Enterprise Chat and Email Mobile Chat Template Developer's Guide, Release 12.6(1)

For Unified Contact Center Enterprise and Packaged Contact Center Enterprise

First Published: May, 2021

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Enterprise Chat and Email Mobile Chat Template Developer’s Guide: For Unified Contact Center Enterprise. May 6, 2021

Contents

Preface

Welcome to the Enterprise Chat and Email (ECE) feature, which provides multichannel interaction software used by businesses all over the world as a core component to the Unified Contact Center Enterprise product line. ECE offers a unified suite of the industry's best applications for chat and email interaction management to enable a blended agent for handling of web chat, email and voice interactions.

About This Guide

Enterprise Chat and Email Mobile Chat Template Developer's Guide provides development resources capable of leveraging the JavaScript Library to build custom chat and callback user experiences leveraging the power of the ECE platform.

Related Documents

The latest versions of all Cisco documentation can be found online at https://www.cisco.com

Subject	Link
Complete documentation for Enterprise Chat and Email, for both Cisco Unified Contact Center Enterprise (UCCE) and Cisco Packaged Contact Center Enterprise (PCCE)	https://www.cisco.com/c/en/us/support/customer-collaboration/cisco-enterprise-chat-email/tsd-products-support-series-home.html

Communications, Services, and Additional Information

- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit Cisco Marketplace.
- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.

Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.

Field Alerts and Field Notices

Cisco can modify its products or determine key processes to be important. These changes are announced through use of the Cisco Field Alerts and Cisco Field Notices. You can register to receive Field Alerts and Field Notices through the Product Alert Tool on Cisco.com. This tool enables you to create a profile to receive announcements by selecting all products of interest.

Sign in www.cisco.com and then access the tool at https://www.cisco.com/c/cisco/support/notifications.html.

Documentation Feedback

To provide comments about this document, send an email message to the following address: contactcenterproducts_docfeedback@cisco.com

We appreciate your comments.

Document Conventions

This guide uses the following typographical conventions.

Convention	Indicates
Italic	Emphasis. Or the title of a published document.
Bold	Labels of items on the user interface, such as buttons, boxes, and lists. Or text that must be typed by the user.
`Monospace`	The name of a file or folder, a database table column or value, or a command.
Variable	User-specific text; varies from one user or installation to another.

Document conventions

1 Basics

The Mobile SDK extends the reach of an ECE deployment, offering eGain Solve for Cisco-enabled engagement options to mobile users through existing or new apps on the iOS and Android platforms.

Key Terms and Concepts

- **Chat activity:** An activity created for a chat session between a customer and an agent. A chat is a real time interaction between an agent and a customer where they exchange text messages. As part of a chat, agents can also push web pages to customers. The chat is routed to a queue, and a message is sent to Unified CCE. Unified CCE processes the activity and assigns the chat to an available agent.
- **Template sets:** Template sets consist of HTML, CSS (cascading style sheets) and JS (JavaScript) files. The CSS files control the look and feel of the customer's Chat and Callback area. The JS files contain the logic used to render data in the Chat and Callback area. Templates are also used to determine the type of information collected on the web form and used to identify the customer (e.g. name, email address, phone number). You can also compose messages that the customer will see under certain circumstances (e.g. if they request a chat session out of hours). For more information about template sets, see *Enterprise Chat and Email Guide to Chat and Collaboration Resources*.
- **Entry points:** An entry point defines the starting point from which customers initiate chat and web callback interactions. Every help link on a website is mapped to an entry point and each entry point has a queue associated with it. Different template sets can be used with the same entry point. The queue is used to route activities to agents while the template set determines the look and feel of the different pages displayed to chat customers. For more details, see *Enterprise Chat and Email Administrator's Guide to Chat and Collaboration Resources*.
- **Secure Chat:** Secure Chat allows chat entry points to transfer customer context information from the company website to the application through SAML. This allows customers who are already recognized on the company website to use a single sign-on enabled entry point to chat with a customer without having to provide redundant information. This feature is available for auto-login configuration only.
  - For details about configuring templates for Android to use secure chat, see "Enabling Mobile Chat for Chat Customer SSO" on page 16.
  - For details about configuring templates for iOS to use secure chat, see "Enabling Mobile Chat for Chat Customer SSO" on page 24.
  - For more details about configuring Chat Customer Single Sign-On, see *Enterprise Chat and Email Administrator's Guide to Administration Console*.
- **Data Masking for Chat:** Data masking allows businesses to ensure that sensitive information, like credit card numbers, Social Security Numbers, bank account numbers, etc. is not transmitted from the system to the customers and vice versa. If the customer and agent do add any sensitive data in the email content and chat messages, all such data is masked before it is displayed to customers and agents and before it is stored in the system. For details about setting up data masking, see the *Enterprise Chat and Email Administrator's Guide to Administration Console*.
- **Chat Attachments:** Customers and agents can send files to each other during a chat. Once configured by an administrator, customers and agents can browse to a file and attach it to their chat messages. Customers can also drag and drop files into the chat text editor. For more details about the types of files that can be allowed or blocked, or how to set up chat attachments, see the *Enterprise Chat and Email Administrator's Guide to Administration Console*.

