` (Session alive)

Method **GetSessionInfo**

`GetSessionInfo` – Returns information regarding the session.

Request3rdServer

No parameters required.

Event3rdServerResponse

Parameter	Value type	Mandatory	Value Description
OccuredAt	String	Yes	Timestamp for when the method was processed.
SessionInfo	Key-value list	Yes	Lists the following info • `CreatedAt` – Timestamp for when the session was created. • `IsRestored` – Specifies whether the session was restored. Possible values : `0` (false) or `1` (true).

Method **IdleControlConfigure**

`IdleControlConfigure` – allows to change the configuration for inactivity control monitoring for a given chat session. Provide the following parameters (while all parameters are optional, at least one parameter must be provided):

Parameter	Mandatory	Notes
reset-parameters	optional	Resets all inactivity control parameters to values provided in the Chat Server application configuration.

Parameter	Mandatory	Notes
		Valid values: true / false (default).
enabled	optional	
include-notices	optional	
message-alert	optional	
message-alert2	optional	Available starting with Chat Server release 8.5.107.11
message-close	optional	
timeout-alert	optional	
timeout-alert2	optional	Available starting with Chat Server release 8.5.107.11
timeout-close	optional	

Event3rdServerResponse

Parameter	Value type	Mandatory	Value Description
OccuredAt	String	Yes	Timestamp for when the method was processed.

Method **ConfigureSession**

`ConfigureSession` - allows you to change the language for the current chat session. At least one of the following parameters must be included:

Parameter	Mandatory	Value Description
async-idle-reset	optional	If the value 1 is provided, Chat Server resets async idle timeouts (in other words, starts counting from zero).
GCTI_LanguageCode	optional	If this parameter is present, other parameters are ignored . The parameter must contain the name of the language business attribute.
GCTI_LanguageName	optional	This parameter is used only if the `GCTI_LanguageCode` parameter is not present. Parameters are processed as described in the How Chat Server Associates Sessions with Languages section.

Event3rdServerResponse

Parameter	Value type	Mandatory	Value Description
OccuredAt	String	Yes	Timestamp for when the method was processed.

Key-value collection format specification

In order to provide data of type `key-value collection` in requests from URS workflow, the following parameters must be included in Interaction Routing Designer:

- The parameter Name should be prefixed with {l} (lowercase L in curly brackets). This triggers URS to process the parameter value as a nesting list of key-value pairs.
- The parameter value can be specified in either of the following formats (Note: no line breaks allowed and whitespaces are ignored):
    - JSON-like style (can only be provided through the variable)

      ```
      {"key1":"value1","key2":"value2","key3":{"subkey1":"subvalue1"}}
      ```

    - List-like style (can be provided directly in the parameter)

      ```
      key1:value1|key2:value2|key3.subkey1:subvalue1
      ```

Example

In order to send a Rich Text message with Quick Replies, the following string must be provided as a value for the `EventAttributes` parameter in the Method `Message`:

```
{"structured-content":{"genesys-
chat":{"content":"{\"type\":\"Message\",\"contentType\":\"quick-
replies\",\"content\":[{\"id\":1,\"type\":\"quick-
reply\",\"action\":\"message\",\"text\":\"Black\"},{\"id\":2,\"type\":\"quick-
reply\",\"action\":\"message\",\"text\":\"Green\"},{\"id\":3,\"type\":\"quick-
reply\",\"action\":\"message\",\"text\":\"Mint\"}]}", "type":"Generic"}}}
```

It is important that the actual Rich Text element body JSON (specified in `content`) is escaped so it will be treated as a value instead of being converted into a nested Key/Value list structure.

Integrating Chat Server with Genesys Historical Reporting

Chat session reporting relies on Interaction Server reporting events to provide session-related data to the products that enable Genesys historical reporting:

- Interaction Concentrator, which comprises the Interaction Concentrator (ICON) server and Interaction Database (IDB) -- Stores detailed reporting data from Interaction Server and other sources.
- Genesys Info Mart -- Extracts, transforms, and loads (ETLs) data from IDB into the Info Mart database, a data mart suitable for contact center reporting.
- Reporting and Analytics Aggregates (RAA) -- Aggregates Info Mart data to provide contact center activity metrics for downstream reporting applications.
- Genesys Customer Experience Insights (GCXI) -- Extracts aggregated data from the Info Mart database and presents it in readable historical reports.

This page describes the component and configuration requirements to enable historical reporting on chat session activity in your deployment.

Important

For information on Asynchronous (Async) Chat, see the Asynchronous Chat section in this guide.

Overview

1. After a chat session is finished, Chat Server attaches reporting statistics to the user data of the interaction in Interaction Server. For more information about the attached user data key-value pairs (KVPs), see Chat Server reporting data.
2. ICON stores the user data in the G_USER_DATA_HISTORY table in IDB in near real-time.
3. On a regular schedule, Genesys Info Mart extracts the IDB data and transforms it into the CHAT_SESSION_FACT table and supporting dimensions in the Info Mart dimensional model. For more information about the session-related tables in the Info Mart database, see the *Genesys Info Mart Physical Data Model* for your RDBMS. For more information about managing the Genesys Info Mart ETL jobs, see the *Genesys Info Mart Operations Guide*.
4. RAA summarizes and organizes the Info Mart data in ways that enable GCXI to extract meaning. For more information about RAA data, see the *RAA User's Guide*.
5. GCXI uses the aggregated data in the Info Mart database to present out-of-box chat session reports, including:
   - Chat Message Statistics Report

- Chat Termination Report
- Async Chat Dashboard

6. In deployments that include Bot Gateway Server (BGS) starting with version 8.5.203.09, there are reports and dashboards on bot-related activity. **Note:** BGS is currently only available as a restricted release.
7. In deployments where Async Chat functionality is enabled, the Async chat dashboard displays async chat metrics.
8. For more information about the GCXI reports, see Chat reports in the GCXI *User's Guide*.

Enabling historical reporting on chat session activity

Prerequisites

The following table summarizes the minimum release requirements for the components that enable chat session historical reporting.

Component	Minimum release for chat	Minimum release for async chat
Chat Server	8.5.203.09 (restricted release) 8.5.301.06 (general release)	8.5.302.03 8.5.315.xx (for intermediate updates)
ICON	8.1.514.11	8.1.514.11
Genesys Info Mart	8.5.011.04	8.5.011.14 8.5.116.20 (for intermediate updates)
RAA	8.5.003	8.5.006
GCXI	9.0.005	9.0.007
WDE	No minimum requirement	8.5.122.08

Setting up historical reporting

Important

Genesys Info Mart release 8.5.011 and later provides support for chat session reporting out-of-box, with no additional configuration required on the Genesys Info Mart side. However, to send chat session data to Genesys Info Mart, as well as to see chat session data in GCXI reports, you need to modify the configuration of Interaction Concentrator and RAA.

1. Ensure that your deployment has been configured as required for Genesys Info Mart to support

reporting on eServices activity in general. If necessary, migrate Genesys Info Mart and RAA to meet the release Prerequisites. For a summary of the configuration requirements, see Enabling Reporting on Multimedia Activity in the *Genesys Info Mart Deployment Guide.*

2. Configure Chat Server:
   - To attach the required statistics, set the Chat Server attach-session-statistics option to `all` (which is not the default value).
   - To send intermediate updates (supported by GIM version 8.5.116.20 or higher) for an async chat session, set the required value both for attach-stats-rep-events and attach-stats-rep-place.
3. In the **Route Interaction properties**, ensure that the workflow always provides a "Queue" in **Interation Queue > Queue for Existing Interaction**. If no Queue is provided, the interaction is stopped by an agent desktop, and Chat Server may not be able to update the interaction with reporting statistics.
4. Configure ICON to capture the user data KVPs that Genesys Info Mart requires. Modify the ICON attached data specification file as necessary, to include the KVPs identified in Chat Server reporting data as KVPs that are used by Genesys Info Mart.

   Tip

   The attached data specification file included in the Genesys Info Mart IP (**ccon_adata_spec_GIM_example.xml**) includes all the KVPs required for the reporting features supported in that Info Mart release. You might need to upload a new version of the attached-data specification file or update your existing version with additional KVPs to enable reporting enhancements.

5. Enable aggregation of chat session data. (Required for GCXI reporting or other applications that use RAA aggregation.) In the **[agg-feature]** section on the Genesys Info Mart application object, specify the enable-chat option.

Chat Server reporting data

After a chat session is finished, Chat Server attaches the following types of reporting statistics to the user data of the interaction in Interaction Server:

- Chat session characteristics
- Chat session end reason codes
- Chat session transcript statistics
- Async chat session statistics
- Bot-related statistics

Important

Starting with release 8.5.107, Chat Server attaches reporting statistics and then stops the interaction (if required by the configuration and scenario). Previously, Chat Server was not able to attach the specified reporting statistics if the stop-abandoned-interaction option was set to a value, different from the default value `never` and the corresponding scenario occurred.

Chat Session Characteristics

The following chat session characteristics are attached at the end of a chat session. If the KVPs are required for the out-of-box chat session reporting provided by Genesys Info Mart and GCXI, the "Info Mart Database Target" column indicates the Info Mart database table and column to which the KVP is mapped.

Unless indicated otherwise, the session characteristics KVPs were introduced in Chat Server 8.5.201.

