API Developer Guide

Chat Solution 3.0

Dec 01, 2018

Corporate Head Office
Expertflow LLC, Jägerweg 18, 3014 Bern,
Switzerland
www.ExpertFlow.com

1 Table of Contents

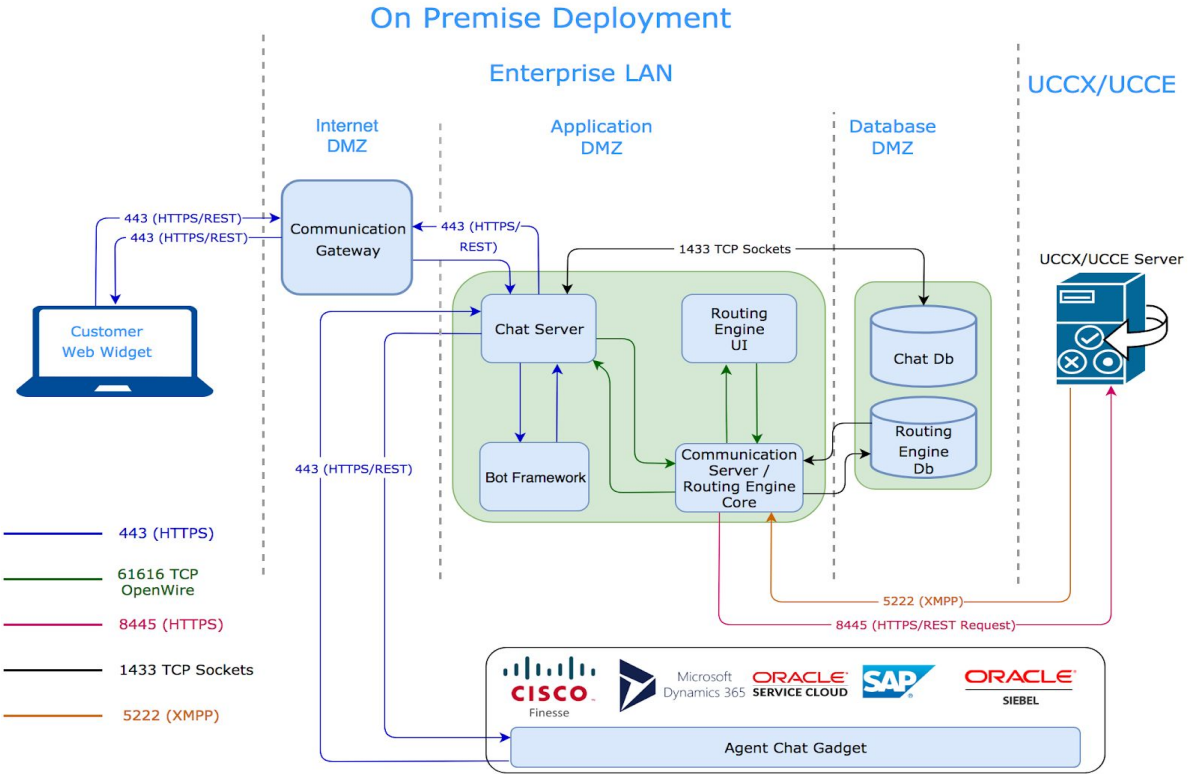
2 Architecture

This is a developer guide explaining Chat solution integration interfaces, their description, examples, and communication flows.

2.1 Abbreviations

Short Name	Full Name
MRE	Media Routing Engine
Gadget	Agent Gadget
Finesse	Cisco Finesse
Server	EF Chat Solution

2.2 Chat Solution Architecture in Enterprise Deployment



Solution Components	
Routing Engine	[Application Layer] Key component to perform precision queuing & agent selection per MRD.

	The Media Routing Engine supports agent-state per MRD and task-state per MRD operations.
Routing Engine UI	[Presentation Layer] Routing Engine UI is used to configure precision queues, routing algorithm, agent selection algorithm and agent attributes
Routing Engine Database	[Database Layer] Routing Engine configuration, queues configuration, attributes configuration and agents configuration is saved in Db.
Communication Server	[Application Layer] It's a middleware for all clients including Cisco Finesse Task Gadget, Siebel CRM, Microsoft Dynamics, SAP, and may be used with any desktop or server based CRM solution. It allows integration of chat bots with chat tasks as well. Communication Server acts as middleware for multiple customer side clients i.e. Skype, Facebook, Web Clients, Slack, Twitter etc.
Chat Server	[Application Layer] Chat server enables chat connectivity between a customer and an agent.

2.4 Chat Solution APIs

There are following kind of API's available for integration.

API Type	Description
Customer API	For Customer channel integration such as web, FB, SMS, and others. API based interface for developing customer chat interfaces in a mobile or a web app. provides support for customer side chat functions such as: - initiating a chat request, - ending chat session and - request for chat transcript.
Third Party API	For 3rd party applications to send messages to customers on different channels like SMS, FB, WeChat etc.
Agent API	Can embed the agent gadget in a 3rd party CRM application or implement the Agent gadget API to develop your own Agent gadget interface in your application for a web or a mobile app supports the agent side functionality such as: - signing in to the contact center,

	- change state, - accept chats, - wrap up etc.
Chat Bot API	To integrate other chatbots
Task Routing API	provides a way to integrate task routing services which can be used to select an appropriate agent for the chat.

