



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Client Samples

Chat Version 2 CometD Sample

Contents

- 1 Chat Version 2 CometD Sample
 - 1.1 Prerequisites
 - 1.2 Run the GMS Chat CometD Sample
 - 1.3 Implementation Details
 - 1.4 Disclaimer

Chat Version 2 CometD Sample

This code sample shows how to implement a WebSocket client application for GMS Chat Version 2 using CometD. In this example, the client application opens a CometD connection, submits a chat request, and starts receiving notifications. Once the chat session is over, the application closes the CometD channel.

The client application uses CometD to connect the Chat Server via GMS.

- GMS initiates the Chat session by sending a request to the Chat Server.
- The Chat Server connects the agent and publishes notifications to the client application through GMS CometD channels.
- The client application submits chat messages and receives notifications through GMS CometD channels.

This code sample uses the [CometD Project](#) for Javascript, in addition to [Vue.js](#) and [Metro4](#), for UI components.

[Link to video](#)

Prerequisites

- Genesys Mobile Services 8.5.2+
- Refer to the [Prerequisites for the CometD API](#).
- Configure [Digital Channels](#).
- Create a chat service channel, for example, customer-support.
- Download the [Chat V2 CometD Sample](#).

Run the GMS Chat CometD Sample

Chat V2 CometD Example

⚠ GMS Server Offline

GMS Cometd URL

For example, `http://gms-host:gms-port/genesys/cometd`

GMS Chat Service Name

For example, `customer-support`

Chat nickname

For example, `John Doe`

Start chat session

⚠ Chat Server Offline ⚠ No chat session

Send

- Unzip the chat sample zip file.

- In the `chatv2-cometD-sample` directory, run the `npm install` command.
- Open the `index.html` file with a browser.

Chat V2 CometD Example

✓ GMS Server Online

GMS Cometd URL

For example, `http://gms-host:gms-port/genesys/cometd`

GMS Chat Service Name

For example, `customer-support`

Chat nickname (required)

For example, `John Doe`

✓ Chat Server Online ✓ Chat started

[End chat session](#)

The screenshot shows a chat window with a scrollable message history. The messages are as follows:

- Joined** (system message)
- 20 May 3:50 PM** (timestamp)
- system** (sender): "(Easy) Agent will be shortly with you..."
- a1001** (sender): "Joined" (timestamp: 20 May 3:51 PM)
- a1001** (sender): "Hello! How can I help you?" (timestamp: 20 May 3:51 PM)
- John Doe** (sender): "I have a question about my insurance." (timestamp: 20 May 3:51 PM)

At the bottom of the chat window is a text input field with a cursor, a close button (X), and a "Send" button.

- Fill in the form, and then click **Start chat session**. You can start submitting chat messages.

Implementation Details

Step 1. Get started with CometD

When you click the **Start Chat** button, the Vuejs chatApp component (or chatUi) calls the chatV2Start function which performs the following actions:

1. Initialize and configure a CometD instance. See [CometD documentation](#) for further details.
2. Register callback listeners for each channel that submits notifications.
 - CometD Meta Channels
 - GMS CometD Chat service channels which are responsible for publishing events and notifications. In this sample, the service used is "customer-support". As a result, the channel for this service is: /service/chatV2/customer-support.
3. Handshake.

```
function chatV2Start(url, channel) {  
  
    chatUi.runToast("Connecting", 'info');  
    // Instantiate CometD  
    // For each chat session, open a CometD connection  
    objCometD = new org.cometd.CometD();  
    // Optionally, install an extension.  
    objCometD.registerExtension('timestamp', new org.cometd.TimeStampExtension());  
  
    // Enable WebSockets  
    objCometD.websocketEnabled = objCometDSettings.websocketEnabled;  
  
    // Provide your GMS CometD Endpoint  
    objCometDSettings.cometURL = url;  
    objCometD.configure({  
        url: url,  
        logLevel: "info"  
    });  
  
    /// Add listeners for cometD meta endpoints  
    objCometD.addListener('/meta/handshake', fHandshake);  
    objCometD.addListener('/meta/connect', fConnect);  
    objCometD.addListener('/meta/disconnect', fDisconnect);  
    objCometD.addListener('/meta/publish', fPublish);  
    // Listen to chat notifications  
    objCometDSettings.channel = channel;  
    objCometD.addListener(objCometDSettings.channel, refresh);  
  
    // Now, handshake  
    objCometD.handshake();  
}
```