KVP	Description	Info Mart Database Target
ChatServerSessionClosedAt	Timestamp of chat session closure. Always attached. **Note:** This KVP is mandatory for Genesys Info Mart reporting.	CHAT_SESSION_FACT.END_DATE_TIME_KEY
ChatServerSessionStartedAt	Timestamp of chat session creation. Always attached. **Note:** This KVP is mandatory for Genesys Info Mart reporting.	CHAT_SESSION_FACT.START_DATE_TIME_KEY
csg_ChatSessionID	The ID (identifier) of chat session. Could be different from Interaction ID. Attached only if the value of **attach-session-statistics** is not none.	Not mapped
csg_LanguageName	The value identifies the language specified for the chat session. Might be absent. Attached only if the initial UserData for the chat session includes the GCTI_LanguageName KVP, and the value of **attach-session-statistics** is not none.	CHAT_SESSION_DIM.LANGUAGE_NAME (referenced through CHAT_SESSION_FACT.CHAT_SESSION_DIM_KEY)
csg_MediaOrigin	The value identifies the origination of the chat session (web chat, social media channels, sms, and so on). Might be absent. Attached only if the initial UserData for the chat session includes the	CHAT_SESSION_DIM.MEDIA_ORIGIN (referenced through CHAT_SESSION_FACT.CHAT_SESSION_DIM_KEY)
KVP	**Description**	**Info Mart Database Target**

KVP	Description	Info Mart Database Target
	MediaOrigin KVP, and the value of **attach-session-statistics** is not none.	
csg_MediaType **Introduced:** 8.5.203.09 (restricted release) / 8.5.301.06 (generally available release)	The MediaType for chat interaction. Always attached.	MEDIA_TYPE.MEDIA_NAME_CODE (referenced through CHAT_SESSION_FACT.MEDIA_TYPE_KEY)
csg_TenantId	The tenant ID for the chat session. Always attached.	CHAT_SESSION_FACT.TENANT_KEY
KVP	**Description**	**Info Mart Database Target**

Chat Session End Reason Codes

The following reason codes describe what triggered the end of a chat session and how it was triggered. If the KVPs are required for the out-of-box chat session reporting provided by Genesys Info Mart and GCXI, the "Info Mart Database Target" column indicates the Info Mart database table and column to which the KVP is mapped.

KVP	Description	Info Mart Database Target
csg_SessionEndedAgent **Introduced:** 8.5.109	The indication of agent presence in chat session. Please note that in this reason code, only human (in other words, non-bot) agents who are visible to a customer are taken into account. **Valid values:** • ABSENT — Session considered as abandoned. No agent (in other words, not-bot participant visible to client) ever joins chat session. • PRESENT — Session considered as not abandoned. At least one agent is still participating in chat session during the moment of chat session closure. • VISITED — Session could be	Not mapped
KVP	**Description**	**Info Mart Database Target**

KVP	Description	Info Mart Database Target
	considered either as abandoned or not abandoned - depending on business requirements. At least one agent participated in chat session, but no agents were present at the moment of chat session closure. **Note:** In the very specific condition of a session restoration having occurred where an agent joins the session before restoration and does not re-join after restoration, and no messages are sent by any chat party before restoration, the value of csg_SessionEndedAgent will be ABSENT.	
csg_SessionEndedBy **Introduced:** 8.5.105	The type of participant that triggered the chat session closure. **Valid values:** • CLIENT — Denotes a customer. This value is provided whenever a client leaves the chat session first. For example, this value will be set when a client leaves while the session continues due to the presence of an agent and ended later by an agent. • AGENT, SUPERVISOR, BOT — Denotes either agent, supervisor or chat bot participant. This type is provided only when: • A session is `closed` because the actor (agent/supervisor/bot) sent the `Release` request with the `close if no more agents`, or `force close` after-action; or • A session without a customer during the course of this chat session	CHAT_SESSION_DIM.ENDED_BY (referenced through CHAT_SESSION_FACT.CHAT_SESSION_DIM_KEY)
KVP	**Description**	**Info Mart Database Target**

KVP	Description	Info Mart Database Target
	is closed because the actor sent a Release request. • SYSTEM — Denotes a server/ system. See the `csg_SessionEndedReason` table for possible reasons.	
csg_SessionEndedReason **Introduced:** 8.5.105	The description of how a chat session was closed. **Valid values:** • DISCONNECT — The participant left due to a disconnect (basic protocol) or a flex timeout expiration (denotes disconnect in flex protocol). **Possible values for the associated `csg_SessionEndedBy`:** CLIENT, AGENT, SUPERVISOR, BOT • QUIT — The participant left a chat session in a normal way (flex logout or basic `self-release` request, that is with the `keep alive` after-action). **Possible values for the associated `csg_SessionEndedBy`:** CLIENT, AGENT, SUPERVISOR, BOT • FORCE — The participant left a chat session in a normal way and requested the session to be closed (either `close if no more agents` or `force closure` after-action). **Possible values for the associated `csg_SessionEndedBy`:** AGENT, SUPERVISOR, BOT • INACTIVE — Chat Server closed a chat session due to	CHAT_SESSION_DIM.ENDED_REASON (referenced through CHAT_SESSION_FACT.CHAT_SESSION_DIM_KEY)
KVP	**Description**	**Info Mart Database Target**

KVP	Description	Info Mart Database Target
	activated inactivity control monitoring. **Possible values for the associated `csg_SessionEndedBy`:** SYSTEM • DB_ERROR — Chat Server closed a chat session because it received the non-recoverable error from UCS while attempting to save the intermediate chat transcript (only possible when the **transcript-save-on-error** option is set to `close`). **Possible values for the associated `csg_SessionEndedBy`:** SYSTEM	
KVP	**Description**	**Info Mart Database Target**

Chat Session transcript statistics

Chat Server attaches general and extended reporting statistics, based on the attach-session-statistics option settings.

General transcript statistics

In the general transcript statistics, an *agent* means both an agent and a supervisor, when either of those is visible to a customer. For example, the statistics do not count/include an activity for an agent who is coaching another agent, or for a supervisor who monitors the session silently.

The following general transcript statistics are attached at the end of a chat session. If the KVPs are required for the out-of-box chat session reporting provided by Genesys Info Mart and GCXI, the "Info Mart Database Target" column indicates the Info Mart database table and column to which the KVP is mapped.

Unless indicated otherwise, the general transcript statistics KVPs were introduced in Chat Server 8.5.101.

KVP	Description	Info Mart Database Target
cse_ActiveIdleMaxTime **Introduced:** 8.5.301.06	The maximum time (in seconds) a chat session has been inactive while at least one agent was connected and a configured inactivity threshold was exceeded.	Not mapped
cse_ActiveIdleTotalCount **Introduced:** 8.5.301.06	The total number of times when an inactivity period exceeded a configured threshold while at least one agent was connected to the chat session (in other words, while the chat session was technically in an active state).	CHAT_SESSION_FACT.ACTIVE_IDLE_COUNT
cse_ActiveIdleTotalTime **Introduced:** 8.5.301.06	The total amount of time (in seconds), exceeding configured threshold, without any activity when the chat session was in the active state (at least one Agent participated).	CHAT_SESSION_FACT.ACTIVE_IDLE_DURATION
cse_SessionHandleMaxTime **Introduced:** 8.5.301.06	The maximum time (in seconds) that at least one agent was connected to a chat session.	Not mapped
cse_SessionHandleTotalCount **Introduced:** 8.5.301.06	The total number of times a session was in an active state, that at least one agent was connected to a chat session.	CHAT_SESSION_FACT.HANDLE_COUNT
cse_SessionHandleTotalTime **Introduced:** 8.5.301.06	The total time (in seconds) that at least one agent was connected to a chat session.	CHAT_SESSION_FACT.HANDLE_DURATION
csg_MessagesFromAgentsCount	The total number of all messages sent by all agents (messages which are visible to customer). **Note:** There can be several agents in a chat session, for example, conferences, transfers, and others.	CHAT_SESSION_FACT.MSG_FROM_AGENTS_COUN
csg_MessagesFromAgentsSize	The total character count (including spaces) of all messages sent by agents.	CHAT_SESSION_FACT.MSG_FROM_AGENTS_SIZE
csg_MessagesFromCustomersCount	The total number of messages sent by customers.	CHAT_SESSION_FACT.MSG_FROM_CUSTOMERS_C
csg_MessagesFromCustomersSize	The total character count (including spaces) of all messages sent by customers.	CHAT_SESSION_FACT.MSG_FROM_CUSTOMERS_S
csg_PartiesAsAgentCount	The number of parties that participated in a session as agents.	CHAT_SESSION_FACT.AGENTS_COUNT
KVP	**Description**	**Info Mart Database Target**

KVP	Description	Info Mart Database Target
	Note: Only unique parties are counted. For example, if the same party joins the session several times, it only counts as one for the purpose of this statistic.	
csg_PartiesAsCoachCount	The number of parties that participated in a session in the coaching mode (for example, an agent joins with the VIP visibility). **Note:** Only unique parties are counted. For example, if the same party joins the session several times, it only counts as one for the purpose of this statistic.	Not mapped
csg_PartiesAsMonitorCount	The number of parties that participated in a session in the monitoring mode (for example, a supervisor join with the INT visibility). **Note:** Only unique parties are counted. For example, if the same party joins the session several times, it only counts as one for the purpose of this statistic.	Not mapped
csg_SessionTotalTime	The total duration of a chat session from the time it was created until it was completely finished/closed in Chat Server. **Note:** This does not include the time between `Chat Session End` and `Mark Done`, as the interaction can still be handled by an agent.	CHAT_SESSION_FACT.SESSION_DURATION
csg_SessionUntilFirstAgentTime	The duration of the waiting period, or the period of time a customer waits until the first agent (visible to a customer) joined the session. **Note:** The 0 (zero) value has two alternative interpretations: no agents ever joined the session (if csg_PartiesAsAgentCount=0) or an agent joined immediately when the session was started (if csg_PartiesAsAgentCount > 0).	CHAT_SESSION_FACT.UNTIL_FIRST_AGENT_DURAT
csg_SessionUntilFirstReplyTime	The period of time until the first agent submits the first visible to a customer greeting/message into a chat session.	CHAT_SESSION_FACT.UNTIL_FIRST_REPLY_DURAT
csg_SessionWithCustomerTime	The period of time a customer is in a chat session.	Not mapped
KVP	**Description**	**Info Mart Database Target**

Extended (wait-reply) statistics

The extended statistics provide details about customer and agent wait and reply times. As in the case of general transcript statistics, an "agent" means both an agent and a supervisor, when either of those is visible to a customer.