2.5 Samples

Sample applications are provided with solution, you may use them to get started with development.

2.6 Communication Interface Types

All the API methods communicate in either of the following two communication styles.

Synchronous (REST)	Synchronous interfaces are based on REST APIs.
Asynchronous (web sockets)	Bidirectional communication can be carried out via HTTPS xhr polling or via secure websockets or a mix of both (Chat server prefers HTTPS xhr over websockets. However, Chat server may be configured to use any of them)

2.6.1 Developing the API Client Applications

The framework communicates over REST and Socket.io. Choose any of the following language frameworks for the custom application to connect with the Chat solution for:

- A new customer channel integration either from the web, mobile, or any social media platform
- Bot connector for IBM Watson, DialogFlow, or any other bot
- Agent gadget for a CRM application, a desktop, or an embedded HTML5 application

Here is a list of available socket.io clients for the popular languages.

- JavaScript (client)
- NodeJS
- Java/Android
- .NET
- Swift (iOS)
- C++

For further details on how to create a socket.io application, explore socket.io documentation.

2.7 Establish Chat Server Connection

To establish a client application connection to the Chat socket server, connect your socket.io client application with the chat server host url ***http://<FQDN>/***

2.7.1 Data Types

The following table lists the data types used in API parameters and notification message fields.

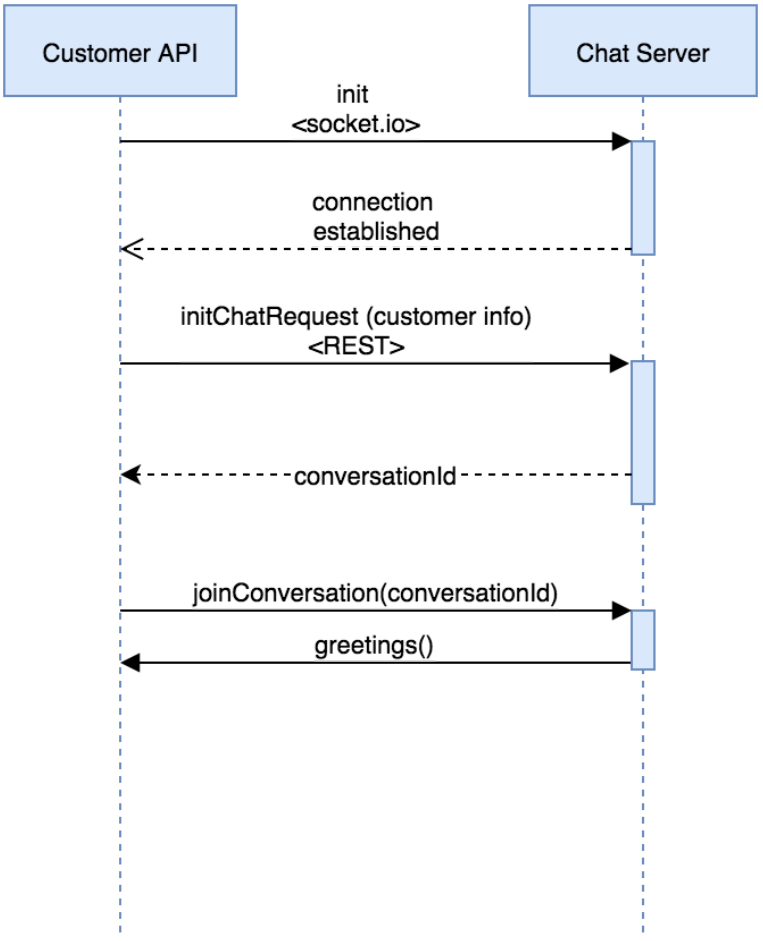
Type	Description
Boolean	A logical data type that has one of two values: true or false.
Integer	A 32-bit wide integer.
Long	A 64-bit wide integer.
String	A variable-length string. If a maximum length exists, it is listed with the parameter description.

3 Customer API

You can use Customer API to develop customer channel applications such as web widget, an SMS client, a Facebook, telegram, whatsapp or similar connector for the Chat solution. The integration app may be embedded in a mobile or a web app.

To develop a customer channel client application, the application would:

1. Establish a connection with the Chat server
2. Upon successful connection, Initiate a chat request
3. Take the sessionId from the response received in step.2 and emit joinConveration in order to be the participant of the conversation for bidirectional communication.
4. Implement Send and Receive messages for bidirectional message exchange.
5. Send campaign messages to customers received from chat server



3.0.1 Connecting Chat server

Establish a socket.io connection with the chat server using the URL http/s://<FQDN-CHAT-SERVER>:PORT/, where

FQDN-CHAT-SERVER	is the Chat server domain name or IP address accessed directly or via reverse proxy.
PORT	is the port number the Chat socket server is listening on. By default, the Chat server listens on 8080 (HTTP) or 443 (HTTPS).

