Enterprise Chat and Email Chat and Callback Javascript SDK 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 Chat and Callback Javascript SDK Developer’s Guide: For Unified Contact Center Enterprise. May 5, 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 Chat and Callback Javascript SDK 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 products may be modified or key processes may be determined to be important. These 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.

Log into www.cisco.com and then access the tool at https://www.cisco.com/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 Library Basics

- Key Concepts
- Getting Started
- Library References

Key Concepts

The JavaScript Library is designed with a core set of objects that developers interact with. The library primarily has the following modules:

- Chat
- Callback

These modules serve as interfaces for Chat functionality and Callback functionality respectively.

When getting started with the modules of the JavaScript library, it is important to understand some basic key concepts associated with the user experience:

- Managing multiple languages should be considered while designing the user experience.
- CORS is required to be enabled on the ECE server when deploying the user experience outside of the application infrastructure. For more details see "Enabling CORS on ECE Server" on page 50.
- The following interfaces are typically used during a user's chat experience:
  - **Pre-Chat Form:** Used to collect information from the user that helps provide context when the chat is routed to an agent.
  - **Interaction UI:** This is the core UI where the customer types messages to send to the agent and receive messages sent by the agent.
  - **Post-Chat Form:** This is a page displayed after the chat has been completed. This is ideal for collecting feedback or presenting surveys.
- The following interfaces are typically used during a user's callback experience.
  - **Pre-Callback Form:** Used to collect information from the user that helps provide context when the callback is routed to an agent.
  - **Post-Callback Form:** This is a page displayed after the callback has been completed. This page displays the status messages such as "Callback successful," "Callback unsuccessful," etc.

Getting Started

To begin working with the JavaScript SDK, it is recommended to start with the working examples provided with this SDK. They are distributed as a quick start. The library itself is located in the `Examples/libs/egain` folder contained within the distribution.

Once extracted locate the web folder inside the location where the files were extracted to. This folder contains the source code for all of the samples provided.

Folder Name	Contains
libs	The libraries used throughout the examples, which includes: ▸ jQuery ▸ jQuery Mobile ▸ Chat JavaScript ▸ Callback JavaScript Both minified and development versions of the Chat JavaScript Library have been included.
samples	Various examples demonstrated in an HTML page with a corresponding JavaScript source file containing the code.
samples/simple-anonymous-chat	Examples with the easiest of scenarios where a simple anonymous chat is desired without any specific customer information.
samples/with-input-parameters	An example where chat is initiated with customer information passed at the time of starting chat.
samples/ chat-attachments	An example demonstrating sample code for sending and receiving attachments in chat.

Library References

Library Objects

For more information about library objects, see "Library Objects" on page 15.

- eGainLibrarySettings
- eGainLibrary
- Chat
- Callback
- CustomerObject
- CustomerParameter
- EventHandlers
- ResultObject
- CustomerParameter

Methods

Library Methods

These are the methods of the eGainLibrary objects:

- AddConnectionParameter
- SetCustomer
- SetSamlResponse
- SetEscalationData
- SetXEgainSession
- SetVisitorHistoryInformation
- GetQueueCurrentStatus

Chat Methods

These are the methods of the Chat objects:

- Initialize
- Start
- GetEventHandlers
- AcceptAuthRequest
- DeclineAuthRequest
- SendMessageToAgent
- SendSystemMessage
- SendCustomerStartTypingStatus
- SendCustomerStopTypingStatus
- End
- Attach
- UploadAttachment
- GetAttachment
- GetArticleAttachment
- GetAttachmentImage
- SendCustomerAttachmentNotification
- SendAcceptChatAttachmentNotification
- SendRejectChatAttachmentNotification
- GetTranscript
- SendMaskedMessageToAgent

- SendBeaconToEndChat

Callback Methods

These are the methods of Callback objects:

- Initialize
- Start
- GetEventHandlers

EventHandlers

Chat EventHandlers

- OnConnectionInitialized
- OnConnectSuccess
- OnConnectionComplete
- OnConnectionFailure
- OnAcceptAuthRequest
- OnDeclineAuthRequest
- OnConnectionAttached
- OnConnectionAttachedFailure
- OnDuplicateSession
- OnAgentsNotAvailable
- OnSystemMessageReceived
- OnGetQueueCurrentStatus
- OnMessagePropertyLoad
- OnErrorOccurred
- OnAgentMessageReceived
- OnAgentJoined
- OnChatTransfer
- OnAgentStartTyping
- OnAgentStopTyping
- OnTranscriptFetched
- OnCobrowseInviteReceived
- OnCustomerAttachmentNotificationSent

- OnGetAttachment
- OnAttachmentAcceptedByCustomer
- OnAttachmentUploadedByCustomer
- OnAttachmentRejectedByCustomer
- OnAttachmentAcceptedByAgent
- OnAttachmentRejectedByAgent
- OnAttachmentInviteReceived
- OnGetAttachmentImageThumbnail
- OnCustomerBlocked

Callback EventHandlers

- OnCallbackInitialized
- OnCallbackMessagePropertyLoad
- OnCallbackConnectionFailure
- OnDuplicateSession
- OnCallBackCompletion
- OnSystemMessageReceived
- OnAgentsNotAvailable
- OnCallbackConnectSuccess
- OnCallbackSucceeded

2 Library Objects

- eGainLibrarySettings
- eGainLibrary
- Chat (egainChatLibrary.Chat)
- Callback (egainChatLibrary.Callback)
- CustomerObject (egainChatLibrary.Datatype.CustomerObject)

eGainLibrarySettings

Initializing the `eGainLibrarySettings` object is the first step to use the chat library. The `eGainLibrarySettings` object is used to set specific configurations to be used by the JavaScript Library.

eGainLibrarySettings Properties

Property Name	Type	Description
IsDevelopmentModeOn	Boolean	This property sets an increased timeout for the chat to ensure empty sessions are less likely to occur during development. It is expected that, during development, breakpoints may hold up the execution, so enabling this mode assists with the development activities.
CORSHost	String	If chat UI built using the SDK is not from an ECE install, CORSHost is the identifier for the ECE server. It should have the ECE URL up till the context path.
ChatPauseInSec	Integer	Time in seconds to specify for how long chat needs to be paused. This would be used when pausing the connection. This time should be less than the MAX PAUSE TIME set in server.
eGainContextPath	String	Signifies context path where ECE templates are deployed. This is the path from which I10 properties files (namely messaging_en_US.properties, etc) required for the library will be referenced.
IsDebugOn	Boolean	Defines whether or not debugging is on for chat. This can be used to record Strophe requests sent to server. Strophe is the underlying Javascript XMPP library used by the chat functionality of ECE.

eGainLibrarySettings Methods

There are no methods to be called on the `eGainLibrarySettings` object.