- `Wait` time - The time between a message from the reporting party (or the last message, if there were a few messages in a row) being sent and the first message from another party being received in a reply.
- `Reply` time - The time between a message (or the first message, for a few messages in a row) from another party being received and the message from reporting party being sent in a reply.

The following extended transcript statistics are attached at the end of a chat session. If the KVPs are required for the out-of-box chat session reporting provided by Genesys Info Mart and GCXI, the "Info Mart Database Target" column indicates the Info Mart database table and column to which the KVP is mapped.

Unless indicated otherwise, the extended transcript statistics KVPs were introduced in Chat Server 8.5.101.

Important
The calculation of TotalCount/MaxTime/TotalTime was adjusted and does not include dormant state for async chat sessions for "Extended (wait-reply) Statistics": `cse_AgentReply` and `cse_AgentWait`.

KVP	Description	Info Mart Database Target
cse_AgentReplyMaxTime	The maximum time (in seconds) an agent spent on replying to a customer. **Note:** For async chat sessions, if a chat session was in a dormant state while a customer message was received, the time until the agent rejoins the session is excluded.	CHAT_SESSION_FACT.AGENT_REPLY_MAX_DURAT
cse_AgentReplyTotalCount	The number of times an agent replied to a customer.	CHAT_SESSION_FACT.AGENT_REPLY_COUNT
cse_AgentReplyTotalTime	The total time (in seconds) an agent spent on replying to a customer.	CHAT_SESSION_FACT.AGENT_REPLY_DURATION
KVP	**Description**	**Info Mart Database Target**

KVP	Description	Info Mart Database Target
	Note: For async chat sessions, if a chat session was in a dormant state while a customer message was received, the time until the agent rejoins the session is excluded.	
cse_AgentWaitMaxTime	The maximum time (in seconds) an agent spent on waiting the reply from a customer. **Note:** For async chat sessions, cumulative dormant time until a customer's reply is received is excluded.	CHAT_SESSION_FACT.AGENT_WAIT_MAX_DURATI
cse_AgentWaitTotalCount	The number of times an agent waited for replies from a customer.	CHAT_SESSION_FACT.AGENT_WAIT_COUNT
cse_AgentWaitTotalTime	The total time (in seconds) an agent spent on waiting the reply from a customer. **Note:** For async chat sessions, cumulative dormant time until a customer's reply is received is excluded.	CHAT_SESSION_FACT.AGENT_WAIT_DURATION
cse_CustomerReplyMaxTime	The maximum time (in seconds) a customer spent on replying to an agent.	CHAT_SESSION_FACT.CUSTOMER_REPLY_MAX_DU
cse_CustomerReplyTotalCount	The number of times a customer replied to an agent.	CHAT_SESSION_FACT.CUSTOMER_REPLY_COUNT
cse_CustomerReplyTotalTime	The total time (in seconds) a customer spent on replying to an agent.	CHAT_SESSION_FACT.CUSTOMER_REPLY_DURATI
cse_CustomerWaitMaxTime	The maximum time (in seconds) a customer spent on waiting the reply from an agent.	CHAT_SESSION_FACT.CUSTOMER_WAIT_MAX_DU
cse_CustomerWaitTotalCount	The number of times a customer waited for the reply from an agent.	CHAT_SESSION_FACT.CUSTOMER_WAIT_COUNT
cse_CustomerWaitTotalTime	The total time (in seconds) a customer spent on waiting the reply from an agent.	CHAT_SESSION_FACT.CUSTOMER_WAIT_DURATIO
KVP	**Description**	**Info Mart Database Target**

Async chat session statistics

Async chat session statistics are provided only for async chat sessions (in other words, when

`GCTI_Chat_AsyncMode=true` is specified during session creation). The calculation of these statistics takes into account the active and dormant phases of an async chat session, as well as async inactivity control (which is different from regular inactivity control).

KVP	Description	Info Mart Database Target
cse_AsyncDormantMaxTime **Introduced:** 8.5.301.06	The maximum time (in seconds) a chat session was staying in dormant state.	Not mapped
cse_AsyncDormantTotalCount **Introduced:** 8.5.301.06	The total number of times session entered dormant state	CHAT_SESSION_FACT.ASYNC_DORMANT_COUNT
cse_AsyncDormantTotalTime **Introduced:** 8.5.301.06	The total amount of time (in seconds), customer chat session was in the dormant state (with no Agent participant). Routing time is excluded from dormant time.	CHAT_SESSION_FACT.ASYNC_DORMANT_DURATIC
cse_AsyncIdleMaxTime **Introduced:** 8.5.301.06	The maximum time (in seconds) an async chat session was staying in idle state.	Not mapped
cse_AsyncIdleTotalCount **Introduced:** 8.5.301.06	Total number of times an async session entered idle state.	CHAT_SESSION_FACT.ASYNC_IDLE_COUNT
cse_AsyncIdleTotalTime **Introduced:** 8.5.301.06	The total amount of time (in seconds), exceeding configured threshold, without any activity when the chat session was in the dormant state (with no Agent participant).	CHAT_SESSION_FACT.ASYNC_IDLE_DURATION
csg_ChatAsyncMode **Introduced:** 8.5.301.06	Denotes async session. Equals "1" for async chat session or "0" for regular chat session.	CHAT_SESSION_DIM.ASYNC_MODE (referenced through CHAT_SESSION_FACT.CHAT_SESSION_DIM_KEY)
KVP	**Description**	**Info Mart Database Target**

Bot-related statistics

In deployments that include BGS, Chat Server also attaches the following KVPs:

- csg_MessagesFromBotsCount
- csg_MessagesFromBotsSize
- csg_SessionUntilFirstBotTime
- csg_PartiesAsBotCount

For more information about the bot-related KVPs, see Bot-related reporting data in the *Bot Gateway Server Quick Start Guide* (accessible only to restricted release customers).

Known limitation

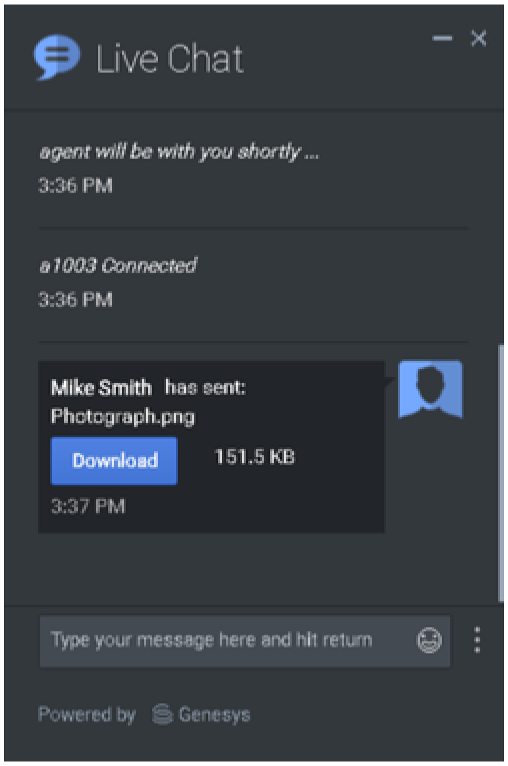
The following is a known limitation with async chat reporting:

- Information about a chat session running in an async mode is available only after the chat session ends.

File Transfer in Chat Solutions

Overview

Genesys Chat solutions allow transfer of files between chat customers and agents, in either direction.



Components and Support

Component, Minimum Supported Version	Remarks	Configuration
Agent Desktop 8.5.115.17	Workspace Desktop Edition supports downloading files, uploading files (from a file system and from the Standard Response Library), and displaying and opening uploaded files. For custom	Allow the privileges **Chat - Can Transfer File From File System** and/or **Chat - Can Transfer File From Standard Response**, as described in "Transferring Files to Contacts" in the

Component, Minimum Supported Version	Remarks	Configuration
	desktops, this functionality can be implemented using Platform Services SDK. Allowances for file uploading by the agent are managed by Workspace settings. Warning Workspace Web Edition does not support file transfer. Make sure to disable file transfer in Chat Server and CX Widget when using WWE.	Workspace Deployment Guide. To disable file transfer for agents, remove those privileges.
Chat Server 8.5.105.05	Chat Server provides the ability to send notifications about files transferred by customers and agents. When using Web Chat with the GMS API, it also manages the allowances for file upload and download operations by the customer. For example, types of files, file size, total file transfer size, and number of files.	The options that control file transfer are: • upload-max-files • upload-max-file-size • upload-max-total-size To disable file transfer for customers set one of these options to zero. All Chat Server options are documented here.
Chat Widget 8.5.004.15	Genesys Chat Widget supports uploading, downloading, and displaying files. To implement this functionality in your own chat widget see GMS Chat API Version 2 and Web Chat Configuration in the Genesys Widgets Reference.	The **Send File** button is hidden by default. To enable it, set the **uploadsEnabled** option to `true`. Note that this only controls the visibility of the button in the UI; to enable or disable the functionality itself, you must configure the servers appropriately. Chat Widget options are documented here.
E-mail Server 8.5.103.11	The chat session transcript email displays the name and size of any files transmitted. To maintain data privacy, the files themselves are not attached.	No special configuration required.