3.0.2 Register Connector

You need to register you connector with Chat Server, in order to receive third party messages which will be forwarded to customers directly via your connector like in case of SMS, any third party will request chat server to send campaign sms to customer, the chat server will deliver this message along with customer contact number to the connector and the connector will send this message directly to the customer

When the client is connected, (socket received **`connect`** event), emit this event.

Protocol	Websockets
Event Name	registerConnector
Source	Client
Input/Output Format	JSON
Request Payload	`{ "id": "sms", "name": "ExpertFlow SMS Connector", "channel": "sms" }`
Payload Parameters	**id:** Connector id used to identify in Chat Server, must be same as channel name. **name:** Friendly name of connector to display **channel:** Name of channel
Event Triggered	

3.0.3 Initiate a chat request

The Init REST API is used to initiate a chat request to server. The client application sends chat request (containing requesting customer information).

The server responds with a chat `conversationId`. This `conversationId` is used to join a room using joinConversation in order to send and receive chat messages.

Protocol	HTTP
URL	*http://<FQDN>/api/customer/init*
Content Type	Application/JSON

HTTP Method	POST
Input/Output Format	JSON
Request Body	Following are the allowed parameters in the Request body. name - full customer name Email - customer email address Phone - customer phone number Channel - any of web, fb, sms, twitter reqestId - (optional) GUID representing a unique chat request timestamp - in the ISO 8601 format

Sample Request Body	{
	"name": "Kashif Sohail",
	"email": "kashif@ef.com",
	"phone": "+1343122554",
	"channel": "web",
	"requestId": "99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"
	}

Request Parameters	Request parameters can be defined by developers, with at least one identifier parameter. Channel parameter is required.
HTTP Response	200: Success 500: Internal Server Error 503: Service Unavailable
Response Body	Returns following parameters: conversationId - the ID that the client application should use to join the conversation Participant - customer identification as a participant for the conversation requestId - the same requestId the client submitted in the request Timestamp - the time in ISO 8601 format

Sample Response Body (in case of success)	{
	"conversationId": "5ae01d346c25be4319a9df95",
	"participant": {
	"id":"d346c25",
	"name":"Kashif Sohail"
	},
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"
	}

Sample Response Body (in case of failure)	

3.0.4 Send and Receive messages

Send messages	Sending message is quite similar to receive messages. To send a message, client needs to emit sendMessage event with payload ChatMessage and ActivityMessage as per need.
Receive messages	Client needs to implement messageArrived event for the conversation. Payload for this event is a message. Message is the parent class of ChatMessage and ActivityMessage. The payload contains property **type** in the JSON root, which can be either ChatMessage or ActivityMessage.

3.0.5 Request Transcript

This request is used to get chat transcript of a particular chat session. Transcript can be requested only after the chat session is ended. In case of ongoing chat session, the response will be Error 404.

Protocol	HTTP
URL	*http://<FQDN>/api/customer/transcript/ID/Type*
Content Type	Application/JSON
HTTP Method	GET
Input/Output Format	JSON
HTTP Request	-
Request Parameters	ID: The ID of session Type: Type of Transcript (email or pdf)
HTTP Response	200: Success 404: Not found 500: Internal Server Error 503: Service Unavailable
Example Response	Response will be different based on different Type. For Type “pdf”, the response will be pdf file For Type “email”, response will be empty

3.1 Events

Here is the list of all possible events which can be used as command sent to server and events received from server. Origin of each event is specified by Source.

Event Name	Description

joinConversation	The joinConversation event is used to join the conversation to send and receive messages.
messageArrived	The messageArrived event is used to receive messages from other participant. Client need to subscribe to this event.
sendMessage	The event use to send message to a conversation. Client need to emit this event.
endConversation	The endConversation event is used to end an active conversation.

3.1.1 joinConversation

The joinConversation event is used to join the conversation to send and receive messages.

Protocol	Websockets
Event Name	joinConversation
Source	Client
Input/Output Format	JSON

Request Payload	{
	"conversationId": "5ae01d346c25be4319a9df95",
	"participant": {
	"id":"d346c25",
	"name":"Alice"
	},
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"
	}

Payload Parameters	**conversationId:** Id of the conversation to join **id:** Participant id, supplied in response of initiate chat request **name:** Name of participant to display in conversation
Event Triggered	messageArrived type: ActivityMessage activityType: greetings

3.1.2 messageArrived

The messageArrived event is used to receive messages from other participant. Client need to subscribe to this event.

Protocol	Websockets
Event Name	messageArrived
Source	Server

Input/Output Format	JSON

Event Payload (for ChatMessage)	{
	"conversationId": "7g5639ju987g",
	"type": "ChatMessage",
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z",
	"from": {
	"id": "abcdedf",
	"name": "Kashif Sohail"
	},
	"to": [
	{
	"id": "yferdt5",
	"name":"Alice"
	},
	{
	"id": "4ty678gu",
	"name":"Bob"
	}
	],
	"refId":"00923011234567",
	"text": "Hello, what is the status of my request",
	"attachments": []
	}

Payload Parameters	conversationId: The id of conversation to which message is being sent type: Type of message, see ChatMessage and ActivityMessae for details.
Event Triggered	

3.1.3 sendMessage

The sendMessage event is used to publish messages to the conversation. Client need to emit this event. It's format is similar to messageArrived.