Sample Code

```
var librarySettings = new eGainLibrarySettings();
librarySettings.CORSHost = Context root of my egain server;
librarySettings.IsDevelopmentModeOn = false;
librarySettings.eGainContextPath = "";
librarySettings.ChatPauseInSec = "30";
ibrarySettings.IsDebugOn = false;
```

eGainLibrary

The `eGainLibrary` object is the primary object used to interact with chat and callback. It holds the `Chat` and `Callback` objects. In addition, it contains methods and properties which are common to both chat and callback.

eGainLibrary Properties

The `eGainLibrary` itself does not contain any properties that need to be set.

eGainLibrary Methods

Core methods required for Chat and Callback are invoked from the respective objects. `eGainLibrary` has some additional methods primarily required for chat like setting customer details, setting visitor history information etc. For details on `eGainLibrary` methods, see "Library Methods" on page 21.

Sample Code:

```
var egainChatLibrary = new eGainLibrary(librarySettings());
```

Chat (egainChatLibrary.Chat)

The `Chat` object is the interface for chat functionality: sending and receiving text messages.

Chat Properties

The `Chat` object itself does not contain any properties that need to be set. It has an initialize method, which takes the parameters of the properties that need to be set for starting a chat.

Chat Methods

The `Chat` object has a host of methods related to:

- Initializing and starting chats
- Sending and receiving messages
- Connecting to an existing chat
- Exchanging files during chat

For more details about chat methods, see "Library Methods" on page 21.

Sample Code

```
var egainChat = new egainChatLibrary.Chat();
var chatEventHandlers = egainChat.GetEventHandlers();
```

```
egainChat.Initialize(Entry Point ID, Language, Country, chatEventHandlers);
```

Callback (egainChatLibrary.Callback)

The `Callback` object is the interface for callback functionality.

Callback Properties

The `Callback` object doesn't contain any properties that need to be set. It has an initialize method, which takes the parameters of the properties that need to be set for initiating a callback.

Callback Methods

The `Callback` object primarily has two related methods: initializing callback and starting a callback session.

For more details about callback methods, see "Library Methods" on page 21.

Sample Code

```
var egainCallback = new egainChatLibrary.Callback();

var eventHandlers = egainCallback.GetEventHandlers();

egainCallback.Initialize(<Entry Point ID>, <Language>, <Country>,
eventHandlers);
```

CustomerObject (egainChatLibrary.Datatype.CustomerObject)

`CustomerObject` is used to populate specific attributes of the customer who is initiating the chat. The following are a list of the methods and properties associated with the `CustomerObject`.

CustomerObject Properties

Property Name	Type	Description
Locale	Object	Locale Object containing the language code and country code.
PrimaryKey	Object	Object containing name,value pair. Name can be either "Email" or "Phone"
CustomerParameters	Array	Array of customer parameters. Details of customer parameters are under CustomerParameter.

CustomerObject Methods

Property Name	Description
AddCustomerParameter	Method used to add details regarding the customer. This can include things like the customer's first name, last name, or other details that are relevant. These values then get mapped to the business objects.
SetPrimaryKey	Used to set whether the email address or phone number will be used to identify the customer.

Sample Code

```
var customer = new egainChatLibrary.Datatype.CustomerObject();
customer.Locale.Language = 'en'
customer.Locale.Country = 'US'
```

ResultObject

Returns a status whether or not the event was successful.

ResultObject Properties

Property Name	Type	Description
StatusCode	String	String indicating status code of chat event. This code can be used for identifying the event.
StatusMessage	String	String indicating detailed status message of chat event.
IsSuccess	Boolean	String indicating status success or failure of chat event.

ResultObject Methods

There are no methods to be called on the `ResultObject` object.

CustomerParameter

The `CustomerParameter` object is used to create contextual parameters for the `CustomerObject`. That object is then passed into the `Chat` constructor.

CustomerParameter Properties

Property Name	Type	Description
eGainParentObject	String	Represents the Business Object within the ECE application that will be used to map the value to. An example of a ParentObject value would be “casemgmt”
eGainChildObject	String	Represents the Child Business Object within the ECE application that will be used to map the value to. An example of a ChildObject value would be “individual_customer_data”.
eGainParamName	String	Name of parameter which will be mapped to an ECE attribute.
eGainAttribute	String	The attribute name for the given Business Object within the ECE application that will be used to map the value to. An example of an Attribute value would be “first_name”.
eGainValue	String	The value that will be used to populate the Business Object within the ECE application. An example data value would be “John”.
eGainPrimaryKey	String	Whether chat parameter is primary key. Can have values “1” or “0”.
eGainMinLength	String	Minimum length of customer parameter.
eGainMaxLength	String	Maximum length of customer parameter.
eGainFieldType	String	Type of field in login form. Can be of values 1,2,3,4. 1-Text, 2-TextArea, 3-Dropdown, 4-Multiselect Dropdown
eGainRequired	String	If the parameter is required. Can have values “1” or “0”.
eGainValidationString	String	Validation pattern for the chat parameter. Can be left blank if not required.

CustomerParameter Methods

There are no methods to be called on the `ChatParameter` object.