Component, Minimum Supported Version	Remarks	Configuration
Genesys Mobile Services 8.5.106.14	GMS provides Chat API Version 2, which enables Chat Widget to upload and download files.	No special configuration required.
Universal Contact Server all versions 8.5.x and later	All transferred files are stored in UCS as documents. The chat session transcript contains only the reference IDs of the documents. No changes are required to UCS to support this functionality.	No special configuration required.

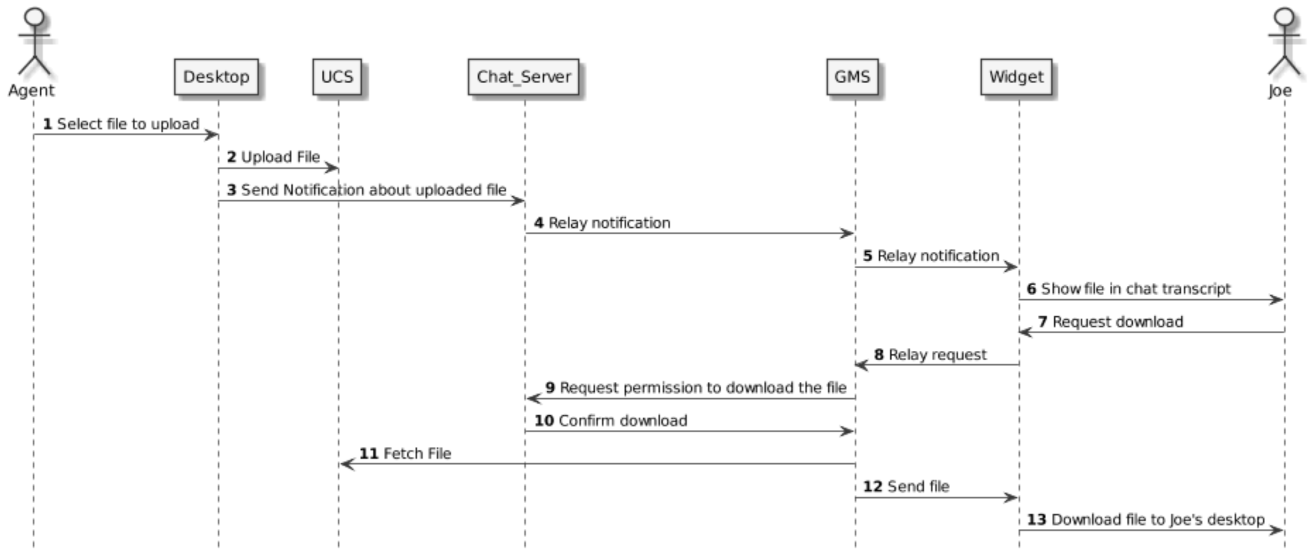
When using file transfer, be aware that:

- File transfer puts an additional load on network communications between Widget and Genesys Mobile Services (GMS), GMS and Universal Contact Server (UCS), and Desktop and UCS. Transferring large files between these components affects CPU, network, and memory usage.
- The UCS database needs to store large volumes of data (files).

How It Works

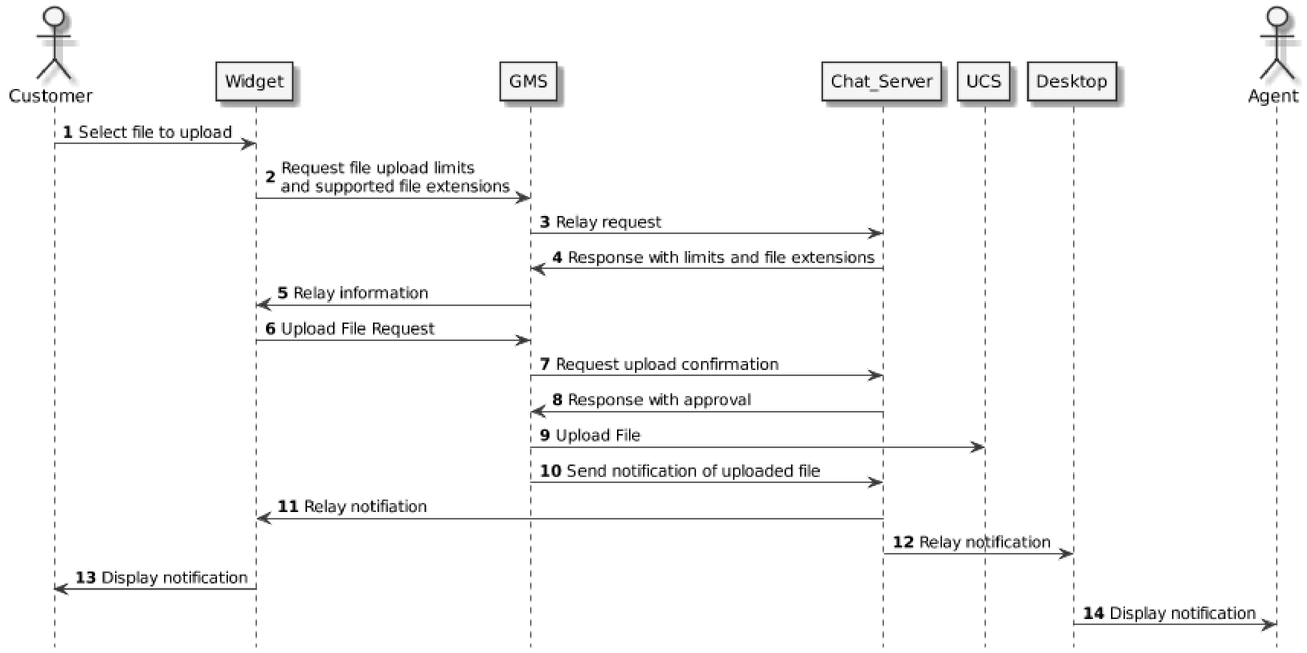
This section is provided for users who want to implement a custom solution.

Agent to Customer



1. The agent selects a file from the disk or selects a Standard Response. When the agent clicks **Send**, the file is uploaded to UCS, and notification is sent about the uploaded file in chat session.
2. When the Widget receives notification about file from agent, it shows it to a customer as an icon.
3. When the customer wants to download the file, they request it from the Widget and the Widget sends a request to GMS that:
    1. Requests permission from Chat Server for file download (only a certain number of attempts are allowed in each chat session)
    2. Downloads the file from UCS and sends it to the Widget, which in turn downloads it to the customer.

Customer to Agent



1. The customer selects a file to send.
2. The Widget sends a request via GMS to Chat Server to provide current size limits and supported file extensions.
3. If the conditions are satisfied, the Widget sends the file to GMS, which:
    1. Requests permission from Chat Server to upload the file.
    2. If permitted, uploads the file to UCS.
    3. Sends a notification in the chat session about the uploaded file.
4. The Widget receives the notification and shows it in that chat transcript window as an icon.
5. When the agent opens the file (by double-clicking the file icon), the Desktop downloads the file directly from UCS.

Known Limitations

- Virus scanning of transferred files is not included in the solution. Genesys recommends installing virus scanning at the Agent's desktop to scan the files during downloading and uploading.
- In high availability mode, during Chat Server failover, all information about the transferred file is preserved in the chat session transcript. If there is a simultaneous failover of UCS and GMS (or Workspace) it is possible that a file might be uploaded to UCS but not be reported as uploaded in the chat session.

Chat Server API selected notes and topics

The following pages describe only selected topics about special use cases which might require additional clarification or explanation for the purpose of being used in custom agent desktops and chat widgets. These pages do not contain the complete description of Chat Server API which is implemented in Genesys PSDK as flex (`Genesyslab.Platform.WebMedia.Protocols` > `FlexChatProtocol`) and basic (`Genesyslab.Platform.WebMedia.Protocols` > `BasicChatProtocol`) protocols.

Throughout these pages we will be using PSDK .NET for this demonstration purpose (Java PSDK is very similar).

- Functional capabilities of chat protocol
- File Transfer API for Agent Desktop
- Description of Chat Protocol Elements
- Reason Codes

Chat Server Client Version

Chat Server exposes two chat protocols: `flex` (connectionless) and `basic` (connection based). These protocols are constantly evolving with new capabilities. In order to allow the protocol negotiation with Chat Server, PSDK (which implements both protocols) has a special hard-coded attribute **`ClientVersion`**. This attribute defines what functionality is available for a Chat Server API client (for example, it prevents Chat Server from sending unsupported events and/or attributes in replies). The following table describes the existing protocol versions:

ClientVersion	Chat Server version	PSDK version	Description (features added or changed)
101	8.0.100.07	8.0.100.06	Base chat functionality. Default version.
102	8.1.000.33	8.1.000.08	Support for user nickname change: new notice type USER_UPDATE_NICK.
103	8.5.102.06	8.5.200.03	Support for Idle control functionality: new notice types `IDLE_CONTROL_ALERT`, `IDLE_CONTROL_CLOSE`, `IDLE_CONTROL_SET`. New transcript attribute `idleTimeExpire`, new user type SYSTEM, new protocol type NONE.
104	8.5.104.07	8.5.201.00	Support for chat system

ClientVersion	Chat Server version	PSDK version	Description (features added or changed)
			commands: new notice type `SYS_COMMAND`. Flex transcript events have been extended with `userData` of type `KeyValueCollection` (only for notice event).
105	8.5.105.04	8.5.301.00	Support for chat session silent monitoring indication: new transcript attribute `monitored`.
106	8.5.109.06	9.0.000.00	All transcript events have been extended with `eventAttributes` of type `KeyValueCollection`.

Functional capabilities of chat protocol

This page describes various Chat Server protocol elements which can be used in the implementation of custom agent desktop applications.