Protocol	Websockets
Event Name	sendMessage
Source	Client
Input/Output Format	JSON

Event Payload (for ChatMessage)	{
	"conversationId": "7g5639ju987g",
	"type": "ChatMessage",
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z",
	"from": {
	"id": "abcdedf",
	"name": "Kashif Sohail"
	},
	"text": "Hello, what is the status of my request",
	"attachments": []
	}

Payload Parameters	**conversationId:** The id of conversation to which message is being sent **type:** Type of message, see ChatMessage and ActivityMessae for details **from:** The sender participant **text:** text content of message.
Event Triggered	

3.1.4 endConversation

The endConversation event is used to end an active conversation.

Protocol	Websockets
Event Name	endConversation
Source	Client
Input/Output Format	JSON

Event Payload	{
	"conversationId": "7g5639ju987g",
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"
	}

Payload Parameters	**conversationId:** Id of the conversation to end
Event Triggered	

3.1.5 requestAgentTransfer

The requestAgentTransfer event is used to transfer chat from bot to human (in case bot is enabled).

Protocol	Websockets
Event Name	requestAgentTransfer
Source	Client
Input/Output Format	JSON

Request Payload	{
	"conversationId": "7g5639ju987g",
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"
	}

Payload Parameters	**conversationId:** Id of the conversation
Event Triggered	

3.2 Schema

Schema defines the object and its properties that your client application can use to communicate with server.

Object	Description
ChatMessage Object	Defines a text message that is exchanged between server and client application.
ActivityMessage Object	Defines a activity message that is exchanged between server and client application.
Participant object	Defines a bot or user account in the conversation.
Attachment Object	Array of Attachment objects that defines additional information to include in the message. Each attachment may be a media file (e.g., audio, video, image, file).
ActivityTypes	Types of activity messages

3.2.1 ChatMessage object

Defines a message that is exchanged between server and client.

Property	Type	Description
conversationId	String	The ID that identifies the conversation. The ID is unique per conversation. This conversationId is generated from service side and sent to client when the client request initiate chat
type	String	Type of message. ChatMessage in this case.
from	Participant	A **Participant** object that specifies the sender of the message
to	Participant[]	Array of **Participant** objects that specifies the recipients of the message. If the list is empty, the message will be broadcasted in the conversation.
text	String	Text of the message that is sent from user to bot or bot to user
refId	String	Account Id of the receiving account. (Phone no, in case of SMS)
attachments	Attachment[]	Array of **Attachment** objects that defines additional information to include in the message. Each attachment may be a media file (e.g., audio, video,

		image, file)

3.2.2 ActivityMessage object

Defines a activity message that is exchanged between server and client.

Property	Type	Description
conversationId	String	The ID that identifies the conversation. The ID is unique per conversation. This conversationId is generated from service side and sent to client when the client request initiate chat
type	String	Type of Message. ActivityMessage in this case
from	Participant	A **Participant** object that specifies the sender of the message
to	Participant[]	Array of **Participant** objects that specifies the recipients of the message. If the list is empty, the message will be broadcasted in the conversation.
activityType	String	Type of activity. One of these values: **greetings**, **typing**, **participantAdded**, **participantRemoved**, **participantJoined**, **participantLeft**, **endOfConversation** For details about activity types, see AcitivityTypes
info	Object	Notification data.

3.2.3 Participant object

Defines a bot or user account in the conversation.

Property	Type	Description
id	String	ID that uniquely identifies the bot or user in the conversation.
name	string	Name of the bot or user.

3.2.4 Attachment object

Array of Attachment objects that defines additional information to include in the message. Each attachment may be a media file (e.g., audio, video, image, file).

Property	Type	Description
type	String	The media type of the content in the attachment. For media files, set this property to known media types such as image/png, audio/wav, and video/mp4
name	String	Name of the attachment.

contentUrl	String	URL for the content of the attachment. For example, if the attachment is an image, set contentUrl to the URL that represents the location of the image. Supported protocols are: HTTP, HTTPS, File, and Data.
thumbnailUrl	String	URL to a thumbnail image that the channel can use if it supports using an alternative. For example, if you set type to application/word and set contentUrl to the location of the Word document, you might include a thumbnail image that represents the document. The widget could display the thumbnail image instead of the document. When the user clicks the image, the channel would open the document.

3.2.5 ActivityTypes

The most common type of activity is **greetings**, when a conversation is initiated, the client receives greetings from server.

The following activity types are supported by the server.

ActivityType	Description
greetings	The greeting message, when a conversation is initiated, the server sends greeting message to client. This message is configurable and can be changed in the admin panel.
typing	Indicates that the user or bot on the other end of the conversation is compiling a response.
participantAdded	Indicates a participant added to the conversation
participantRemoved	Indicates a participant has removed from the conversation
participantJoined	Indicates a participant has joined the conversation
participantLeft	Indicates a participant has left the conversation
endOfConversation	Indicates the end of a conversation.

4 Third Party Integration API

Third party integration API to send messages to customer on any channel via chat server and update delivery status as well.

4.0.1 Message API

The message REST API is used to send a message (such as a campaign message) to customers on different channels via chat server. The message will be submitted to the chat server via this API. The chat server will transmit this message to connector of the channel specified in the REST call. If the connector is not connected to the chat server, the server will returns the status 503 for the request.