Configuring the System for Chat

- It is recommended that you configure your system for Chat before adjusting the template files. For a comprehensive list of configurations that should be performed, see the *Enterprise Chat and Email Administrator's Guide to Administration Console*.

2 Aria Chat Template Integration with Android

Using the Chat Entry Point URL

An entry point defines the starting point from which customers initiate chat interactions. Every chat help link on a website is mapped to an entry point. Each entry point has a queue associated with it and the queue is used to route chats to agents. For more details, see *Enterprise Chat and Email Administrator's Guide to Chat and Collaboration Resources*.

In order to perform network operations such as accessing entry point URL in android application, the application manifest file must include the following permission:

```
android.permission.INTERNET
```

for ex: Following line needs to be added in manifest file

```
<uses-permission android:name="android.permission.INTERNET" />
```

To use the entry point URL:

1. Create a web view and load the chat URL in the web view. This allows chat to be embedded in the android app.
2. Add the `android.permission.INTERNET` permission to the manifest file of an android app.
3. The following settings must be set in the WebSettings object for the WebView in which the chat is launched:

   ```
   setDomStorageEnabled(true);
   setAppCacheEnabled(false);
   setCacheMode(WebSettings.LOAD_NO_CACHE);
   setJavaScriptEnabled(true);
   setAllowFileAccess(true)
   ```

4. Next, in the Administration Console, create the entry point and obtain the URL. For details, see *Enterprise Chat and Email Administrator's Guide to Chat and Collaboration Resources*. An example URL is such:

   `http(s)://`*server_name*`/system/templates/chat/`*template_name*`/index.html?entryPointId=`*entrypoint_id*`&templateName=`*template_name*`&ver=v11&locale=locale`

   Where the *server_name* is the web server, the *template_name* is the chat template deployed on the application server, the *entrypoint_id* matches the id of the desired entry point for incoming chat activities, and the locale is the language code and country code, such as `en_us`.
5. Copy the URL from the Administration Console and add it to your webpage.
6. If you wish to make the entry point proactive, the following parameters should be appended to the URL with values from an Offers Banner URL with the eGain Solve for Cisco Offers add-on:

   ```
   aId=
   sId=
   uId=
   ```

Interface Between the Android App and the Chat Template

To bind a new interface between your JavaScript and Android code, call `addJavascriptInterface()`, pass it a class instance to bind to your JavaScript code and an interface name that the JavaScript can call to access the class. This allows the Java Object's methods to be accessed from JavaScript. For example:

```
webView.addJavascriptInterface(new WebAppInterface(this), "AndroidInterface");
```

At this point, the web application has access to the WebAppInterface class. To call an interface method, create an object of the implemented class to refer to the function using that object.