Direct Messages

Chat Server can send so called direct (or private) messages and notices to a participant in chat session. Only chat basic protocol provides such functionality. In order to send a message or a notice which will be visible only to a certain participant in chat session, **`ReceiverId`** in methods **`RequestMessage`** and **`RequestNotify`** (defined in `Genesyslab.Platform.WebMedia.Protocols.BasicChat.Requests`) must be initialized with the `userId` of the intended participant (which can be obtained from the transcript event). In this case, only two participants will see this message in the transcript: the sender and the recipient.

When receiving a direct message, the transcript will contain either **`MessageInfo`** or **`NoticeInfo`** (defined in `Genesyslab.Platform.WebMedia.Protocols.BasicChat`) with corresponded **`ReceiverId`**.

Supported:

Chat Server	PSDK	Workspace (both) Edition	Chat Widget	GMS
8.5.108	8.5.1x	not supported	not supported	not supported

Enhancing security when joining a chat session

Using configuration option `session-password-enforce`, it is possible to force Chat Server to generate the `crypto-random` security token (we call it "session password") which will be associated with a chat session during its creation. In this case, Chat Server will require this session password each time a new participant sends a request to join an existing chat session (it must be provided in **`GCTI_Chat_SessionPassword`** key/value pair in `userdata` of **`RequestJoin`**). Chat Server attaches the session password to the `userdata` of the interaction (submitted to Interaction Server) in **`ChatServerSessionPassword`** key/value pair). Only in basic chat protocol it is possible to specify a user-defined session password by adding **`GCTI_Chat_SessionPassword`** key/value pair in `userdata` of **`RequestJoin`** when creating a chat session.

Supported:

Chat Server	PSDK	Workspace (both) Edition	Chat Widget	GMS
8.5.109	8.5.1x	not supported	joining an existing chat session is not supported	joining an existing chat session is not supported

Chat bot participant special treatment

Only the agent or supervisor in a chat session can be marked as "bot" participants. It happens when the `userdata` of **`RequestJoin`** (when participant joins chat session) contains **`GCTI_Chat_SetPartyStyle`** key/value pair with value "BOT". Chat Server attaches another key/value pair **`GCTI_Chat_PartyStyle="BOT"`** to the `newParty'` event in basic protocol chat transcript and **`GCTI_SYSTEM/party-into/style"="BOT"`** in `eventAttributes` property (both in `newParty` event in basic protocol and in all events for bot participant in flex protocol).

For "bot" participants:

- Chat Server does not take such participants into account when processing after-action in **`RequestReleaseParty`** with value `CloseIfNoAgents`.
- Agent Desktop must not take such participants into account when making a decision to stop the processing of chat session and interaction.
- Reporting statistics (see Chat Server Reporting Statistics) will not count such participants as an agent or supervisor.

Supported:

Chat Server	PSDK	Workspace Desktop Edition	Workspace Web Edition	Chat Widget	GMS
8.5.109	8.5.1x for userdata location, 9.0.000.01 for eventAttributes	8.5.118	not supported	not supported	8.5.201.04

Notifications about detected and masked out PII data

Chat Server can be configured to detect and replace PII data in a chat session (see Masking Sensitive Data). If such PII data is detected according to the configuration provided, the message event (both in flex and basic chat transcripts) will contain information in the **`eventAttributes`** property about what parts of the message contains detected PII data, and how this data was masked out. In Chat Server logs it can be seen as (text is formatted for presentation):

```
eventAttributes={'GCTI_SYSTEM'={'pii-cleanup'={
    'rule-0001'={
        'description'='<rule-description>',
        'id'='<rule id>',
        'name'='<rule name>',
        'positions'={
            '70-81'={'replaced'='digits'}
}}}}}
```

Supported:

Chat Server	PSDK	Workspace (both) Edition	Chat Widget	GMS
8.5.109	9.0.000.01	not supported	not supported	8.5.201.04

Read confirmation notice

Chat Server provides the possibility for chat session participants to signal about messages being seen/read. For that, a participant must send **`RequestNotify`** with notice type **`SYS_COMMAND`** and notice text **`read-confirm`**. The userdata of the request must contain key-value pair with key **`last-event-id`**, and the value must contain the transcript event ID (which is being reported as being seen). Chat Server processes read confirmation notices as follows:

- Other chat participants will receive corresponding notification with provided `last-event-id` in userdata of the notice transcript event.
- The notice event will be saved in UCS transcript only if option `transcript-save-notices = all`.

Participant’s read confirmation notice events get annihilated from transcript:

- When a participant leaves the session.
- When another read confirmation notice is received from the same participant.
- During the session restoration.

Supported:

Chat Server	PSDK	Workspace Desktop Edition	Workspace Web Edition	Chat Widget	GMS
8.5.105	8.5.1x	8.5.122.08	not supported	not supported	8.5.201.04

Nickname change

Chat Server provides the possibility for chat session participants to change their nickname during the session. For that, a participant must send **`RequestNotify`** with notice type **`USER_UPDATE_NICK`** and text containing a new nickname. The nickname of a participant can be changed more than once. Upon receiving such request:

- Chat Server updates the nickname for a participant.
- Chat Server adds this notice to the session transcript.
- Only when updated the nickname for the first time, Chat Server records the original nickname value in **`GCTI_Original_Nickname`** key-value pair of `userdata` of the initial `newParty` event for that participant.

Supported:

Chat Server	PSDK	Workspace (both) Edition	Chat Widget	GMS
8.5.0	8.1.1	not supported	not supported	8.5.201.04

Using rich messaging

Chat Server provides chat session participants (either an agent or chat bot) with the ability to use Rich Messaging (in other words to send structured messages) in a chat session. To do so, a participant must send `RequestMessage` (com.genesyslab.platform.webmedia.protocol.basicchat) with the following mandatory parameters:

- `MessageText` with plain text message. For backward compatibility, structured messages must be always accompanied with so called "fallback" plain text messages. This way, a component which does not know how to process the structured content will continue to use a regular plain text message supplied together with the structured content.
- `eventAttributes` with `KeyValueCollection` with the following content (nested nodes):
  - "`structured-content`" (of type kvlist)
    - "`<origin name>`" (of type kvlist)
      - "`content`" (of type string) - contains structured message as a valid JSON string (which varies by channel).
      - "`type`" (of type string) - contains the content type, and is mandatory for all channels (except for Widgets).

"`<origin name>`" can be one of the following supported channels:

Channel	<origin name> value
Apple Business Chat	applebc-session
WhatsApp	`genesys-chat`
Genesys Chat Widget Rich Messaging	`genesys-chat`

If supported by a channel, the reply message from a customer can contain a "`related-event-id`" attribute (under the "`general-properties`" node in the `eventAttributes`). It contains the corresponding `EventId` of the original outgoing (to a customer) structured message, which is returned in the `EventSessionInfo` when sending a `RequestMessage` in PSDK.

In Chat API Version 2, structured messages are delivered in the messages array, as shown below:

```
"messages":[
  {
    "from":{"nickname":"The Bot", "participantId":3, "type":"Agent"},
    "index":15,
    "text":"<plain text message (fallback version)>",
    "type":"Message",
    "utcTime":1561059828000,
    "eventAttributes":{
      "GCTI_SYSTEM":{"party-info":{"style":"BOT"}},
      "structured-content":{"genesys-chat":{"type":"Quick replies","content":"<contains
structured message as a valid escaped JSON string>"}}
    }
  }
]
```

Supported versions

Component	Version
Chat Server	8.5.109.06
PSDK	9.0.000.01
Workspace Web Edition and Workspace Desktop Edition	Not supported
Chat Widget	9.0.006.02
Genesys Mobile Services	8.5.201.04

File Transfer API for Agent Desktop

The page describes how to implement file transfer (uploading and downloading) in custom agent desktop application using Platform SDK (supported from version 8.5.201.00). For high-level overview of file transfer in Chat Solution please refer to File Transfer in Chat Solutions.

Overview

It is important to understand the following general concepts about file transfer in Chat Solution:

- File storage is independent of Chat Server. Even though only Universal Contact Server (UCS) is currently supported by Genesys components (GMS, Interaction Workspace) as a file storage, other file storage may be used (ensuring that every Chat Server client could operate with the storage).
- Neither Chat Server nor UCS impose any restrictions on files transferred by agent applications (contrary to a customer-faced applications like GMS), therefore performance, network load, security and other aspects should be carefully planned (for example, large files or executables). Genesys Interaction Workspace has a special configuration for agent-initiated file uploads.
- Security protection should be provided by a third-party system as neither Chat Server nor UCS perform antivirus or other security checks of transferred files.

Protocol Description

Agent desktop application can perform the following operations:

- File upload (two-step process):
    1. An application uploads a file to the storage (UCS). A file stored in UCS is referred to as document and is given a unique identifier `document-id`.
    2. The document is attached to a chat session by sending a special notice (“file-uploaded”) and is then made available for chat participants to download. Chat Server assigns another identifier `file-id` for the purpose of identifying the file in chat session.
- Whenever the application receives the “file-uploaded” notice in chat session (from any participant), it:
    - Displays (to the agent) the information about the file such as the file name and file size as well as any custom additional information, in the event/notice.
    - Downloads a file from UCS by using `document-id` (for example, when the agent clicks on the file pictogram).

In code samples, it is assumed that connection objects to UCS and Chat Server are initialized with the proper connection parameters, and that Chat Server basic protocol user has registered and joined the session.