Protocol	HTTP
URL	*http://<FQDN>/api/external/message*
Content Type	Application/JSON
HTTP Method	POST
Input/Output Format	JSON
Request Body	Following are the allowed parameters in the Request body. **channel** - Channel ID, a connector must be connected with Chat Server with this channel. **messageId** - a unique string to identify message **destination** - the reference of customer, to which this message will be delivered. Mobile number in case of SMS, facebook account id in case of Facebook message. **from.id** - the id of campaign or sender agent **from.name** - the display name of campaign or sender agent. **from.app** - name of the application which is sending message. **text** - text content of message to be delivered. **timestamp** - in the ISO 8601 format.

Sample Request Body	{
	"channel": "sms",
	"messageId": "abbcda323",
	"destination": "3015642191",
	"from": {
	"id": "sales_2018_winter",
	"name": "Sale Campaign",
	"app": "Expertflow Campaign Manager"
	},
	"timestamp": "2018-11-23T08:51:28.255",
	"text": "Expertflow campaign message"
	}

HTTP Response	200: Success 500: Internal Server Error 503: Service Unavailable

Response Header	requestId - Chat server will respond with requestId in response header, this id can be used to cross check the logs in both systems.

6 Agent API

The purpose of Agent API is to enable human agents to connect with customers and respond to their queries.

6.1 Events

Here is the list of all possible events which can be used as command sent to server and events received from server. Origin of each event is specified by Source.

Event Name	Description
amq-sub	This event is used to subscribe to ActiveMQ topic.
amq-send	This event is used to send ActiveMQ related messages.
amq-msg	This event is used to receive message from ActiveMQ
login	This event is used to login in the system
receiveChatRequest	The receiveChatRequest event is used to show a chat request from customer
acceptChatRequest	This event is used to accept the chat request. Client need to emit this event.
changeState	Client emit this event to change his state
addParticipant	This event is used to add a participant in the chat. Client needs to emit this event.
removeParticipant	This event is used to remove a participant from the chat. Client needs to emit this event.
transferChat	The transferChat event is used to transfer a chat from one agent to another agent/supervisor. Client needs to emit this event.
raiseHand	The raiseHand event is used to add supervisor in the chat as participant. Client needs to emit this event.
wrapUp	The wrapUp is used at the end of conversation to upload wrapup data. Client needs to emit this event.
silentMonitor	The silentMonitor event is used by supervisor to monitor any active conversation of some agent in his team. Client needs to emit this event.
bargeIn	The bargeIn event is used by supervisor to join the conversation. Client needs to emit this event.

messageArrived	The messageArrived event is used to receive messages from other participant. Client need to subscribe to this event.

6.1.1 amq-sub

This request is used for agent to subscribe on chat solution's activemq.

Protocol	Websockets
Event Name	amq-sub
Source	Client
Input/Output Format	JSON

Request Payload	{
	"AgentId": "abc",
	}

Payload Parameters	AgentId: Id of the agent to login
Event Triggered	

6.1.2 amq-send

For communication with media routing engine, chat solution uses activemq. Any message which needs to be passed to activemq is sent through amq-send event.

Protocol	Websockets
Event Name	amq-send
Source	Client
Input/Output Format	JSON

Request Payload	{
	"head": "Login",
	}

Payload Parameters	head: this is the JMS type of activemq message There will be other parameters of the request as well inside request payload.
Event Triggered	

6.1.3 amq-msg

This event is used for sending activemq message data to client.

Protocol	Websockets
Event Name	amq-msg
Source	Server

Input/Output Format	JSON

Request Payload	<pre>{
	"head": "AgentInfo",
	}</pre>

Payload Parameters	head: this is the JMS type of activemq message
Event Triggered	

6.1.4 login

This request is used for agent to login in chat solution.

Protocol	Websockets
Event Name	login
Source	Client
Input/Output Format	JSON

Request Payload	<pre>{
	"agentId": "abc",
	"password":"*******",
	"mrd": "mrd-id"
	}</pre>

Payload Parameters	agentId: Id of the agent to login Password: password of the agent Mrd:Id of the Mrd in which agent want to login
Event Triggered	messageArrived type: activityMessage

6.1.5 changeState

It allows the agent to change state between ready and not-ready.

Protocol	Websockets
Event Name	changeState
Source	Client
Input/Output Format	JSON

Request Payload	<pre>{
	"state": "NOT_READY",
	"mrd": "mrd-id",
	}</pre>

Payload Parameters	state: agent requested state
Event Triggered	messageArrived type: activityMessage

6.1.6 receiveChatRequest

When agent is in ready state, receives a new chat request from customer,

Protocol	Websockets
Event Name	receiveChatRequest
Source	server
Input/Output Format	JSON

Request Payload	{
	"customer": "customer object",
	}

Payload Parameters	customer: Object of the customer containing customer information and conversationId
Event Triggered	messageArrived type: activityMessage

6.1.7 acceptChatRequest

Agent can accept the chat request once receives a chat request. After accepting the chat request, chat will be started between agent and customer.