Calling Java Methods in Android Application from Template Code

The following Java methods in android application gets called from chat template code:

Android Interface Method	Description
public String getCustomerData()	Returns a string representation of JSON Object containing customer values.
public void onChatStarted()	Event raised by template when chat session is started successfully
public void onChatClosed()	Event rased by template when user clicks **end chat** button on template.
public void onChatEnded()	Event raised by template when chat session ends.
public void saveTranscript(Stringdata)	This method is called by the Aria template when a user selects the option to save transcript. Chat transcripts are displayed in HTML format.
public void printTranscript(Stringdata)	This method is called by the Aria template when a user selects the option to print transcript. Chat transcripts are displayed in HTML format.
public void setSId(String sid)	This method is called by the Aria template when a chat connection is successful. App should save the SID passed as it will be used while downloading attachments (see attachments section).

Calling Javascript Functions from Android Application Code

The following javascript functions in chat template can be called from android application webview:

MobileInterface	Description
CloseChat()	Method call for closing chat. This will be raised by app for closing chat from within the app.
ShowCloseButton(boolean)	Method for showing/hiding close button inside template. By default button is visible.

Attachments

During a chat session, file attachments can be exchanged between customers and agents. Administrators can enable attachments for chat and specify the maximum allowed size for chat attachments. Additionally, administrators can specify the file types that can be attached to emails and chat messages. For more details, see *Enterprise Chat and Email Administrator's Guide to Administration Console*.

Permissions

The following permissions are required to be added in manifest file for android application:

- `android.permission.WRITE_EXTERNAL_STORAGE`
- `android.permission.ACCESS_DOWNLOAD_MANAGER`

The following setting must be set in the WebSettings object for the Chat application WebView

```
setAllowFileAccess(true);
```

Uploading Attachments

Implement `onShowFileChooser` method inside `WebChromeClient` to allow a user to upload a file from their device.

Downloading Attachments

To allow for attachments to be downloaded during the chat session, the following is required.

- `onDownloadAttachments/ handleFileResponse: Webview` should add `setDownloadListener and override onDownloadStart method of DownloadListener.`
  - `public void onDownloadStart(String url, String userAgent, String contentDisposition, String mimetype, long contentLength)`
- In the `onDownloadStart` method, make a `POST` request to the URL. In the request, set `"Content-Type"` header as `"application/x-www-form-urlencoded"` and pass `"sid"` parameter in the form-data. To fetch the value of sid, implement `setSId` method as `JavascriptInterface`. Aria template calls this method on chat connection success passing SID to it.

  ```
  @JavascriptInterface
  public void setSId(String sid){}
  ```

Displaying the Attachment Upload Window

The dialogue window that appears when uploading attachments can be displayed when attempting to upload a file.

To display the attachment upload window:

1. Set `WebChromeClient` for the `WebView` using `chatView.setWebChromeClient(new WebChromeClient());`

2. Ensure `WebChromeClient` implements the method `onShowFileChooser`.

Handling Dialog Windows

Handling Attachment Upload Dialogs

To handle attachment upload dialog:

- Set `WebChromeClient` for the `WebView` using `chatView.setWebChromeClient(new WebChromeClient());`

  `WebChromeClient` needs to implement method `onShowFileChooser`

Handling JavaScript window.open Events

The app can handle JavaScript `window.open` events, which include links that are opened from a chat window, such as the **FAQ** link and the alternate engagement options.

To handle JavaScript window.open events:

- Implement `shouldOverrideUrlLoading (WebView view, String url)`.

  The app can handle these scenarios differently based on the URL value. For example, if the portal, "SelfService," is configured for the FAQ link, then the URL pattern is `/templates/selfservice/`.

Page Push

During a chat session, agents can send web pages to customers. With the page push feature, agents can quickly and accurately direct customers to specific webpages.