File Upload

The following code snippet demonstrates how to upload a file:

```
// Prepare connections
BasicChatProtocol chatConnection = ...;
UniversalContactServerProtocol ucsConnection = ...;

// Uploading file into UCS
RequestAddDocument docReq = RequestAddDocument.Create();
// mime-type is mandatory, however not verified by UCS or Chat Server.
docReq.MimeType = "application/octet-stream";
docReq.Content = File.ReadAllBytes("C:\\myfile.txt");
docReq.TheName = "myfile.txt";
docReq.TheSize = docReq.Content.Length;
IMessage addDocReply = ucsConnection.Request(docReq);
EventAddDocument addDocEvent = addDocReply as EventAddDocument;
// Sending notification in chat session
RequestNotify fileReq = RequestNotify.Create();
fileReq.SessionId = ...;
fileReq.NoticeText = NoticeText.Create(NoticeType.SystemCommand, "file-uploaded");
fileReq.UserData = new KeyValueCollection();
fileReq.UserData.Add("file-name", docReq.TheName); // mandatory
fileReq.UserData.Add("file-size", docReq.TheSize); // mandatory
fileReq.UserData.Add("file-source", "ucs"); // mandatory
fileReq.UserData.Add("file-document-id", addDocEvent.DocumentId); // mandatory for UCS
fileReq.UserData.Add("file-upload-type", "file-system"); // optional
fileReq.UserData.Add("file-description", "Sample text file.");  // optional
IMessage fileUploadedReply = chatConnection.Request(fileReq);
```

Input Parameters

Below is the description of input parameters which must be provided in the “file-uploaded” notice request to Chat Server.

Name	Mandatory	Description
file-name	Yes	File name including extension.
file-size	Yes	File size in bytes.
file-source	Yes	File storage type. Currently supported values by Genesys Solution (but not limited by Chat Server itself): • “ucs” – Universal Contact Server
file-document-id	Yes only for `file-source=“ucs"`	Document Id in UCS.
file-upload-type	Optional	Values used by Genesys Interaction Workspace: • “file-system” (Uploaded from file system) • “standard-response” (selected

Name	Mandatory	Description
		from standard responses)
<any additional parameters>	Optional	Could be used by application logic.

File Download

The following code snippet demonstrates how to download file:

```
// Prepare connections
UniversalContactServerProtocol ucsConnection = ...;

// Download file from UCS
RequestGetDocument docReq = RequestGetDocument.Create();
docReq.DocumentId = ...; // obtain from chat transcript event userdata
IMessage getDocReply = ucsConnection.Request(docReq);

// Saving file in filesystem
EventGetDocument getDocEvent = getDocReply as EventGetDocument;
string filePath = Path.Combine("C:\\", getDocEvent.TheName);
File.WriteAllBytes(filePath, getDocEvent.Content);
```

Description of Chat Protocol Elements

Chat Protocol user types and visibility levels

The following user types are defined in both (flex and basic) chat protocols:

User type	Represents	Description	Possible visibility level
CLIENT	A customer	The only one type available through flex protocol when starting chat session	ALL
AGENT	An agent	Used by agent desktop	ALL, INT
SUPERVISOR	An agent with supervisor capabilities	A supervisor can monitor chat session invisibly for other chat participants	ALL, INT, VIP
EXTERNAL	Workflow	Used for messages and notices sent from routing strategies	ALL, INT, VIP
SYSTEM	Chat Server	Used by the server for special notifications (for example, idle control notices)	ALL, INT, VIP
Note: Agent and Supervisor user types can also be used by bots. This is specified by the presence of `GCTI_Chat_PartyStyle=BOT` in the userdata and/or by `GCTI_SYSTEM.party-info.style='BOT'` in the event attributes.			

The visibility levels are defined in the table below:

Visibility level	Represent a mode	Description
ALL	Conference	Used by default to conduct a conversation between chat participants
INT	Coaching	Can be used by Agents and Supervisors to communicate invisibly from a customer
VIP	Monitoring	Can be used by Supervisors to invisibly monitor chat participants' activity

Reason Codes

When customer leaves a chat session

Chat Server provides a "`quit-reason-code`" in the "GCTI_SYSTEM" node in the event attributes for the "ABANDON" event (only about that particular participant). This code is provided through GMS Chat API and in GMS custom-http push notifications.

Value	Description
"0"	The participant left the chat session by sending a logout request. This reason code is not delivered in response through GMS Chat API, it is only sent in GMS custom-http push notifications.
"3"	The participant was removed by the server due to chat protocol inactivity (as configured by flex-push-timeout and/or flex-disconnect-timeout options). This reason code is not delivered in response through GMS Chat API, it is only sent in GMS custom-http push notifications.
"4"	The participant was removed by another participant (for example, by the agent, supervisor, or bot).
"5"	The participant was removed by the server (possibly for system reasons; for example workflow stopped the interaction, or a non-recoverable error received from UCS when option `transcript-save-on-error=close`).
"7"	The participant was removed by the server due to chat session inactivity (both for regular and async modes).
"8"	The participant was marked as removed by the server during a chat session restoration (transcript normalization procedure) in HA mode.

Example from Chat server log

```
ABANDON: sc='18', ei='3', nk='my nick', ut='CLIENT', pd='1'
 eventAttributes={'GCTI_SYSTEM'={'quit-reason-code'='3'}}}
```

Asynchronous Chat

Link to video

Asynchronous (async) chat in Genesys Chat Solution means:

- Conducting single chat session between a customer and a contact center for a long period of time (could last for days).
- Providing the possibility for an agent to return a chat session back into the workflow (in other words, putting it into *dormant* state), and to reconnect to chat session later (through workbin or contact history).
- Providing the possibility for a workflow to wake up a dormant chat session for processing upon detecting customer activity or upon the expiration of async inactivity control timeout. Workflow tries to route the interaction to the last handling agent for some period of time before sending it to any other available agent.

The following table lists quick links to helpful topics on async chat within this guide:

Topic	Description
Async Requirements	This topic is a general overview of how to enable async chat capabilities.
Asynchronous Chat in Workspace Desktop Edition	This topic covers how to configure and use asynchronous chat in Workspace.
Asynchronous Chat in Widgets	This topic covers how to configure and use asynchronous chat in Widgets.
Deployment guidelines for async and regular chat	This topic includes guidelines regarding sizing, short polling vs. CometD, disconnects, and idle timeouts.
Chat Business Process Sample	This topic describes a procedure on how to deploy the workflow sample as well as information on testing.
Integrating Chat Server with Genesys Historical Reporting	This topic covers information on historical reporting.

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.

Push notifications via GMS to HTTP server

Starting with version 8.5.311.06, the Chat solution allows you to request push (in other words, unsolicited) notifications through Genesys Mobile Server (GMS) to an HTTP server even when a customer-facing chat web application (Chat Widget) communicates with GMS via "Chat API Version 2". Previously, this was only possible with "Chat API Version 2 with CometD".

To enable this functionality, do the following:

Application	Instructions
GMS	1. Deploy GMS using Cluster Application 2. Configure GMS for Custom HTTP notification 3. Configure GMS with push_notification_include_payload (optional)
Chat Server	1. Add new configuration option flex-push-on-join in the **settings** section with value `true`. This forces Chat Server to acknowledge the push notification subscription during the creation of a chat session. 2. Ensure that option flex-push-enabled is set to `true`, and option flex-push-timeout is set with a larger value (for example, "86400 seconds"). For more information, see Async Requirements. 3. Review the schedule for resending push notifications, when using **GCTI_GMS_PushResend**, defined by the configuration options, flex-push-resend-attempts and flex-push-resend-delay. 4. Adjust, if needed, the value for configuration option flex-push-content. In addition to `session-id` and `user-id`, it is now possible to receive `app-dbid` and `secure-key in push notifications.`
Customer-facing chat web application	1. The web application must supply a set of mandatory key-value pairs in the userdata for the "Request Chat" HTTP method (using a `userData[key-name]` notation): • **GCTI_Chat_PushSubscribe** with the value `true`. This enables push notifications in Chat Server when "Chat API Version 2" is used . • **GCTI_GMS_NodeGroup** with the GMS

Application	Instructions
	cluster name. If the GMS version is 8.5.213.03 or greater, this key-value pair is not required, as it is automatically provided by GMS to Chat Server. • **GCTI_GMS_PushDeviceId** with a unique device ID. This ID is returned in the push notifications as `deviceId`. • **GCTI_GMS_PushDeviceType** with the value `customhttp`. This defines the type of push notification used. • **GCTI_GMS_NotifyRequestor** with value `true`. This forces Chat Server to send push notifications to GMS about the customer's own activity. • **GCTI_GMS_PushIncludePayload** with value `true`. This forces GMS to include the payload (in other words, the chat transcript event content) with a custom-http push notification. Without providing this key-value pair, GMS sends only the `deviceId` (provided in **GCTI_GMS_PushDeviceId**) in the push notification, which can prevent the distribution of sensitive content. When reliable delivery is requested by **GCTI_GMS_PushResend**, this key-value pair must be provided however, in this case, no event-specific payload is provided in the push notification (it only contains some ad hoc data that can be used to send a “Refresh” request). 2. Chat Server provides the ability to request a reliable delivery of push notifications. For that, the web application must additionally supply the **GCTI_GMS_PushResend** key-value pair with value `true` in the userdata. This forces Chat Server to activate the mechanism of resending push notifications according to a schedule defined in the configuration. Chat Server will start resending push notifications if no "refresh" (in other words "pull transcript update") request is being received within the amount of time specified by option flex-push-resend-delay. See below for more information about reliable push notifications delivery. 3. The web application can additionally supply a set of key-value pairs in the userdata: • **GCTI_GMS_PushProvider** Must be provided if you specified the configuration for the non-default provider in

Application	Instructions
	GMS. • **GCTI_GMS_PushDebug** Must be provided if you specified the debug mode for the provider configured in GMS. • **GCTI_GMS_ClientChannel** Must be provided if you want to include the GMS service name in obfuscated secure-key in the push notification.