Protocol	Websockets
Event Name	acceptChatRequest
Source	client
Input/Output Format	JSON

Request Payload	{
	"conversationId": "abc333"
	}

Payload Parameters	conversationId:
Event Triggered	

6.1.8 addParticipant

Once chat is active between agent and customer, agent can add other participants in the chat.

Protocol	Websockets
Event Name	addParticipant
Source	client
Input/Output Format	JSON

Request Payload	{
	"participant": "participant object"
	}

Payload Parameters	participant: participant object having participant information such as agentId etc.
Event Triggered	messageArrived type:activityMessage activityType: participantAdded

6.1.9 removeParticipant

Agent, who first received the chat request from customer is the owner of the conversation. If there are multiple participants in the conversation other than customer than agent (owner of the chat) can remove any participant from the conversation other than customer.

Protocol	Websockets
Event Name	removeParticipant
Source	client
Input/Output Format	JSON

Request Payload	{
	"participantId": "participantId",
	"conversationId": "conversationId"
	}

Payload Parameters	participantId: Id of the participant conversationId: Id of the conversation
Event Triggered	messageArrived type: activityMessage activityType: participantRemoved

6.1.10 transferChat

Once agent has added other participants in the chat, he can transfer chat to any of the participants (agents/supervisor).

Protocol	Websockets
Event Name	transferChat
Source	client
Input/Output Format	JSON

Request Payload	<pre>{
	"participantId": "participantId",
	"conversationId": "conversationId"
	}</pre>

Payload Parameters	participantId: Id of the participant conversationId: Id of current conversation
Event Triggered	

6.1.11 raiseHand

During a chat, if agent needs help, he can raise hand. By raising hand a chat request will be sent to his supervisor. Once supervisor accepts the request, chat history will be visible to him and now he can help the agent.

Protocol	Websockets
Event Name	raiseHand
Source	client
Input/Output Format	JSON

Request Payload	<pre>{
	"conversationId": "conversationId",
	"participantId": "supervisorId"
	
	}</pre>

Payload Parameters	participantId: Id of the supervisor conversationId: Id of current conversation
Event Triggered	

6.1.12 wrapUp

At the end of conversation with the customer, agent can submit wrap-up.

Protocol	Websockets
Event Name	wrapUp
Source	client

Input/Output Format	JSON

Request Payload	<pre>{
	"conversationId": "conversationId",
	},
	"wrapup": "wrapup text",</pre>

Payload Parameters	conversation: conversation object containing conversationId wrapup: wrapup reason text
Event Triggered	

6.1.13 silentMonitor

Supervisor can see all active conversations of his team. He can select and monitor any of the active conversation but cannot participate in the conversation. This is called silent monitoring.

Protocol	Websockets
Event Name	silentMonitor
Source	client
Input/Output Format	JSON

Request Payload	<pre>{
	"conversationId": "conversationId",
	}</pre>

Payload Parameters	conversation: conversation object containing conversationId
Event Triggered	

6.1.14 bargeIn

During silent monitoring, if supervisor wants to participate in the conversation, he will use bargeIn functionality. An activity message will be sent to the participants of the conversation other than customer that supervisor has joined the conversation. Now supervisor can send message as well.

Protocol	Websockets
Event Name	bargeIn
Source	client
Input/Output Format	JSON

Request Payload	<pre>{
	"conversationId": "conversationId",
	}</pre>

Payload Parameters	conversation: conversation object containing conversationId

Event Triggered	messageArrived type: activityMessage activityType: participantJoined

6.2.1 messageArrived

The messageArrived event is used to receive messages from other participant. Client need to subscribe to this event.

Protocol	Websockets
Event Name	messageArrived
Source	Server
Input/Output Format	JSON

Event Payload (for ChatMessage)	{
	"conversationId": "7g5639ju987g",
	"type": "ChatMessage",
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z",
	"channel": "web",
	"from": {
	"id": "abcdedf",
	"name": "Kashif Sohail"
	},
	"to": [
	{
	"id": "yferdt5"
	},
	{
	"id": "4ty678gu"
	}
	],
	"text": "Hello, what is the status of my request",
	"attachments": []
	}

Payload Parameters	conversationId: The id of conversation to which message is being sent type: Type of message channel: The channel on which messaging
Event Triggered	

6.3 Schema

6.3.1 Participant Object

Defines a participant of the conversation and who can send or receive messages.

Property	Type	Description

displayName	String	Name of the participant to be displayed to other participants of the conversation
type	String	Type of participant. It can be **Agent, Supervisor, Bot** or **Customer**
Role	String	Role of the participant. **Owner** or **Participant**
status	String	Status of the participant. **Ready, Active, Busy**

6.3.2 Conversation Object

Defines a conversation which contains information of participants and messages being sent among participants.

Property	Type	Description
Id	String	The ID that identifies the conversation. The ID is unique per conversation. This conversationId is generated from service side and sent to client when the client request initiate chat
type	String	Type of Message. ActivityMessage in this case
participants	Participant[]	Array of **Participant** objects.
activityType	String	Type of activity. One of these values: **greetings**, **typing**, **participantAdded**, **participantRemoved**, **participantJoined**, **participantLeft**, **endOfConversation** For details about activity types, see AcitivityTypes
messages	Message[]	List of messages exchanged in current conversation.