3 Library Methods

Chat and Callback Constructors

Method Name	Description	Return Value
Chat	Used to create a new instance of the eGainLibrary Chat object that will be used to invoke methods for chat functionality.	New instance of eGainLibrary.Chat object
Callback	Used to create a new instance of the eGainLibrary Callback object that will be used to invoke methods for callback functionality.	New instance of eGainLibrary.Callback object

Chat Setup

Method Name	Description	Return Value
AddConnectionParameter	Used to add parameters to the chat connection URL.	None
SetCustomer	Used to set customer object for chat. Customer object is created by creating instance of CustomerObject and setting attributes on it.	None
SetSamlResponse	This method is used for secure chat to set the SAML token which will be used to establish chat connection.	None
SetXEgainSession	This method is used to set context of deflection data to chat.	None
SetVisitorHistoryInformation	Used to set visitor history information for chat.	None
GetQueueCurrentStatus	Gets details of about the current queue load and estimated wait time.	None. Passes information retrieved to callback handler.

Text Chat

Method Name	Description	Return Value
Initialize (EntryPointId, Language, Country, EventHandlers, TemplateName, Version)	Used to create initialize a chat connection. This does not start a chat but sets up all the parameters required to start a chat. **Input Parameters** ▸ **EntryPointId:** Chat Entry Point ID ▸ **Language:** This determines the locale (language+country) to start chat. ▸ **Country:** This determines the locale (language+country) to start chat. ▸ **ChatCallbacks:** This is the object which would have all the callback handlers for chat events. ▸ **TemplateName:** This is the chat template name. This is required by chat server to determine formatting of mailed transcripts. ▸ **Version:** This is the chat template version number. Set this to v11.	None
Start	Used to start the chat after the settings have been configured and the instance of the eGainLibary.Chat object has been created. **Input Parameters** None	None
End	Used to end a chat connection. **Input Parameters** None	None
GetEventHandlers	Used to get all the event handlers associated with chat. Event handlers are required to define functionality for each event. **Input Parameters** None	Object containing reference to all event handlers.
SendMessageToAgent (MessageToSend, IsMessageOffRecord)	This is the primary method used to send the customer messages to the agent. **Input Parameters** ▸ **MessageToSend:** HTML message sent by customer to agent. ▸ **IsMessageOffRecord:** The optional boolean parameter indicates whether or not the current message should be sent, but not stored in the transcript. Also, if masking is turned on, any messages sent with a true value for isMessageOffRecord will send the clear text to the agent NOT the masked version of the message.	String value representing the message that was sent to the agent. If message was masked, it will return masked message
SendMaskedMessageToAgent (maskedMessage)	This method is used to send the masked customer messages to the agent. **Input Parameters** ▸ **maskedMessage:** masked HTML message sent by customer to agent.	String value representing the message that was sent to the agent.

Method Name	Description	Return Value
SendSystemMessage (HtmlMessage, Command)	Used to send "System Messages" that the client needs to communicate to the server. A common use case where this method is used is to display an information message that sensitive data has been masked	None
	Input Parameters ▸ **HtmlMessage:** HTML message sent as a system message ▸ **Command:** Command is used to specify commands like offRecord. No need to pass for normal system messages For Example: egainChatLibrary.SendSystemMessage(data,'offrecord') is used to notify server of off-record message egainChatLibrary.SendSystemMessage(data,'onrecord') is used after an 'offrecord' message to notify server that forthcoming messages would again be on-record.	
SendCustomerStartTypingStatus	Used to send a notification to the chat agent that the customer is typing or has started typing. Can be called multiple times.	None
	Input Parameters None	
SendCustomerStopTypingStatus	Used to send a notification to the chat agent that the customer has stopped typing.	None
	Input Parameters None	
Attach (SessionID, RequestID)	API to attach to an existing chat. This needs to be called when user navigates from one page to another and on load of second page; this is needed to attach chat to existing chat on previous page.	None
	Input Parameters None	
AcceptAuthRequest (sessionID, requestId)	Used to get SAML request object and IDP URL after customer accepts mid chat authentication request sent by an agent	None
	Input Parameters ▸ **sessionID:** Chat session ID ▸ **requestId:** Unique identifier for the authentication request sent by an agent	
DeclineAuthRequest (sessionID, requestId)	Used to notify the server that the customer has declined the authentication request sent by an agent	None
	Input Parameters ▸ **sessionID:** Chat session ID ▸ **requestId:** Unique identifier for the authentication request sent by an agent	

Method Name	Description	Return Value
SendBeaconToEndChat	Used to send end chat request asynchronously on using the browser close button on some browsers where navigator.sendBeacon is supported.	None
	Input Parameters None	

Callback

Method Name	Description	Return Value
Initialize (EntryPointId, Language, Country, EventHandlers, TemplateName, Version, SubActivity)	Used to create initialize a callback connection. This does not start a callback but sets up all the parameters required to start a chat.	None
	Input Parameters ▸ **EntryPointId:** Chat Entry Point ID ▸ **Language:** This determines the locale (language+country) to start chat ▸ **Country:** This determines the locale (language+country) to start chat ▸ **ChatCallbacks:** This is the object which would have all the callback handlers for chat events. ▸ **TemplateName:** This is the chat template name. This is required by chat server to determine formatting of mailed transcripts ▸ **Version:** This is the chat template version number. Please set this to v11. ▸ **SubActivity:** This is the type of callback. Values can be “Callback” or “Delayedcallback”	
Start	Used to start the callback after the settings have been configured and the instance of the eGainLibary.Callback object has been created.	None
	Input Parameters None	
GetEventHandlers	Used to get all the event handlers associated with callback. Event handlers are required to define functionality for each event.	None
	Input Parameters None	