Additional notes

- It is important to provide adequate throughput of the Web Server which processes the `customhttp` notification. The latency (in other words, the processing time for a single HTTP POST request) must be as low as possible as GMS sends all notifications sequentially. The next request is only sent after a reply from the previous one. For example, if the latency is 5 milliseconds on average, then a single GMS node is able to send 200 notifications per second. Enabling `GCTI_GMS_PushResend` could increase the volume of notifications, so it must be taken into account.
- If push notifications are enabled, Chat Server tries to find the GMS node in the GMS cluster (specified by **GCTI_GMS_NodeGroup**) and to associate that found node for further notifications (until the node is disconnected). Starting with version 8.5.311.06, if no GMS node is available (in other words, registered in Chat Server) in a given cluster, Chat Server selects another GMS cluster to seek for an available GMS node. Otherwise, if no other cluster and/or node is available, Chat Server attempts to find an available node the next time an activity is generated in the chat session or upon chat session restoration in HA mode.
- If reliable delivery of a push notification is not requested by sending **GCTI_GMS_PushResend**, no attempts to resubmit the same push notification will be made in case of a delivery failure between the GMS and HTTP server, and between Chat Server and GMS. The following log messages are logged in the event of this error condition:
  - **In GMS:** `"Dbg 09900 [com.genesyslab.PCT.invoker.default] DC Chat Server Persistent Listener: Event 17 was not (GMS is not running in full mode or incompatible Chat Server version) pushed for delivery to customhttp for device..."`
  - **In Chat Server:** `"Trc 59758 push-flex: could not send notification - no gms node found in group=..."`

Sample configuration for custom HTTP notifications in GMS

```
[chat]
enable_notification_mode=true
push_notification_include_payload=true

[push]
customhttp.url=http://<hostname>:<port>/<path>
pushEnabled=comet,customhttp
```

Important

Ensure that the `[push]` section does not contain the option *customhttp.message*. If it is present, the value of this option overrides the content of push notification.

Reliable push notifications delivery

When requesting reliable delivery for a push notification (in other words, when **GCTI_GMS_PushResend**=`true`):

- All push notifications are of **type**:`PushUrl` and **participantId**: `0` (which is not a valid participant ID).
- No payload is provided in the push notification. Instead, each push notification must be considered a trigger to send a “Refresh” request to GMS in order to obtain the newly published events in the chat session.

The following is the sample JSON which is delivered in the HTTP request for a push notification.

```
{
  "message":{

"secureKey":"G1xBGx9aTUYVBEECD0UZAVwTQEQDFgRZFVJTXEI3QSFFIShAHyVcRUI2GUJXXUEeAikkNSNTJFddQRc=",
    "chatId":"deprecated",
    "nextPosition":17,
    "messages":[
      {
        "from":{
          "nickname":"",
          "participantId":0,
          "type":"Client"
        },
        "index":0,
        "text":"PUSH-NOTIFICATION",
        "type":"PushUrl",
        "utcTime":1568662361000,
        "userData":{
          "notify-attempt":"0",
          "notify-position":"16",
          "secure-key":"c6c9a6d96dc14cef5f94",
          "app-dbid":"131",
          "user-id":"007D5D7FE31F001B",
          "session-id":"00020aEQFW6V0029"
        }
      }
    ],
    "alias":"0",
    "chatEnded":false,
    "userId":"deprecated",
    "statusCode":0,
    "monitored":false
  },
  "deviceId":"a1a23456789123456789"
}
```

Important field descriptions

Field	Description
participantId	Always `0` and must be ignored.
notify-position	Contains the starting position of content not retrieved. It can have a value of `-1` meaning that chat participant has been removed from the chat session.
notify-attempt	Contains the number of attempts to deliver the push notification.
secure-key	Secure key to be used with GMS REST API. The presence depends on flex-push-content.
app-dbid	App DBID (or alias) to be used with GMS REST API. The presence depends on flex-push-content.
user-id	User ID to be used with GMS REST API. The presence depends on flex-push-content.
session-id	Session ID to be used with GMS REST API. The presence depends on flex-push-content.
chatEnded	If the value is `true` it means the chat session is finished.

Warning

Starting with version 8.5.311.06, the `secure-key` for REST API requests is provided in the `userData` based on the value of the configuration option flex-push-content. The `secureKey` provided in `message` must be ignored by the REST API client, and only used for the CometD API.

Asynchronous Chat in Workspace Desktop Edition

Asynchronous chat is supported in this restricted release of Workspace Desktop Edition. This feature keeps chats open after the last agent leaves the session, and the agent can rejoin the session until the session is marked Done.

This topic covers the following information about configuring and using asynchronous chat in Workspace:

- Configuring Asynchronous Chat in Workspace Desktop Edition
- Workspace Desktop Edition Agent Use Cases

Configuring Asynchronous Chat in Workspace Desktop Edition

The *Workspace Desktop Edition* describes how to deploy Workspace in your Genesys environment and configure the Chat channel. All the Workspace documentation can be found here.

Refer to the following topics for specific information about setting up agents and the chat channel after you deploy Workspace:

- Setting Up Agents On The System
- Enabling Internal And External Communications: Chat
- Enabling Internal And External Communications: Chat Monitoring
- Team Leads and Supervisors
- Configuration Options Reference / Section interaction-workspace / Chat Options
- Configuration Options Reference / Section interaction-workspace / Chat Server Options
- Configuration Options Reference / Section interaction-workspace / Team Lead Options
- Role Privileges / Chat Privileges

The following sections list the privileges and configuration options that you must set to enable Asynchronous Chat:

Privileges

To enable asynchronous chat in Workspace you must allow the following privileges on the agent role:

- Chat - Can Place On Hold — Allows agents to leave and rejoin an asynchronous chat session. Depends on the `Chat - Can Use Chat Channel` privilege.

- Chat - Can Release Async — Allows agents to manually terminate an asynchronous chat session. Depends on the `Chat - Can Use Chat Channel` privilege.
- Chat - Can Release — Allows agents to manually terminate a standard non-asynchronous chat conversation. Depends on the `Chat - Can Use Chat Channel` privilege.

Configuration

To configure the behavior of asynchronous chat, set the values of the following options on the application, tenant, agent group, or agent in the `interaction-workspace section`:

chat.on-hold-queue

- Default Value: ""
- Valid Values: Any valid Script name of type 'Interaction Queue'.
- Changes take effect: At next interaction attempt to put a chat in a queue.
- Description: Specifies the Interaction Queue where the chat interaction is placed when the agent clicks **Place chat on hold**. This option can be overridden by a routing strategy, as described in Overriding Options by Using a Routing Strategy.

keyboard.shortcut.interaction.chat.hold

- Default Value: ""
- Valid Values: The name of a key or a key combination that begins with one of the following modifier key names: Ctrl, Shift, and Alt, and ends with a character key. Separate the modifier key name from the character key with the '+' character.
- Changes take effect: When the application is started or restarted.
- Description: The combination of keys that can be used as a keyboard shortcut to place a chat on hold. For example: `F1`, `Ctrl+Alt+V`, `Ctrl+Shift+Alt+V`.

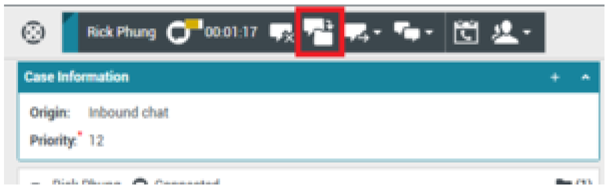
Workspace Desktop Edition Agent Use Cases

The *Workspace Desktop Edition Agent Help* provides help topics for logging into and using the various features of Workspace. The Chat topic describes how to use the features of the Chat interface.

Use Case - Agent puts chat on hold

An agent needs time to prepare a response to a contact in a chat session.

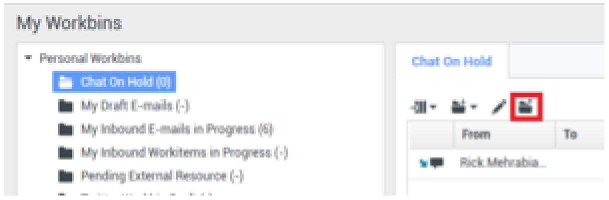
To avoid having the chat session time out, the agent can put the chat session on hold to keep it active in an interaction queue or workbin. To put a chat on hold, click **Place chat on hold**



Use Case - Agent accesses chat from Workbin, Queue, or History

Agents who have the privileges to pull interactions from queues, workbins, and shared workbins can select and open an in-progress (asynchronous) chat from one of the following locations in Workspace:

- The agent's personal workbin
- A group workbin to which the agent is a member
- My Interaction Queues
- My History view
- Contact History view
- Interaction Search view


My Workbins
Personal Workbins
Chat On Hold (0)
My Draft E-mails (-)
My Inbound E-mails in Progress (6)
My Inbound Workitems in Progress (-)
Pending External Resource (-)
Chat On Hold
From
To
Rick Mehrabia...

Related Resources

The *Workspace Desktop Edition User's Guide* (English only) provides detailed lessons for using all the features of Workspace. You might find the following lessons useful:

- Handle A Chat Interaction
- Handle A Chat Consultation
- Transfer A Chat Interaction
- Conference A Chat Interaction
- Main Window Basics
- Workspace Windows and Views

Related topics

- Functionality Overview
- Components, Features, and Controls

- Workspace Desktop Edition Help

Top 10 pages

1. Workspace Desktop Edition Help
2. Main Window
3. My Status
4. Contact Directory
5. Workbins
6. Functionality Overview
7. My Messages
8. Login
9. Voice Consultation
10. Components, Features, and Controls

Asynchronous Chat in Widgets

Async chat is supported in Genesys Widgets from the 9.0.002.06 release.

Configuring Async Chat in Genesys Widgets

The *Genesys Widgets Deployment Guide* describes how to deploy Widgets in your Genesys environment. All the Widgets documentation can be found here.