Important

The above CometD code is a one-time setup only during the lifecycle of an active CometD connection.

Step 2. Handle ChatV2 API Queries

To simplify the code, this sample calls the request method for CometD requests once the chat session is started. The supported operations match the API queries listed in the [Chat V2 CometD API](#) reference page, for example:

- sendMessage
- disconnect
- requestNotifications

The request function always provides the secureKey parameter which is available in the message responses as soon as the chat session is started.

```
function request (operation, params) {
  if (objCometD && boolCometDConnected) {
    var data = {
      operation: operation,
      secureKey: objSessionData.secureKey,
    };
    var finalData = $.extend(data, params);
    objCometD.publish(objCometDSettings.channel, finalData);
  } else {
    chatUi.runToast("CometD is not connected!", "alert");
  }
}
```

Step 3. Implement CometD Meta Callbacks

fHandshake

The CometD library calls this function automatically to submit the result of the Handshake operation.

- If successful, the GMS communication channel is ready and the sample submits a requestChat request.
 - If the sample detects a reconnection, it sends a requestNotifications request to make sure that you get all chat notifications. Note that the handshake callback function gets automatically called by CometD during any failure trying to re-establish the connection to GMS.
- If not, the Handshake is failed and the sample needs to disconnect CometD.

Important

In this sample, the callback function counts the number of handshake exceptions and disconnects from CometD only when reaching the `intHandshakeExceptionLimit` limit. By default, CometD makes several attempts if you don't close the connection. This ensures to display a connection error only if the handshake keeps failing.

```
function fHandshake(response) {
  console.log('fHandshake', response);
```

```
if (response.successful === false) {
    if (++intHandshakeExceptionCount === intHandshakeExceptionLimit) {
        /// Too many exceptions, close CometD connection
        disconnectFromGMS();
        intHandshakeExceptionCount = 0;
        /// Update UI
        chatUi.stopWaiting();
        chatUi.runToast("Handshake failed", 'alert');
    }
} else if (response.successful === true) {
    chatUi.runToast("Handshake success", 'info');

    boolCometDConnected = true;
    chatUi.boolGMSConnected = true;
    intHandshakeExceptionCount = 0;

    if (response.secureKey && boolRestoring) {
        chatUi.runToast("Restoring notifications", 'info');
        /// If the handshake is successful, submit a request notifications request
        /// otherwise client messages are not exchanged properly.
        console.log("request notifications handshake");
        request("requestNotifications", {
            secureKey: response.secureKey,
            transcriptPosition: intLastTranscriptPosition,
            message: "" // get current text in text input
        });
        boolRestoring = false;
    } else {
        chatUi.stopWaiting();
        chatUi.runToast("Start chat session", 'info');
        /// Start a chat session
        var requestChatData = {
            operation: "requestChat",
            nickname: chatUi.strNickname,
            subject: "Test CometD Chat v2",
            text: "",
            "userData": {
                "interests": "javascript"
            }
        }
        objCometD.publish(objCometDSettings.channel, requestChatData);
    }
}
}
```

fConnect

When implementing Chat V2 using CometD, you can get disconnected from GMS and from the Chat Server, which interrupts the chat session.

- CometD connection status is tracked with boolCometDConnected in the **fConnect()** function.
- Chat Server connection status is tracked with boolChatServerOffline in the **fDisconnect()** function.