Chat Attachments

Method Name	Description	Return Value
UploadAttachment	API to upload a file to chat server.	None
	Input parameters ▸ **File:** Javascript File object which has information about file selected by customer for upload. ▸ **AgentName:** Name of chat agent.	
GetAttachment	API to download file sent by agent.	None
	Input parameters ▸ **FileId:** File ID of file. This is received when agent sends the attachment/file invite.	
GetArticleAttachment (FileId)	API to download an article attachment.	None
	Input Parameters ▸ **FileID:** ID of attachment to be downloaded. This ID is found in the chat message when agent sends an article with the attachment.	
GetAttachmentImage (attachmentId, uniqueFileId)	API to fetch attachment thumbnail.	None
	Input Parameters ▸ **AttachmentD:** ID of attachment to be downloaded. This ID is found in the chat message when agent sends an article with the attachment. ▸ **UniqueFileId:** This is a unique file name to be provided for the file for chat server to uniquely identify the file.	
SendCustomerAttachmentNotification (files, customerName)	Used to send notification to agent when attachment is sent by customer.	None
	Input Parameters ▸ **Files:** Array of File objects selected by customer to send to chat agent. ▸ **CustomerName:** Name of customer	
SendAcceptChatAttachmentNotification (fileId, fileName, customerName)	Used to send notification to agent when attachment sent by agent is accepted by customer.	None
	Input Parameters ▸ **FileId:** ID of file received in the attachment invite ▸ **FileName:** Name of file ▸ **CustomerName:** Name of customer	
SendRejectChatAttachmentNotification (fileId, fileName, customerName)	Used to send notification to agent when attachment sent by agent is rejected by customer.	None
	Input Parameters ▸ **FileId:** ID of file received in the attachment invite ▸ **FileName:** Name of file ▸ **CustomerName:** Name of customer	

4 Library Event Handlers

Text Chat

Callback Name	Description	Event Arguments Returned	Mandatory
OnConnectionInitialized	Event raised when chat connection is successfully initialized. This event handler would return all the chat settings-masking data, if attachments are enabled, etc.	Initialization data retuned by server	No
OnConnectSuccess	Event raised when customer chat client is connected successfully. Does NOT indicate the user is connected to an agent.	ChatConnectEventArgs	Yes
OnConnectionComplete	Event raised when chat connection is successfully ended.	None Returned	Yes
OnConnectionFailure	Event raised when the chat fails to connect or if a disruption to the chat occurs.	ChatConnectionFailureEventArgs	Yes
OnDuplicateSession	Event raised if chat session is attempted to start when there is already another chat session established.	DuplicateSessionEventArgs	No
OnAgentsNotAvailable	Event raised if there are no agents available for the entry point.	AgentsNotAvailableEventArgs	Yes
OnSystemMessageReceived	Event raised each time a system message is received.	SystemMessageReceivedEventArgs	No
OnGetQueueCurrentStatus	Event raised when queue status details are received from server.	GetQueueLiveStatusArgs	No
OnMessagePropertyLoad	Event raised after messaging property file is loaded.	ChatMessagePropertyLoadEventArgs	No
OnErrorOccurred	Event raised when any network error occurs during an ongoing chat.	ErrorOccurredEventArgs	Yes
OnAgentMessageReceived	Event raised when chat agent message is received by client.	AgentMessageReceivedEventArgs	No
OnAgentJoined	Event raised every time agent joins a chat session.	AgentJoinedEventArgs	No
OnChatTransfer	Event raised when chat is transferred to another agent/queue.	ChatTransferEventArgs	No
OnAgentStartTyping	Event raised when chat agent starts typing.	AgentStartTypingEventArgs	No
OnAgentStopTyping	Event raised when chat agent stops typing.	AgentStopTypingEventArgs	No
OnTranscriptFetched	Event raised when chat transcript is fetched.	GetTranscriptArgs	No
OnCobrowseInviteReceived	Event raised when agent invites customer to join cobrowse session.	CobrowseInviteReceivedEventArgs	No
OnConnectionAttached	Event raised when new chat is attached to an existing chat connection.	None Returned	No
OnConnectionAttachedFailure	Event raised if there is a failure in attaching to existing chat.	None Returned	No

Callback Name	Description	Event Arguments Returned	Mandatory
OnAcceptAuthRequest	Event raised when customer accepts mid chat authentication request sent by an agent	AcceptAuthReqArgs	No
OnDeclineAuthRequest	Event raised when customer declines mid chat authentication request sent by an agent	DeclineAuthReqArgs	No
OnCustomerBlocked	Event raised when the customer is blocked by an agent	None Returned	No

Callback

Callback Name	Description	Event Arguments Returned	Mandatory
OnCallbackInitialized	Event raised after Callback is initialized, i.e egainLibrary.Callback method is completed.	Initialization data retuned by server	No
OnCallbackMessagePropertyLoad	Event raised after messaging properties file is loaded.	CallbackMessagePropertyLoadEventArgs	No
OnCallbackMessagePropertyLoad	Event raised when the Callback fails to connect or if a disruption to the Callback occurs.	CallbackConnectionFailureEventArgs	Yes
OnDuplicateSession	Event raised when Callback session on same browser already exists.	DuplicateSessionEventArgs	No
OnCallBackCompletion	Event raised when Callback connection is successfully ended.	None Returned	Yes
OnSystemMessageReceived	Event raised each time a system message is received.	SystemMessageReceivedEventArgs	No
OnAgentsNotAvailable	Event raised if there are no agents available for the entry point.	AgentsNotAvailableEventArgs	Yes
OnCallbackConnectSuccess	Event raised when customer Callback client is connected successfully. Does NOT indicate the user is connected to an agent.	CallbackConnectSuccessEventArgs	Yes
OnCallbackSucceeded	Event raised when Call is successfully placed.	CallbackSuccessEventArgs	Yes

Chat Attachments

Callback Name	Description	Event Arguments Returned	Mandatory
OnGetAttachment	Event raised when attachment file is downloaded.	AgentAttachmentArgs	No
OnCustomerAttachmentNotificationSent	Event raised when attachment invite sent by customer is received by server.	CustomerAttachmentNotificationSentEventArgs	No