For pushed pages to appear in a separate `WebView`, a `WebViewClient` must be provided for the `WebView`. `WebViewClient` should implement `shouldOverrideUrlLoading(WebView view, String url)` to handle pushed pages that have been clicked.

Any URL that does not have the `/templates/chat/` pattern in the URL should be considered a page push URL.

For more information about page push and configuring it for chats, see *Enterprise Chat and Email Administrator's Guide to Routing and Workflows*.

Enabling Mobile Chat for Chat Customer SSO

Chat entry points can be configured to transfer customer context information from the company website to the application through SAML. To learn more about Secure Chat for customers, see *Enterprise Chat and Email Administrator's Guide to Administration Console*.

To configure mobile templates for customer single sign-on:

1. On the web server, navigate to *Cisco_home*`\web\templates\chat\`*Template_Name*`\` and `open application-chat-default.js`.
2. Enable auto-login by setting the value of the `EnableAutologin` parameter to true.
3. If any of the attributes will be transferred to the application in the SAML assertion, set the `secureAttribute` property value to **1**.
4. Implement the method `getCustomerData` in `AndroidInterface` to pass customer context parameters to the web view. This method returns a string representation of `JSONObject` with `key,value` pairs corresponding to login parameters configured in `application-chat-defaults.js` where:

   `Key = fieldname_1, fieldname_2 etc` OR `Key = providerAttributeName` configured in template login parameters

   `Value =` the value of customer attribute

   ```
   @JavascriptInterface
   public String getCustomerData(){
   JSONObject customerObject = new JSONObject();
   try {
   customerObject.put("fieldname_1","SDK Customer");
   customerObject.put("fieldname_2","sdk@egain.com");
   customerObject.put("fieldname_4","My question");
   } catch (JSONException e) {
   e.printStackTrace();
   }
   return customerObject.toString();
   }
   ```

5. Implement the method `getCustomerData` in `AndroidInterface` to pass SAML the token. The JSON object should have a key value pair where:

   `Key =` *SAMLResponse*

   `Value =` *Saml Token*

   `customerObject.put("SAMLResponse",`*"SAML_token"*`);`

6. To enable autologin and secure chat, the following snippet must be added to the URL: `postChatAttributes=true`

Print and Save Transcript

During a chat session and at the end of a chat session, customers have access to the **Print Transcript** and **Save Transcript** buttons. Customers can use these buttons to print or save the chat transcript for future reference. For more details on the functionality of these features, see *Enterprise Chat and Email Administrator's Guide to Chat and Collaboration Resources*.

To enable the save transcript feature on mobile templates:

- Implement the JavaScript Interface method: `saveTranscript`. The Aria template calls this method when the user selects the **Save Transcript** option.

```
@JavascriptInterface
    public void saveTranscript(String data) {}
data — chat transcript in HTML format.
```

To enable the print transcript feature on mobile templates:

- Implement the JavaScript Interface method: `printTranscript`. The Aria template will call this method when the user selects **Print Transcript** option.

```
@JavascriptInterface
    public void printTranscript(String data) {}
data — chat transcript in HTML format.
```

Resuming an Ongoing Chat on Back button click

If the Android Activity where chat view is loaded has a **Back** button, a user may click the **Back** button during an ongoing chat. In such cases, it would be desirable that user can resume with ongoing chat when he returns to the chat view.

For such scenarios, an Activity is created for the chat is loaded as a "`singleinstance`" activity. Upon clicking of the **Back** button, the activity is not closed. The activity is closed when chat ends.

Using this approach ensures that the same instance of the Chat webview activity is invoked whenever chat is launched.

Terminating Chats

The `MobileInterface` object can be referred from the Android app to call JavaScript methods from the application code.

MobileInterface Method	Description
CloseChat()	Method call for closing chat. This is raised by app for closing chat from within the app.
ShowCloseButton(boolean)	Method for showing/hiding close button inside template. By default, the button is visible.