In the **fConnect()** callback, the sample submits a requestNotifications request if the Chat Server is disconnected. Otherwise, if the Chat Server was previously connected and if the response is not successful (response.successful === false), it means that the /meta/connect request failed. In

this scenario, the sample checks for the error received and, if the error code is 402, it processes it as if the Chat Server is disconnected.

Based on the advice received in the response, the CometD library submits the handshake request again, if needed, and does not require that you take care of it.

```
function fConnect(response) {
    if (!objCometD || objCometD.isDisconnected()) {
        boolCometDConnected = false;
        chatUi.runToast("CometD is no longer connected!", "alert");
        return;
    }

    boolCometDConnected = response.successful == true;

    // If chat was disconnected but CometD has reconnected
    if (boolDisconnected && boolCometDConnected) {
        boolDisconnected = false;
        if (!chatUi.boolChatServerOnline) {
            /// Requesting chat messages$
            request("requestNotifications", {
                transcriptPosition: intLastTranscriptPosition
            });
            /// Updating the chat server status
            chatUi.boolChatServerOnline = true;
            chatUi.runToast("CometD reconnected", "info");
        }
    } else if (!boolDisconnected && !boolCometDConnected) {
        if (response.error !== "402:: Unknown client") {
            boolDisconnected = true;
            chatUi.boolGMSConnected = false;
            /// Notify disconnection status
            chatUi.runToast("CometD is disconnected! " + response.error, "alert");
        }
    }
}
```

fDisconnect

When implementing Chat V2 using CometD, the sample or the agent can submit a GMS disconnect request, which interrupts the chat session and also notifies the '/meta/disconnect' channel.

- The CometD connection status is tracked with boolCometDConnected in the **fConnect()** function.
- The Chat Server connection status is tracked with boolChatServerOffline in the **fDisconnect()** function.

The **fDisconnect** function is called when CometD is disconnected. First, it checks if it needs to re-establish the connection, and if so, it submits a /handshake request.

Important

To get notifications after the reconnection, the sample needs to request notifications again. This action is performed in the handshake handler.

```
function fDisconnect(response) {
    console.log('disconnect', response);
    if (response.successful) {
        boolDisconnected = true;
        chatUi.boolChatServerOnline = false;
        chatUi.boolGMSConnected = false;
        chatUi.boolSessionActive = false;
        /// if CometD is not set to false,
        /// trying to reconnect
        if (objCometD) {
            chatUi.runToast("CometD - disconnection", "alert");
            boolRestoring = true;
            objCometD.handshake();
        } else {
            objSessionData.secureKey = "";
            chatUi.runToast("CometD - disconnection", "info");
        }
    } else {
        chatUi.runToast("CometD - disconnection " + response.error, "error");
    }
}
```

Step 4. Process Notifications

Refresh

The refresh handler receives chat notifications either received after a successful requestChat query or requested with the requestNotifications query. In addition to the status code, it receives all of the transcripts and events related to chat messages. In the refresh handler, the sample checks the response status code and updates accordingly.

- An API **statusCode** value of 0 indicates that the operation was successful and all fields in the response have valid values.
 - In this scenario, the sample retrieves the secureKey parameter required to submit further Chat V2 requests during the chat session.
- An API **statusCode** value of 1 or 2 indicates that there was an exception and Chat Server is offline.