Callback Name	Description	Event Arguments Returned	Mandatory
OnAttachmentAcceptedByCustomer	Event raised when attachment invite sent by agent is accepted by customer.	AttachmentAcceptedByCustomerEventArgs	No
OnAttachmentUploadedByCustomer	Event raised when attachment upload by customer is complete.	AttachmentUploadedByCustomerEventArgs	No
OnAttachmentRejectedByCustomer	Event raised when attachment invite sent by agent is rejected by customer.	AttachmentRejectedByCustomerEventArgs	No
OnAttachmentAcceptedByAgent	Event raised when attachment invite sent by customer is accepted by agent.	AttachmentAcceptedByAgentEventArgs	No
OnAttachmentRejectedByAgent	Event raised when attachment invite sent by customer is rejected by agent.	AttachmentRejectedByAgentEventArgs	No
OnAttachmentInviteReceived	Event raised when attachment invite sent by agent is received on client.	AttachmentInvitedAgentEventArgs	No
OnGetAttachmentImageThumbnail	Event raised when attachment thumbnails are received on client.	AttachmentThumbnailArgs	No

5 Return Parameters From Event Handlers

Text Chat

Event Arguments	Property Name
ChatConnectEventArgs	▸ **ChatID:** Unique identifier of the activity that was created when the chat was initiated ▸ **SessionID:** Chat session ID ▸ **Name:** Customer name ▸ **Subject:** Subject entered when starting chat
ChatConnectionFailureEventArgs	▸ Used to create a new instance of the eGainLibrary Callback object that is used to invoke methods for callback functionality.
DuplicateSessionEventArgs	▸ **IsSuccess:** False ▸ **StatusCode:** DUPLICATE_SESSION ▸ **StatusMessage:** DUPLICATE SESSION
AgentsNotAvailableEventArgs	▸ **IsSuccess:** False ▸ **StatusCode:** AGENTS_UNAVAILABLE ▸ **StatusMessage:** AGENTS UNAVAILABLE
SystemMessageReceivedEventArgs	▸ **Message:** HTML version of the message sent by the system ▸ **AgentJoinedMessage:** True if it is the system message when an agent joins a chat, 'undefined' otherwise ▸ **ChatTransferMessage:** True if it is the system message when a chat is transferred, 'undefined' otherwise ▸ **ArticleAttachmentMessage:** True if it is the system message about an agent sending an article attachment, 'undefined' otherwise
GetQueueLiveStatusArgs	▸ **QueueDepth:** Position of chat in queue ▸ **WaitTime:** Wait Time for chat ▸ **AltEngmtTime:** Time after which alternate engagement options need to be shown
ChatMessagePropertyLoadEventArgs	▸ String containing I10n strings from the messaging property file
ErrorOccurredEventArgs	▸ **Status:** Status of error. Values can be – error(fatal error) and log(not a fatal error, should only be logged) ▸ **Message:** Details of error
AgentMessageReceivedEventArgs	▸ **Message:** HTML version of the message sent by the agent ▸ **AgentScreenName:** Screen name of the agent who is typing the message ▸ **PagePushMessage:** True if it is the message about a page push event, false otherwise
AgentJoinedEventArgs	▸ **AgentName:** Name of the agent
ChatTransferEventArgs	▸ **TransferType:** Method of activity transfer. Can be of values 1,2,3: 1-If chat is transferred to Department; 2-If chat is transferred to Queue; 3-If chat is transferred to Agent ▸ **TransferEntityName:** Name of the user/queue/department ▸ **ChatAttachmentEnabled:** True if attachments are enabled in the transferred queue, false otherwise
AgentStartTypingEventArgs	▸ **AgentScreenName:** Name of the agent
AgentStopTypingEventArgs	▸ **AgentScreenName:** Name of the agent

Event Arguments	Property Name
GetTranscriptArgs	▸ **CustomerName:** Name of the customer ▸ **Subject:** Subject entered when starting chat ▸ **Assignee:** Name of the agent if chat is assigned ▸ **StartTime:** Time when chat started in XSD format ▸ **Messages:** Array of chat messages
CobrowseInviteReceivedEventArgs	▸ **Action:** Cobrowse action ▸ **Session:** Cobrowse session ID ▸ **CustomerName:** Customer name
AcceptAuthReqArgs	▸ **?RequestId:** unique identifier for an authentication request ▸ **?SamlRequest:** SAML data ▸ **?IdpLoginUrl:** Identity provider login URL ▸ **?RelayState:** relaystate
DeclineAuthReqArgs	▸ **?RequestId:** unique identifier for an authentication request

Callback

Event Arguments	Property Name
DuplicateSessionEventArgs	▸ **IsSuccess:** False ▸ **StatusCode:** DUPLICATE_SESSION ▸ **StatusMessage:** DUPLICATE SESSION
AgentsNotAvailableEventArgs	▸ **IsSuccess:** False ▸ **StatusCode:** AGENTS_UNAVAILABLE ▸ **StatusMessage:** AGENTS UNAVAILABLE
SystemMessageReceivedEventArgs	▸ **Message:** HTML version of the message sent by the system ▸ **AgentJoinedMessage:** True if it is the system message when agent joins chat, 'undefined' otherwise ▸ **ChatTransferMessage:** True if it is the system message when chat is transferred, 'undefined' otherwise ▸ **ArticleAttachmentMessage:** True if it is the system message about agent sending an article attachment, 'undefined' otherwise
CallbackMessagePropertyLoadEventArgs	▸ String containing I10n strings from the messaging property file
CallbackConnectionFailureEventArgs	▸ **IsSuccess:** False ▸ **StatusCode:** Status code to help identify error condition ▸ **StatusMessage:** Details of error condition

Event Arguments	Property Name
CallbackConnectSuccessEventArgs	▸ **CallBackID:** Unique identifier of the activity that was created when the callback was initiated ▸ **SessionID:** Callback session ID ▸ **Name:** Name of the customer ▸ **Subject:** Subject entered when starting callback
CallbackSuccessEventArgs	▸ **IsSuccess:** True ▸ **StatusCode:** CALLBACK_SUCCESS ▸ **StatusMessage:** CALLBACK_SUCCESS

