Chat Server Administration Guide

Genesys Engage Chat 8.5.1

1/25/2022

Table of Contents

Chat Server Administration

The following topics provide information for Chat Server administrators:

Feature	Description
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 Statistics	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.

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=").

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.
  - limit-for-sessions—Maximum number of concurrent chat sessions.

    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`
reset	Resets all counters to zero.	`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.

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 UCS 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 currently 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.

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

- 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.

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

Chat Server supports Apache Cassandra 2.2. Download the latest stable release of Cassandra 2.2.x 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 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_AE
 #,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.

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}[

Order	Name	Regular Expression
		-.]?\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 configured in the `inactivity-control` section. To enable this functionality set the enabled option to `true`.

If inactivity monitoring is enabled, it works as following:

- Chat Server enables inactivity monitoring only if at least one customer and one agent are participating in the chat session. If the inactivity monitoring is enabled 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.
code	`country`	optional	ISO 3166 code	Only used if the `language` option is specified. The value is converted to uppercase when

Section	Option	Mandatory	Possible values	Notes
				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`.

Matching Contact Attributes

When a home user asks to open a chat session, the web interface gets him or her to fill in some identifying information, such as e-mail address, phone number, first name, last name, and so on.

This identifying information becomes a part of the *user data* that is associated with the interaction. The web interface relays this user data to Chat Server, and Chat Server sends it to UCS.

UCS then looks to see if the home user matches any of the people that it has represented as contacts in its database. It does this according to the following algorithm:

Attribute Name	Search Order
EmailAddress	0
PhoneNumber	1
FirstName	2
LastName	2

UCS is hard-coded to use this algorithm with interactions coming from Genesys media servers, namely e-mail, chat, and callback interactions. For other media the algorithm can be customized.

So 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. (For details on the structure of this part of the UCS database, see the "Contact Package" chapter in *eServices 8.0 Selected Conceptual Data Models for the UCS Database*.) 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`.

If UCS finds no matching contact, it creates a new one using the user data (see Contact Identification and Contact Creation for more information).

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 e-mail 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 e-mail 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 e-mail 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. However, it is not possible to avoid the use by the system of user data to generate the message prompt (`SpeveJ` in the example).

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/ORS 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.

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

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` – could be used for any custom purpose (completely transparent for Chat Server).
NoticeText	optional	Any arbitrary text.
Nickname and Visibility		The same as in the `Message` Method.

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. 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.
message-close	optional	
timeout-alert	optional	
timeout-alert2	optional	Available starting with Chat Server release 8.5.107.
timeout-close	optional	

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
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.

Chat Server Reporting Statistics

Overview

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

- Chat session end reason codes – always attached, no configuration required.
- Chat session transcript statistics – controlled by the attach-session-statistics option.

> **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 End Reason Codes

The following reason codes describe what triggered the end of a chat session and how it was triggered:

- `csg_SessionEndedBy` – The type of participant that triggered the chat session closure.
- `csg_SessionEndedReason` - The description of how a chat session was closed.
- `csg_SessionEndedAgent` - 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.

csg_SessionEndedBy		
Value	**Description**	**Notes**
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:

csg_SessionEndedBy		
		• 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 is closed because the actor sent a Release request.
SYSTEM	Denotes a server/system	See the `csg_SessionEndedReason` table for possible reasons.

csg_SessionEndedReason		
Value	**Description**	**Possible values for `csg_SessionEndedBy`**
DISCONNECT	The participant left due to a disconnect (basic protocol) or a flex timeout expiration (denotes disconnect in flex protocol).	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).	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).	AGENT, SUPERVISOR, BOT
INACTIVE	Chat Server closed a chat session due to activated inactivity control monitoring.	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`).	SYSTEM

csg_SessionEndedAgent		
Value	**Description**	**Notes**
ABSENT	Session considered as abandoned.	No agent (in other words, not-bot participant visible to client) ever joins chat session.

csg_SessionEndedAgent		
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 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.

Chat Session Transcript Statistics

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

In the General Statistics table, an agent means both an agent and a supervisor, when either of those is visible to a customer. For example, it does not count/include an activity for an agent who is coaching another agent, or for a supervisor who monitors the session silently.

General Statistics

General Statistics	
KVP key name implemented	**Description**
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.
csg_MessagesFromAgentsSize	The total size of all messages sent by agents.
csg_MessagesFromCustomersCount	The total number of messages sent by customers.
csg_MessagesFromCustomersSize	The total size of all messages sent by customers.
csg_PartiesAsAgentCount	The number of parties that participated in a session as agents. Tip 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). Tip 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_PartiesAsMonitorCount	The number of parties that participated in a session

General Statistics	
	in the monitoring mode (for example, a supervisor join with the INT visibility). Tip 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_SessionTotalTime	The total duration of a chat session from the time it was created until it was completely finished/closed in Chat Server. Tip This does not include the time between `Chat Session End` and `Mark Done` as the interaction can still be handled by an agent.
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. Tip 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).
csg_SessionUntilFirstReplyTime	The period of time until the first agent submits the first visible to a customer greeting/message into a chat session.
csg_SessionWithCustomerTime	The period of time a customer is in a chat session.

Extended (wait-reply) Statistics

Extended (wait-reply) statistics	
KVP key name implemented	**Description**
cse_AgentReplyTotalCount	The number of times an agent replied to a customer.
cse_AgentReplyMaxTime	The maximum time (in seconds) an agent spent on replying to a customer.
cse_AgentReplyTotalTime	The total time (in seconds) an agent spent on replying to a customer.
cse_AgentWaitTotalCount	The number of times an agent waited for replies from a customer.
cse_AgentWaitMaxTime	The maximum time (in seconds) an agent spent on waiting the reply from a customer.
cse_AgentWaitTotalTime	The total time (in seconds) an agent spent on waiting the reply from a customer.
cse_CustomerReplyTotalCount	The number of times a customer replied to an agent.

Extended (wait-reply) statistics	
cse_CustomerReplyMaxTime	The maximum time (in seconds) a customer spent on replying to an agent.
cse_CustomerReplyTotalTime	The total time (in seconds) a customer spent on replying to an agent.
cse_CustomerWaitTotalCount	The number of times a customer waited for the reply from an agent.
cse_CustomerWaitMaxTime	The maximum time (in seconds) a customer spent on waiting the reply from an agent.
cse_CustomerWaitTotalTime	The total time (in seconds) a customer spent on waiting the reply from an agent.

The terms `wait` and `reply` are defined as follows:

- `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.

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

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 commands: new notice type `SYS_COMMAND`. Flex transcript events have

ClientVersion	Chat Server version	PSDK version	Description (features added or changed)
			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 allows to 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 available	not available	not available

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 available	not available	not available

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 "B0T". Chat Server attaches another key/value pair **`GCTI_Chat_PartyStyle="B0T"`** to the `newParty'` event in basic protocol chat transcript and **`GCTI_SYSTEM/party-into/style"="B0T"`** 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, 8.5.303 for eventAttributes	8.5.118	not available	not available	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	8.5.303	not available	not available	not available

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 available	not available	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 available	not available	not available