Further processing of the chat messages is performed in the **parseTranscript()** function detailed in a separate section.

```
function refresh(response) {
    console.log('refresh', response, boolCometDConnected);
    chatUi.runToast("Received notification", "info");
    if (boolCometDConnected) {
        response = response.data;

        /// Make sure to retrieve session parameters
        /// only secureKey is required
        if (response.secureKey) objSessionData.secureKey = response.secureKey;

        if (response.statusCode === 2 && response.errors && response.errors.length > 0) {
            /// Process the received errors
            chatUi.boolSessionActive = false;
            var errorToDisplay = "Chat API error: <br/>";
            response.errors.forEach(element => { errorToDisplay += element.advice + "<br/>"
        });
    }
}
```

```
        chatUi.runDialog(errorToDisplay, "alert");
    }

    // Prevents situation where you can't end an expired chat session
    if (response.statusCode === 0) {

        if (!chatUi.boolChatServerOnline) {
            /// Chat Server is back online
            chatUi.boolChatServerOnline = true;
            chatUi.runToast("Chat server is back online", "info");
        }

        if (response.messages.length > 0) {
            // Parse the chat transcript
            parseTranscript(response);
        }

    } else if (response.statusCode === 1) {
        chatUi.boolChatServerOnline = false;
        chatUi.runToast("Chat server is offline", "warn");
    }
}
}
```

Parse Transcript

In this sample, the **ParseTranscript** function checks all the messages for the current chat session. To avoid publishing multiple times the same messages, it checks the index of each message and publishes only new ones. Additionally, if the chat session has ended, the function also calls the **disconnectFromGMS()** function to close the GMS CometD connection.

```
function parseTranscript(transcript) {
    // Update chat display if needed
    $.each(transcript.messages || [], function () {
        /// TIP: Use the index to check if you previously processed this message
        if (this.index > intLastTranscriptPosition) {
            switch (this.type) {
                case 'ParticipantJoined':
                    chatUi.displayMessage(this.from.nickname, '<i> Joined </i>');
                    break;
                case 'Message':
                    if (this.from.type !== 'Client') {
                        // In this sample, the chat widget already displayed
                        // the client message
                        chatUi.displayMessage(this.from.nickname, this.text);
                    }
                    break;
                case 'ParticipantLeft': chatUi.displayMessage(this.from.nickname, '<i> Left
</i>');
                    break;
                default: console.log(this);
            }
            intLastTranscriptPosition = this.index;
        }
    });
    // Check if the session has ended
    if (transcript.chatEnded === true) {
        chatUi.boolSessionActive = false;
        disconnectFromGMS();
    }
}
```

```
    }  
}  
  
function disconnectFromGMS() {  
    console.log("disconnect from GMS");  
  
    /// Clean up  
    objSessionData.secureKey = "";  
    intLastTranscriptPosition = -1;  
  
    /// Stop cometD  
    if (boolCometDConnected && objCometD) {  
        objCometD.disconnect();  
        objCometD = false;  
        boolCometDConnected = false;  
    }  
}
```

Step 5. Terminate the Chat session

To terminate the Chat session, if the user clicks the **End Chat Session** button, the sample sends a Chat V2 disconnect request.

As a result, the sample disconnects from CometD after receiving a response notification with `chatEnded = true`. Note that it also receives a meta disconnect event processed in the `fDisconnect` handler.

```
/// Terminate the chat session before disconnecting CometD  
function terminateChatSession() {  
    chatUi.runToast("Terminate Chat session...", "warn");  
    intLastTranscriptPosition = -1;  
    request('disconnect', {});  
}
```

Disclaimer

THIS CODE IS PROVIDED BY GENESYS TELECOMMUNICATIONS LABORATORIES, INC. ("GENESYS") "AS IS" WITHOUT ANY WARRANTY OF ANY KIND. GENESYS HEREBY DISCLAIMS ALL EXPRESS, IMPLIED, OR STATUTORY CONDITIONS, REPRESENTATIONS AND WARRANTIES WITH RESPECT TO THIS CODE (OR ANY PART THEREOF), INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. GENESYS AND ITS SUPPLIERS SHALL NOT BE LIABLE FOR ANY DAMAGE SUFFERED AS A RESULT OF USING THIS CODE. IN NO EVENT SHALL GENESYS AND ITS SUPPLIERS BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, ECONOMIC, INCIDENTAL, OR SPECIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, ANY LOST REVENUES OR PROFITS).