Chat Attachments

Event Arguments	Property Name
CustomerAttachmentNotificationSentEventArgs	▸ Status: 'success' if attachments are enabled ▸ File ▸ Message
AgentAttachmentArgs	▸ FileId ▸ UniqueFileId ▸ Data
AttachmentAcceptedByCustomerEventArgs	▸ FileId ▸ FileName ▸ CustomerName
AttachmentUploadedByCustomerEventArgs	▸ Status ▸ AttachmentId ▸ AttachmentName ▸ AttachmentInternalName
AttachmentRejectedByCustomerEventArgs	▸ FileId ▸ FileName
AttachmentAcceptedByAgentEventArgs	▸ AgentName ▸ UniqueFileId ▸ FileName
AttachmentInvitedAgentEventArgs	Attachment: ▸ ID ▸ Name ▸ AgentName ▸ Type ▸ AttachmentSize
AttachmentThumbnailArgs	▸ FileId ▸ UniqueFileId ▸ Data

6 Chat Code Snippets

The following are a series of code snippets demonstrating specific aspects of the Chat JavaScript Library.

Adding a Reference

In order to leverage the library, you must first add a reference to the JavaScript in your HTML page. The snippet below demonstrates a reference to the minified version of the JavaScript library for production use.

```
<!-- SAMPLE CHAT CLIENT -->
<!DOCTYPE html>
<html>
<head>
<title>Sample Chat Client</title>
<script src="egain-client-library-X.X.X.min.js"
type="text/javascript"></script>
</head>
</html>
```

Simple Startup Example

This example demonstrates the very basics of how to get a new instance of the library created and call the `StartChat()` method. The result of this code would be an anonymous chat with an agent for entry point ID 1000.

```
/* Create a new instance of the eGainLibrarySettings Object */
var librarySettings = new eGainLibrarySettings();
librarySettings.CORSHost ="http://myegainserver.com/system";
librarySettings.IsDevelopmentModeOn = false;
ibrarySettings.eGainContextPath = "";
librarySettings.ChatPauseInSec = "30";
librarySettings.IsDebugOn = false;

/* Next create a new instance of the eGainLibrary */
/* passing in the settings you have just created. */
var myLibrary = new eGainLibrary(librarySettings);

/* Now create an instance of the Chat Object */
var myChat = new myLibrary.Chat();
```

```
/* Next get the event handlers for chat. It is mandatory to provide definition
for the mandatory event handlers before initializing chat */
var myEventHandlers = myChat.GetEventHandlers();


/* Example browser alert when chat is connected */
        myEventHandlers.OnConnectSuccess = function () {
            alert('Chat Started!');
        };


  /* Example browser alert when there is a connection failure */
        myEventHandlers.OnConnectionFailure = function () {
            alert('Oops! Something went wrong');
        };


/* Example browser alert when there is an error during chat */
        myEventHandlers.OnErrorOccurred = function () {
            alert('Oops! Something went wrong');
        };


        /* Example output of agent messages to a DIV named TransScript with
jQuery */
        myEventHandlers.OnAgentMessageReceived = function
(agentMessageReceivedEventArgs) {
            $('#TransScript').append("<br />Agent: " +
agentMessageReceivedEventArgs.Message);
        };
        /* Example output of system messages to the same DIV */
        myEventHandlers.OnSystemMessageReceived = function
(systemMessageReceivedEventArgs) {
            $('#TransScript').append("<br />" +
systemMessageReceivedEventArgs.Message);
        };
        /* Example browser alert when agents are not available */
        myEventHandlers.OnAgentsNotAvailable = function
(agentsNotAvailableEventArgs) {
            alert('Sorry no agents available');
        };
        /* Example browser alert when the chat is completed */
        myEventHandlers.OnConnectionComplete = function () {
            $.mobile.changePage("#SimpleAnonymousChatPostChatScreen")
        };
```

```
/* Now call the Chat initialization method with your entry point and callbacks
*/
myChat.Initialize($('#ChatEntryPointId').val(),'en', 'US', myEventHandlers,
'aria', 'v11');
/* Start chat */
myChat.Start();
```

Adding Customer Parameters and Setting Primary Key

In this example specific context is added to the customer object before it is passed into the `StartChat()` method.

```
/* Create the customer object */
var myCustomer = new myLibrary.Datatype.CustomerObject();

/* Set the primary key as email and specify the email address */
myCustomer.SetPrimaryKey(myCustomer.PrimaryKeyParams.PRIMARY_KEY_EMAIL,"jdoe@no
mail.com");

/* Next we'll demonstrate adding the customer first name as a parameter */
var customerFirstName = new myLibrary.Datatype.CustomerParameter();
customerFirstName.eGainParamName = " full_name";
customerFirstName.eGainParentObject = " casemgmt";
customerFirstName.eGainChildObject = " individual_customer_data";
customerFirstName.eGainAttribute = "full_name";
customerLastName.eGainValue = $("#FirstName").val
customerFirstName.eGainMinLength = "1";
customerFirstName.eGainMaxLength = "50";
customerFirstName.eGainRequired = "1";
customerFirstName.eGainFieldType = '1';
customerFirstName.eGainValidationString ="";
myCustomer.AddCustomerParameter(customerFirstName);

/* Next we'll demonstrate adding the customer last name as a parameter */
var customerLastName = new myLibrary.Datatype.CustomerParameter ();
customerFirstName.eGainParamName = "last_name";
customerLastName.eGainParentObject = "casemgmt";
```

```
customerLastName.eGainChildObject = "individual_customer_data";
customerLastName.eGainAttribute = "last_name";
customerLastName.eGainValue = $("#LastName").val();
customerLastName.eGainMinLength = "1";
customerLastName.eGainMaxLength = "50";
customerLastName.eGainRequired = "1";
customerLastName.eGainFieldType = '1';
customerLastName.eGainValidationString ="";
myCustomer.AddCustomerParameter(customerLastName);
```