A user may end the chat session by:

- Clicking the **Close** button on the template
- Attaching an event to a button listener in the app
- Closing the application via the application manager on the device

The app code can detect whether the **Close** button for the app has been clicked or if the `AndroidInterface.onChatClosed()` method has been called.

See the following table for a quick reference to the behavior of the app in the event of a chat termination and the actions that correspond to this behavior.

Behavior	Action
Trigger an event with the Close button of the template	`AndroidInterface.onChatClosed()` called by the template when the Close button is clicked.
Close the chat from within the app	Call the `MobileInterface.CloseChat()` method from the app.
Hide the web view upon closing the chat	Set the visibility of `WebView` to hidden from within the app.
Show or hide the Close button on the chat template toolbar	Call the `MobileInterface.ShowCloseButton(false)` method from the app..
Notify the app when the chat ends	`AndroidInterface.onChatEnded()` method called by the template when the chat session ends.

Aria Chat Template Integration with iOS

Important: **The code examples provided in this chapter are based on IOS 8+ using the WKWebView object.**

Using the Chat Entry Point URL

An entry point defines the starting point from which customers initiate chat interactions. Every chat help link on a website is mapped to an entry point. Each entry point has a queue associated with it and the queue is used to route chats to agents. For more details, see *Enterprise Chat and Email Administrator's Guide to Chat and Collaboration Resources*.

To use the entry point URL:

1. Create a web view and load the chat URL in the web view to embed chat in the IOS app.
2. If the chat URL protocol is using HTTP, permissions for the mobile app must be set:

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

3. Next, in the Administration Console, create the entry point and obtain the URL. For details, see *Enterprise Chat and Email Administrator's Guide to Chat and Collaboration Resources*. An example URL is such:

   `http(s)://`*server_name*`/system/templates/chat/`*template_name*`/index.html?entryPointId=entrypoint_id&templateName=`*template_name*`&ver=v11&locale=`*locale*

   Where the *server_name* is the web server, the *template_name* is the chat template deployed on the application server, the *entrypoint_id* matches the id of the desired entry point for incoming chat activities, and the *locale* is the language code and country code, such as *en_us*.
4. Copy the URL from the Administration Console and add it to your webpage.
5. If you wish to make the entry point proactive, the following parameters should be appended to the URL with values from an Offers Banner URL with the eGain Solve for Cisco Offers add-on:

```
aId=
sId=
uId=
```

Interface Between the iOS App and the Chat Template

In order for the iOS app and JavaScript to communicate, the app must have event listeners configured. `ViewController` must have `WKScriptMessageHandler` and listeners in the following table added to `WKUserContentController`.

iOS Interface Method	Description
IOSInterface	This will be available to the template to identify that chat is launched through iOS app.
IOSChatStarted	Javascript would call this handler to pass chat session ID to native app at the time chat starts.
IOSChatClosed	This would be the handler for chat button close event. JavaScript would post message to this handler.
IOSChatEnded	This handler is called when chat ends. Any cleanup on the native app can be done at this time.
IOSGetCustomerData	JavaScript would call this handler to get customer information for secure chats and auto login.
IOSSetSessionId	JavaScript called this handler to pass chat session ID to native app. This session ID is later required to download attachment.
IOSSaveTranscript	This handler is called when customer clicks on “Save Transcript” link. Formatted chat transcript is sent to this listener in the message body.
IOSPrintTranscript	This handler is called when customer clicks on “Print Transcript” link. Formatted chat transcript is sent to this listener in the message body.
IOSDownloadAttachment	This handler is called when user any time user clicks on attachment icon/thumbnail in the chat window.

For example, in Swift 4, use the following steps to add listeners in `ViewController`.

To add listeners in ViewController:

1. Create an Instance for `WKUserContentController` and add message handlers to the content controller.

   let `contentController = WKUserContentController()`

   ```
   contentController.add(
       self,
       name: "IOSInterface"
   )
   ```

   Likewise, add all the handlers listed in the table in `contentController`.