6.3.3 Customer Object

Defines a customer which is also a participant of the conversation.

Property	Type	Description
displayName	String	Name of the participant to be displayed to other participants of the conversation
email	String	Email of the customer
phone	String	Phone number of the customer
channel	String	The channel on which messaging

6.3.4 Agent Object

Defines an agent who is a participant of the conversation.

Property	Type	Description
displayName	String	Name of the participant to be displayed to other participants of the conversation
type	String	Type of participant. In this case it will be **Agent**.
role	String	Role of the participant. **Owner** or **Participant**
status	String	Status of the participant. **Ready, Active, Busy**
teams	Team[]	List of teams in which agent is the member
conversations	Conversation[]	List of active conversations.

6.3.6 Supervisor Object

Defines supervisor, he is also a participant of the conversation.

Property	Type	Description
displayName	String	Name of the participant to be displayed to other participants of the conversation
type	String	Type of participant. In this case it will be **Agent**.
role	String	Role of the participant. **Owner** or **Participant**
status	String	Status of the participant. **Ready, Active, Busy**
team	Team	Supervisor's team, in which his role is Supervisor

6.3.7 Team Object

Defines team, an agent and supervisor are always part of some team.

Property	Type	Description
id	String	Each team has always a unique identification.
name	String	Name of the team
members	Agent[]	Agents are added as members of the team. A team will consists of multiple agents
supervisors	Supervisor[]	List of supervisors. A team can have 1 or more supervisors

6.3.8 ActivityMessage object

Defines a activity message that is exchanged between server and client.

Property	Type	Description

conversationId	String	The ID that identifies the conversation. The ID is unique per conversation. This conversationId is generated from service side and sent to client when the client request initiate chat
type	String	Type of Message. ActivityMessage in this case
from	Participant	A **Participant** object that specifies the sender of the message
to	Participant[]	Array of **Participant** objects that specifies the recipients of the message. If the list is empty, the message will be broadcasted in the conversation.
activityType	String	Type of activity. One of these values: **greetings**, **typing**, **participantAdded**, **participantRemoved**, **participantJoined**, **participantLeft**, **endOfConversation** For details about activity types, see AcitivityTypes
info	Object	Notification data.

7 Chatbot API

The chatbot API facilitates the developers to integrate a chatbot with ExpertFlow Chat server and make the bot available for conversation.

Connecting a bot to Chat server

In order to communicate with ExpertFlow Chat server the chatbot API should be a socket.io client and it should send a registerBot request to make itself available for chat.

Send/Receive messages via Websocket

Client have to implement initConversation to start the conversation, the response includes ***conversationId*** *and* ***participant***. Client needs to emit joinConversation in order to be a participant of the conversation.

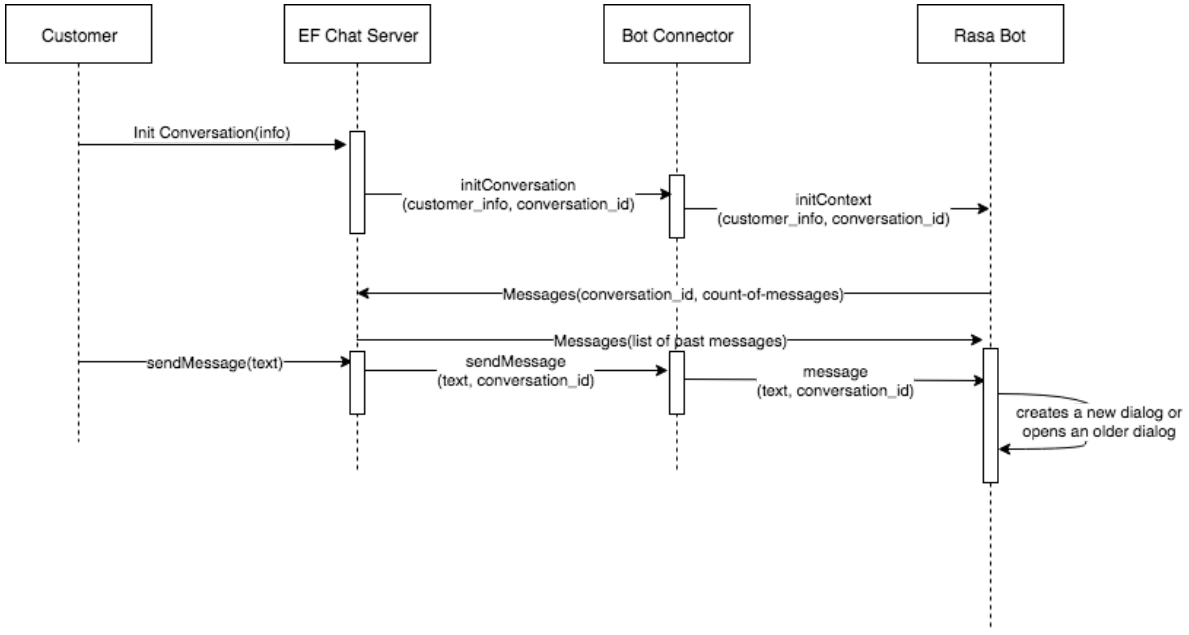
Receive messages

Client needs to implement messageArrived event for the conversation. Payload for this event is a message. The payload contains property **type** in the JSON root, which can be either ChatMessage or ActivityMessage.