Starting the Chat Session

In this example, the settings for the library have already been specified, set the callbacks, and set the customer object.

```
/* Now call the Chat initialization method with your entry point and callbacks
*/
myChat.Initialize($('#ChatEntryPointId').val(),'en', 'US', myEventHandlers,
'aria', 'v11'); /* Then call the StartChat to create a chat */
myLibrary.SetCustomer(myCustomer);
/* Then call the StartChat to create a chat */
myChat.Start();
```

Sending Customer Messages to Agent

To send a message to the agent from the customer simply call the `SendMessageToAgent()` method.

```
/* Simply place a call with the message you want to send */
myChat.SendMessageToAgent ("Hello agent");
```

Handling Messages Received from an Agent

To handle the messages sent by a contact center agent, simply output the `OnAgentMessageRecivedEventArgs.Message` property.

```
/* Example output of agent messages to a DIV named TransScript with jQuery */
myCallbacks.OnAgentMessageReceived = function (agentMessageReceivedEventArgs) {
$('#TransScript').append("<br />Agent: " +
agentMessageReceivedEventArgs.Message);
};
```

Handling System Messages

System messages are items sent by the chat application. These include messages like “Agent has joined”, “Agent has ended the session” or other system related items. To process these messages simply output the `OnSystemMessageReceivedEventArgs.Message` property.

```
/* Example output of system messages to the Transcript DIV */
myCallbacks.OnSystemMessageReceived = function(systemMessageReceivedEventArgs)
{
$('#TransScript').append(systemMessageReceivedEventArgs.Message);
};
```

Chat Completion

It is common to transition to a post-chat UI when the chat is completed. To accomplish this, place the page navigation along with any additional calls inside the `OnChatCompletion` callback.

```
/* Example navigation when the chat is completed */
myCallbacks.OnConnectionComplete = function () {
window.location = "http://yourdomain.com/post-chat.html";
};
```

Masking Sensitive Information & Off-Record

The chat application supports the configuration of masking sensitive information during a chat session. If this option is configured in the console, you can leverage the capability with the following example. Note that by setting the On/Off record flag to “true” or “false”, you can determine whether or not the information is masked for a specific message. To display the result in the transcript section, simply append the result of this call to the transcript object.

```
/* Send sensitive information to agent On Record */
var sentSensitiveInfo = myChat.SendMessageToAgent("My SSN is 333-22-
4444",false);
/* Send sensitive information to agent Off Record */
var sentSensitiveInfo = myChat.SendMessageToAgent("My SSN is 333-22-
4444",true);
/* Now you can append the result of this call to the transcript */
$('#TransScript').append(sentSensitiveInfo);
```

Accepting Mid-Chat Authentication Request Sent by an Agent

When agent requests for mid chat authentication,server will publish a message with accept and decline options for customer.

When customer chooses to accept the mid chat authentication request, simply call **AcceptAuthRequest** method.

```
/* Call the method with chat session id and unique identifier for
authentication request received from the server */

myChat.AcceptAuthRequest(sessionID, requestId);
```

Declining Mid-Chat Authentication Request Sent by an Agent

When agent requests for mid chat authentication, server will publish a message with accept and decline options for customer.

When customer chooses to decline or not to continue with the mid chat authentication request, simply call **DeclineAuthRequest** method.

```
/* Call the method with chat session id and unique identifier for
authentication request received from the server */

myChat.DeclineAuthRequest(sessionID, requestId);
```

7 Callback Code Snippets

The following are a series of code snippets demonstrating specific aspects of the Callback JavaScript Library.

Adding a Reference

In order to leverage the library, you must first add a reference to the JavaScript in your HTML page. The snippet below demonstrates a reference to the minified version of the JavaScript library for production use.

```
<!-- SAMPLE CHAT CLIENT -->
<!DOCTYPE html>
<html>
<head>
<title>Sample Callback Client</title>
<script src="egain-client-library-X.X.X.min.js"
type="text/javascript"></script>
</head>
</html>
```

Simple Startup Example

This example demonstrates the very basics of how to get a new instance of the library created and call the `Start()` method. The result of this code would be an anonymous chat with an agent for entry point id 1000.

```
/* Create a new instance of the eGainLibrarySettings Object */
var librarySettings = new eGainLibrarySettings();
librarySettings.CORSHost ="http://myegainserver.com/system";
librarySettings.IsDevelopmentModeOn = false;
ibrarySettings.eGainContextPath = "";
librarySettings.ChatPauseInSec = "30";
librarySettings.IsDebugOn = false;

/* Next create a new instance of the eGainLibrary */
/* passing in the settings you have just created. */
var myLibrary = new eGainLibrary(librarySettings);

/* Now create an instance of the Callback Object */
var myCallback = new myLibrary.Callback();

/* get an instance of event handlers object */
var myCallbacks = myCallback.GetEventHandlers();
```

```
/* and provide the function calls you want to happen on each event type */

/* Example browser alert when Callback is connected */
myCallbacks.OnCallBackConnectSuccess = function (args) {
console.log('OnCallBackConnectSuccess..');
$('#TransScript').append("<br/> CallBack Initiated!");

};

/* Example browser alert when Call is placed */
myCallbacks.OnCallBackSucceeded = function () {
console.log('OnCallBackSucceeded');
$('#TransScript').append("<br/> Call Placed!");

};

/* Example browser alert when there is a connection failure */
myCallbacks.OnCallBackConnectionFailure = function (args) {
console.log('OnCallBackConnectionFailure');
$('#TransScript').append("<br/> Oops! Something went
wrong..Status="+args.StatusCode);
};

/* Example output of system messages to the same DIV */
myCallbacks.OnSystemMessageReceived = function (systemMessageReceivedEventArgs)
{
console.log('OnSystemMessageReceived');
$('#TransScript').append("<br/>" + systemMessageReceivedEventArgs.Message);
};

/* Example browser alert when agents are not available */
myCallbacks.OnAgentsNotAvailable = function (agentsNotAvailableEventArgs) {
console.log('OnAgentsNotAvailable')
$('#TransScript').append("<br/> Sorry no agents available");;
};

/* Example browser alert when the chat is completed */
myCallbacks.OnCallBackCompletion = function () {
```

```
console.log('OnCallBackCompletion');
$.mobile.changePage("#WithParametersPostCallbackScreen")
};
```