The following sections list the configuration options that you must set to enable Async Chat within Widgets:

- WebChat Configuration
- WebChatService Configuration
- WebChatService API Commands
- WebChatService API Events

Chat Business Process Sample

Overview

Chat Business Process (BP) Sample provides a sample workflow (in other words, a set of URS strategies combined into a business process) which demonstrates how to process chat interactions both for regular and asynchronous chat for different channels, including web chat, Apple Business Chat (ABC), WhatsApp. Chat BP Sample is not intended to be used in production deployments without careful evaluation and adjustments according to customer business requirements. While initially Chat BP Sample was developed for async chat only (which explains the naming convention for queues and strategies), it was adjusted to support regular chat as well.

Deployment

To deploy Chat BP Sample, following must be done:

1. Stop Interaction Server.
2. Upgrade the Interaction Server Database - execute script `AsyncChatSample_<database name>.sql` from "script" folder of Chat BP Sample IP.
3. Create new interaction custom properties; these are used in the workflow sample for **view** conditions. In Configuration (using GAX):
    - Navigate to the **Business Attributes** folder and create new **Business Attribute** with:
        - type - Custom
        - name - InteractionCustomProperties
        - display name - Interaction Custom Properties
    - Navigate to **Attribute Values** and create the following values (**Note:** that **name** and **display name** must be the same and each value must have a *translation* section in the **Annex** tab with a "translate-to" option):
        - `GCTI_Chat_AsyncStatus` with translate-to=async_status
        - `GCTI_Chat_AsyncCheckAt` with translate-to=async_check_at
4. Start Interaction Server.
5. In Interaction Routing Designer (IRD), import `ChatBusinessProcessSample.wie`, located in "workflow" folder of Chat BP Sample IP, and activate strategies.
6. Connect Chat Server endpoint with `async-chat-greet-queue`, queue.
7. In Workspace Desktop Edition application configuration options (see Configuring Asynchronous Chat in Workspace Desktop Edition for more information), set:
    - value `<media-type-name>.on-hold-queue` to `async-chat-return-queue` (where `<media-type-`

name> must be replaced with a media type value such as `chat`, or `whatsappsession`.

- value `workbin.chat.on-hold` to `async-chat-main-workbin`.

Testing through web chat

While Genesys Chat Widget only supports async mode for CometD connections, the testing of back-end components (Chat Server, workflow and Agent Desktop) can be done using the Chat Widget without enabling CometD. Launch the Chat Widget with the following userdata:

```
{ GCTI_Chat_AsyncMode: "true"}
```

Additionally, you can provide the following key-value pairs:

Value	Description
Chat_Async_RoutingTimeout: "5"	Allows the routing wait time to decrease to 5 seconds (default value in workflow is 120 seconds).
Chat_Async_WorkflowDebug: "true"	Forces the workflow to send debug chat messages about workflow execution.
Chat_Async_SendRichMessage: "true" **Introduced in: 8.5.309.12**	Forces the workflow to send the welcome rich message, "Welcome to Genesys chat".

Functional description

Functionally, Chat BP Sample can be divided into the following parts:

1. **Initialization.** The strategy (`async-chat-greet-strategy`):
   a. Sends greeting message to a customer
   b. Initializes routing parameters either for async or regular chat
   c. Moves interaction for processing into the main (`async-chat-main-queue`) queue
2. **Routing preparation and routing.** The strategy (`async-chat-main-strategy`):
   a. Checks if the routing rules are already assigned to the interaction (`Chat_Async_LastAgentAttempts` key/value pair in userdata)
   b. Tries to route to last handling agent for the specified number of attempts
   c. Continues routing to any available agent as soon as attempts are exhausted
3. **Processing interactions which are placed on hold by an agent.** The strategy (`async-chat-return-strategy`):
   a. Resets routing rules (`Chat_Async_LastAgentAttempts`)
   b. Tries to find last agent ID and place the interaction into agent's workbin (`async-chat-main-workbin`)

   c. Otherwise places the interaction into the main queue

4. **Processing stuck async interactions.** This happens only in very rare cases when chat session processing may get stuck due to failed components. The strategy (`async-chat-stuck-strategy`):
   a. Checks if chat session is still alive (by sending `GetSessionInfo` request) and places the interaction back into the main queue with a special async status value (to resume the routing)
   b. Otherwise moves the interaction to stop queue
5. **Implements transfer and conference routing.** The strategy (`async-chat-x-intercom-strategy`):
   a. Detects the type of target (from `IW_RoutingBasedTargetType`): **Skill**, **AgentGroup**, **InteractionQueue**.
   b. Routes the interaction according to detected type.
   c. Demonstrates how to send the ESP message correctly from the internal auxiliary (`InteractionSubtype=InternalConferenceInvite`) interaction.

> **Important**
> `async-chat-x-intercom-strategy` is introduced in **8.5.309.12**; the required Chat Server version is **8.5.312.10** or higher.

6. **Stopping of the completed chat**. The strategy (`async-chat-stop-strategy`):
   a. Stops the interaction
   b. Notifies UCS

Important notes

- The main processing queue (`async-chat-main-queue`) contains several views (used for selecting interaction for the processing by a strategy) with different conditions:
  - `async-chat-main-proc-view` fetches chat sessions which are not in a dormant state, thus locking the dormant chat sessions in the queue until a qualified event changes the status (these events can be a new interaction, a message from a customer, idle control notification, agent desktop failure, and others).
  - `async-chat-main-check-view` fetches stuck chat sessions.
  - `async-chat-main-stop-view` fetches chat interactions for which sessions were stopped by Chat Server while the interaction was sitting in the queue.
- In this Chat BP Sample, the agent workbin is associated with the main routing queue thus forcing the router to apply the same processing rules to workbin interactions. If you need to keep interactions in the agent workbin without forcing routing upon a qualified event in a chat session, you need to modify the workflow accordingly.
- In case of failure, strategies move the interaction to stop queue (`async-chat-stop-queue`). However, in production the workflow must consider making a few attempts before finally surrendering the interaction processing any further. For example, workflow may send an ESP message a few times during a Chat Server switchover and when chat session is being moved to another Chat Server. Chat BP Sample does not provide this logic for simplification reasons.

- Chat BP Sample contains `async-chat-media-proc-sub` subroutine which provides channel-specific logic for interaction processing. For ABC and WhatsApp channels, the strategy verifies if a customer's nickname ("_msg_ProfileNickname") is already assigned for the contact, and assigns the default nickname if needed.
- You can send Rich Text Messages from the workflow. For more information, see the **Example** in the "Key-value collection format specification" section of the How to send ESP requests to Chat Session from Workflow topic.

Rich Messaging Support

Overview

Genesys chat solution provides the ability to use structured messages (in other words, Rich Messaging) across various chat channels, including:

Channel	Components	Channel name
Web chat	Genesys Mobile Services (GMS) (min version required 8.5.201.04) and Chat Widget (for supported elements, see Rich Messaging in the Genesys Widgets Deployment Guide)	`genesys-chat`
Apple Business Chat (ABC)	Digital Messaging Server (DMS) and ABC driver (see Deploying Apple Business Chat in the Apple Business Chat Guide)	`applebc-session`
WhatsApp	DMS and Genesys Driver for use with Genesys Hub (see Deploying WhatsApp in the WhatsApp Guide)	`genesys-chat`

Important

Support for Rich Messaging varies by channel based on what each channel service provider supports and what is implemented in Genesys Engage. Not every Rich Messaging element is supported in all channels.

Additionally, the following components are also involved:

Component	Purpose
Chat Server	Conduct chat session. Min version required 8.5.109.06.
eServices Manager	An authoring tool for creating standard responses which can contain structured messages. Graphical editing capabilities are provided for some channels together with the ability to provide raw (for example JSON) representation of a structured message.
Bot Gateway Server	A chat bot deployment platform that provides an API for bots to use either standard responses with structured messages, or send Rich Messaging

Component	Purpose
	containing native or normalized JSON format.
Workspace Desktop Edition (WDE)	WDE is extended with Rich Messaging functionality for specific channels via a plugin architecture. Plugins are currently supported for ABC and WhatsApp only.
Workflow	Workflow allows rich messages to be sent through the `EventAttributes` parameter.

Important

For backward-compatibility, structured messages are always accompanied with so called "fallback" plain text messages. So, if an application representing a chat participant does not know how to process the structured content, it will continue to use a regular plain text message (which is supplied together with structured content).

How to deploy and use structured messages

In order to start using structured messages, you'll need to:

1. Configure the channel; see, Configuring structured messages.
2. Deploy eServices Manager; see, Install eServices Manager.
3. Using eServices Manager, create standard responses with structured content:
    - For ABC and WhatsApp, see Structured Messages.
    - For web chat (Chat Widget) provide the raw JSON.
4. Use standard responses:
    - From WDE, for Apple Business Chat, see Standard Response Library, and for WhatsApp, see Standard Response Library.
    - In bots developed for BGS, see Bot implementation guidelines.
5. Alternatively, you can send structured messages from a custom desktop (or custom virtual agent) through the Chat Server API in `EventAttributes`.

Important

When creating a standard response in eServices Manager, the fallback text messages must be provided in the **Plain Text Part** window.

Chat Widget support

In general, the same directions described above apply to Rich Messaging elements supported by Chat Widget, with a few special notes:

- When creating the `MediaOrigin` business attribute in configuration management, `genesys-chat` must be used as a name and the **[rich-media-types]** section must be populated with elements supported by the Widget.
- Within the standard response for eServices Manager, raw JSON must be provided. At this time, no graphical authoring tool is provided.
- Chat Widget Rich Messaging can currently only be used from bots (running through BGS) or by a custom desktop.