2. Add the `contentController` in `userContentController` property of webview configuration.

   let `config = WKWebViewConfiguration()`

```
config.userContentController = contentController
```

3. Handle all the function call from javascript in `func userContentController` which is automatically called by IOS at runtime with two parameters. The function is a s follows:

```
func userContentController(_ userContentController: WKUserContentController,
didReceive message: WKScriptMessage) {

 //message.name is the name of the Interface which we added in
contentController

// message.body has the parameters passed in javascript function call

}
```

JavaScript Interface

An object MobileInterface would be available in IOS app which enables calling JavaScript methods from application code.

Mobile Interface	Description
CloseChat()	Method call for closing chat. This will be raised by app for closing chat from within the app.
ShowCloseButton(boolean)	Method for showing/hiding close button inside template. By default button is visible.
OnCustomerDataReceived	JavaScript callback to be invoked by app to send customer information after the `IOSGetCustomerData` call.

Attachments

During a chat session, file attachments can be exchanged between customers and agents. Administrators can enable attachments for chat and specify the maximum allowed size for chat attachments. Additionally, administrators can specify the file types that can be attached to emails and chat messages. For more details, see *Enterprise Chat and Email Administrator's Guide to Administration Console*.

Uploading Attachments

Users are able to upload attachments only if the following permissions are set:

- `<key>NSPhotoLibraryUsageDescription</key>`
- `<string>App permission for attachment upload</string>`

Downloading Attachments

`IOSDownloadListener` is called whenever a user clicks an attachment after sending the attachment to agent or receiving attachment from agent. This listener then receives the URL to be called for fetching the attachment:

```
if(message.name == "IOSDownloadAttachment"){

let attachmentUrl = URL(string:message.body as! String)
```

```
load(url: attachmentUrl!)
}
```

The native app makes a request to this API and can write to a file. The API call is a POST request. In the request, set the `Content-Type` header as `application/x-www-form-urlencoded` and pass the `sid` parameter in the form-data equal to the chat session ID.

This chat session ID can be set by implementing the `IOSSetSessionId` listener, which is called when chat session starts. This listener receives chat session ID in message body:

```
if(message.name == "IOSSetSessionId"){
chatSessionId = message.body
}
```

Adjusting Dialog Windows

If using `WKWebView`, the native app handles JavaScript confirmation messages, alerts, and `window.open` events. An implementation of `WKUIDelegate` is required and it must be set to `chatView.uiDelegate`.

To display JavaScript confirmation messages within the native app:

- Implement the `runJavascriptConfirmPanelWithMessage` method.

To display JavaScript alert messages within the native app:

- Implement the `runJavascriptAlertPanelWithMessage` method.

To handle JavaScript window.open events:

- Implement the `createWebViewWithConfiguration` callback method on the native app. This implementation applies to links that are opened from the chat window, such as the **FAQ** link in the alternate engagement options.

Page Push

During a chat session, agents can send web pages to customers. With the page push feature, agents can quickly and accurately direct customers to specific webpages.

To view pushed pages in a separate `WebView`, an implementation of `WKNavigationDelegate` is required on the native app. The implementation must be set to `chatView.navigationDelegate`, where `chatView` is the web view where the chat URL is loaded. The delegate requires the `decidePolicyForNavigationAction` callback method be implemented to open the URL in new web view.

Any URL that does not have the `/templates/chat/` pattern in the URL should be considered a page push URL.

For more information about page push and configuring it for chats, see *Enterprise Chat and Email Administrator's Guide to Routing and Workflows*.

Enabling Mobile Chat for Chat Customer SSO

Chat entry points can be configured to transfer customer context information from the company website to the application through SAML. To learn more about Secure Chat for customers, see *Enterprise Chat and Email Administrator's Guide to Administration Console*.