Adding Customer Parameters and Setting Primary Key

In this example specific context is added to the customer object before it is passed into the `StartChat()` method.

```
/* Create the customer object */
var myCustomer = new myLibrary.Datatype.CustomerObject();

/* Set the primary key as email and specify the email address */
myCustomer.SetPrimaryKey(myCustomer.PrimaryKeyParams.PRIMARY_KEY_EMAIL,"jdoe@no
mail.com");

/* Next we'll demonstrate adding the customer full name as a parameter */
var myCustomer = new myLibrary.Datatype.CustomerObject();

var customerFirstName = new myLibrary.Datatype.CustomerParameter();
customerFirstName.eGainParentObject = "casemgmt";
customerFirstName.eGainChildObject = "individual_customer_data";
customerFirstName.eGainAttribute = "full_name";
customerFirstName.eGainValue = "Joe Brown";
customerFirstName.eGainParamName = "full_name";
customerFirstName.eGainMinLength = "1";
customerFirstName.eGainMaxLength = "120";
customerFirstName.eGainRequired = "1";
customerFirstName.eGainFieldType = "1";
customerFirstName.eGainPrimaryKey = "0";
customerFirstName.eGainValidationString = "";
myCustomer.AddCustomerParameter(customerFirstName);

/* Next we'll demonstrate adding the customer email address as a parameter */

var customerEmail = new myLibrary.Datatype.CustomerParameter();
customerEmail.eGainParentObject = "casemgmt";
customerEmail.eGainChildObject = "email_address_contact_point_data";
```

```
customerEmail.eGainAttribute = "email_address";
customerEmail.eGainValue = "jdoe@nomail.com";
customerEmail.eGainParamName = "email_address";
customerEmail.eGainMinLength = "1";
customerEmail.eGainMaxLength = "50";
customerEmail.eGainRequired = "1";
customerEmail.eGainFieldType = "1";
customerEmail.eGainPrimaryKey = "1";
customerEmail.eGainValidationString = "";
myCustomer.AddCustomerParameter(customerEmail);

/* Next we'll demonstrate adding the customer phone number as a parameter */

var customerPhone = new myLibrary.Datatype.CustomerParameter();
customerPhone.eGainParentObject = "casemgmt";
customerPhone.eGainChildObject = "phone_number_data";
customerPhone.eGainAttribute = "phone_number";
customerPhone.eGainValue = "1112223333";
customerPhone.eGainParamName = "phone_number";
customerPhone.eGainMinLength = "1";
customerPhone.eGainMaxLength = "18";
customerPhone.eGainRequired = "1";
customerPhone.eGainFieldType = "1";
customerPhone.eGainPrimaryKey = "1";
customerPhone.eGainValidationString = "";
myCustomer.AddCustomerParameter(customerPhone);

/* Next we'll demonstrate adding the Delay Time (in minutes) as a parameter */

var delayTimeInMin = new myLibrary.Datatype.CustomerParameter();
delayTimeInMin.eGainParentObject = "casemgmt";
delayTimeInMin.eGainChildObject = "activity_data";
delayTimeInMin.eGainAttribute = "delay_time_in_min";
delayTimeInMin.eGainValue = "15"; // Note, this value will be 0 for CallBack.
It will be 0 or higher for Delayed Callback
delayTimeInMin.eGainParamName = "delay_time_in_min";
delayTimeInMin.eGainMinLength = "1";
delayTimeInMin.eGainMaxLength = "120";
```

```
delayTimeInMin.eGainRequired = "0";
delayTimeInMin.eGainFieldType = "1";
delayTimeInMin.eGainPrimaryKey = "0";
delayTimeInMin.eGainValidationString = "";
myCustomer.AddCustomerParameter(delayTimeInMin);

/* Next we'll demonstrate adding the Subject as a parameter */

var questionPrompt = new myLibrary.Datatype.CustomerParameter();
questionPrompt.eGainParentObject = "casemgmt";
questionPrompt.eGainChildObject = "activity_data";
questionPrompt.eGainAttribute = "subject";
questionPrompt.eGainValue = "0";
questionPrompt.eGainParamName = "subject";
questionPrompt.eGainMinLength = "1";
questionPrompt.eGainMaxLength = "120";
questionPrompt.eGainRequired = "0";
questionPrompt.eGainFieldType = "2";
questionPrompt.eGainPrimaryKey = "0";
questionPrompt.eGainValidationString = "";
myCustomer.AddCustomerParameter(questionPrompt);
```

Starting the Callback Session

In this example, the settings for the library have already been specified, set the callbacks, and set the customer object.

```
/* Now call the Callback initiliaztion method with your entry point and
callbacks. Also specify the subActivity as 'Callback' or 'DelayedCallback' */
myCallback.Initialize('1000','en', 'US', myCallbacks, 'rainbow', 'v11',
'Callback');

/* Start the callback */
myCallback.Start();
```

Handling System Messages

System messages are items sent by the callback application. They will include things like the “Agent has joined”, “Agent has ended the session” or other system related items. To process these messages simply output the `OnSystemMessageReceivedEventArgs.Message` property.

```
/* Example output of system messages to the Transcript DIV */
myCallbacks.OnSystemMessageReceived = function(systemMessageReceivedEventArgs)
{
$('#TransScript').append(systemMessageReceivedEventArgs.Message);
};
```

Appendix: Reference Information

Enabling CORS on ECE Server

If the chat application is deployed on the ECE server, then `egainLibrary. CORSHost` should be set to the server context root. However, if the chat application is deployed outside the application, the API requests made from client to server require CORS to be enabled on the server. `CORSHost` in this case should be set to the server context root with FQDN, for example:

```
egainLibrary. CORSHost = http://myserver.com/system
```

For details on how to enable CORS in the application, see *Enterprise Chat and Email Administrator's Guide to Administration Console*.