To configure mobile templates for customer single sign-on:

1. On the web server, navigate to *Cisco_home*`\web\templates\chat\`*Template_Name*`\` and `open application-chat-default.js`.
2. Enable auto-login by setting the value of the `EnableAutologin` parameter to **true**.
3. If any of the attributes will be transferred to the application in the SAML assertion, set the `secureAttribute` property value to **1**.
4. Implement the method `userContentController` to listen to messages posted by JavaScript. When JavaScript uses a `postMessage` to get customer data, the message name is `IOSGetCustomerData`.

   Customer data needs to be in the form of key value pairs and should be passed to JavaScript as String representation of the key, value object:

   ```
   let customerObject: NSDictionary = ["SAMLResponse":"Test
   Customer","fieldname_2":"test@test.com","fieldname_4":"Test"]
   if JSONSerialization.isValidJSONObject(customerObject) {
   do {
       let data = try JSONSerialization.data(withJSONObject: customerObject)
       if let string = NSString(data: data, encoding:
       String.Encoding.utf8.rawValue) {
       return string as String
       }
       } catch {
       return ""
       }
   ```

   This String value needs to be passed by invoking JavaScript callback `MobileInterface.OnCustomerDataReceived`
5. Implement the method `userContentController` to listen to messages posted by JavaScript. When JavaScript does a `postMessage` to get customer data, message name is `IOSGetCustomerData`. The JSON object should have a key value pair where:

   `Key =` *SAMLResponse*

   `Value =` *SAML Token*
6. To enable `autologin` and secure chat, the following snippet must be added to the URL: `postChatAttributes=true`

Print and Save Transcript

During a chat session and at the end of a chat session, customers have access to the **Print Transcript** and **Save Transcript** buttons. Customers can use these buttons to print or save the chat transcript for future reference. For more details on the functionality of these features, see *Enterprise Chat and Email Administrator's Guide to Chat and Collaboration Resources*.

To enable the save transcript feature on mobile templates:

- `IOSSaveTranscript` is called when a user clicks the **Save Transcript** link. The native app requires a handler for `message.name= IOSSaveTranscript` inside the `userContentController` function. JavaScript sends the formatted chat transcript as a string to this handler and the native app can choose to display this in a separate web view or file.

```
if(message.name == "IOSSaveTranscript"){
let text = message.body
}
```

To enable the print transcript feature on mobile templates:

- `IOSPrintTranscript` is called when a user clicks the **Print Transcript** link. The native app requires a handler for `message.name= IOSPrintTranscript` inside the `userContentController` function. JavaScript sends the formatted chat transcript as a string to this handler.

```
if(message.name == "IOSPrintTranscript"){
let text = message.body
}
```

Terminating Chats

The `MobileInterface` object can be referred from the iOS app to call JavaScript methods from the application code.

MobileInterface Method	Description
CloseChat()	Method call for closing chat. This is raised by app for closing chat from within the app.
ShowCloseButton(boolean)	Method for showing/hiding close button inside template. By default, the button is visible.
OnCustomerDataReceived	JavaScript callback to be invoked by app to send customer information after the `iOSGetCustomerData` call.

A user may end the chat session by:

- Clicking the **Close** button on the template
- Attaching an event to a button listener in the app
- Closing the application via the application manager on the device

These different methods are there for the users convenience, but each method triggers a different event within the application, so it is important to consider which method the user may utilize and which objects in the template are called.

The app code can detect whether the **Close** button for the app has been clicked or if the `IOSChatClosed` event handler has been called.

See the following table for a quick reference to the behavior of the app in the event of a chat termination and the actions that correspond to this behavior.

Behavior	Action
Trigger an event with the Close button of the template	Call the `IOSInterface.onChatClosed()` method from the chat template by clicking the **Close** button.
Close the chat from within the app	Call the `MobileInterface.CloseChat()` method.
Hide the web view upon closing the chat	Set the visibility of `WebView` to hidden from within the app.
Show or hide the Close button on the chat template toolbar	Call the `MobileInterface.ShowCloseButton(false)` method.