Send Messages

Sending message is quite similar to receive messages. To send a message, client needs to emit sendMessage event with payload BotMessage.

Communication Flow



Events

Here is the list of all possible events which can be used as command sent to server and events received from server. Origin of each event is specified by Source.

Event Name	Description
registerBot	This event register's a bot with EF chat server to make the bot available for chat. Client need to emit this event.
initConversation	This event initiates a conversation so that bot can start communicating.
joinConversation	This event is used to join the conversation to send and receive messages. Client need to emit this event.
messageArrived	This event is used to receive messages from other participant. Client need to subscribe to this event.
sendMessage	This event is used to send message to a conversation. Client need to emit this event.
endConversation	This event ends an active conversation.

7.0.2 registerBot

This event register's a bot with EF chat server to make the bot available for chat. Client need to emit this event. The initializations required for session or context maintenance can be done in this event.

Protocol	Websockets
Event Name	registerBot
Source	Client
Input/Output Format	JSON

Event Payload	{
	"id": Bot",
	"name":"Rasa Bot",
	"type":"Rasa"
	}

Payload Parameters	id: The id of bot to be registered name: Display name of bot type: The type of the bot to be registered
Event Triggered	

7.0.3

7.0.4 initConversation

The connector will receive this event when a new conversation is initiated on chat server for a customer. It comes along with the conversationId and customer information. At the point, we expect

from bot is to expose an api to create a session/dialog/context on the bot side if not exist already and load it in other case. The Bot connector will invoke Bot's initContext api to achieve this goal.
The connector need to emit joinConversation in order to be the participant of this particular conversation to receive chat messages. The session maintenance and context maintenance can be done in this event.

Protocol	Websockets
Event Name	initConversation
Source	Server
Input/Output Format	JSON

Event Payload	{
	"conversationId": "-OPoRo7eDS",
	"customerInfo": {
	"accountId":"LK9VmDiwM",
	"name":"Ali",
	"email":"abc@gmail.com",
	"accountBalance":"$350",
	"phone":"12345",
	"channel":"web"
	}
	}

Payload Parameters	**conversationId**: The id of conversation to which message is being sent
Event Triggered	joinConversation

7.0.5

7.0.6 joinConversation

This event is used to join the conversation to send and receive messages. Client need to emit this event.

Protocol	Websockets
Event Name	joinConversation
Source	Client
Input/Output Format	JSON

Request Payload	{
	"conversationId": "-OPoRo7eDS",
	"participant": {
	"id":"bot",
	"name":"Rasa bot"
	},
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"

	}
Payload Parameters	**conversationId:** Id of the conversation to join **id:** Participant id, supplied in response of initConversation **name:** Name of participant to display in conversation
Event Triggered	messageArrived type: ActivityMessage activityType: participantJoined

7.0.7 messageArrived

This event is used to receive messages from other participant. Client need to subscribe to this event.

Protocol	Websockets
Event Name	messageArrived
Source	Server
Input/Output Format	JSON

Event Payload (for ChatMessage)	{
	"conversationId": "7g5639ju987g",
	"type": "ChatMessage",
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z",
	"from": {
	"id": "abcdedf",
	"name": "Kashif Sohail"
	},
	"to": [
	{
	"id": "yferdt5",
	"name":"Alice"
	},
	{
	"id": "4ty678gu",
	"name":"Bob"
	}
	],
	"text": "Hello, what is the status of my request",
	"attachments": []
	}

Payload Parameters	conversationId: The id of conversation to which message is being sent type: Type of messages are ChatMessage , ActivityMessage or BotMessage.
Event Triggered	

7.0.8

7.0.9 sendMessage

This event is used to send message to a conversation. Client need to emit this event.

Protocol	Websockets
Event Name	sendMessage
Source	Client
Input/Output Format	JSON

Event Payload	{
	"conversationId":"1L3TP8jvjw",
	"messageId":"h7VcOjUimR",
	"type":"BotMessage",
	"from":{
	"id":"ZaWmuxevX",
	"name":"Ali"},
	"bot":{
	"id":"bot",
	"name":"EF Rasa Bot",
	"type":"Rasa"},
	"timestamp":"2018-08-28T10:47:55.823Z",
	"input":" what would be the price of 100 agents",
	"intents":[
	{
	"confidence":0.23290375626207,
	"name":"offer_requested"
	}],
	"entities" :[
	{
	"confidence":0.5421172436275492,
	"end":31,
	"entity":"NbrAgents",
	"extractor":"ner_crf",
	"start":28,
	"value":"100"
	}],
	"output":[
	{
	"name":"Sorry I could not recogonize 'None'. Please repeat your question, this time, clearly mention your product / service of interest!",
	"confidence":0.999995648695882
	},
	{
	"name":"Please repeat your question, this time, also mention your product / service of interest!",
	"confidence":0.0026695198393466
	}]
	}

Payload Parameters	**conversationId:** The id of conversation to which message is being sent **type:** Type of messages are ChatMessage , ActivityMessage or BotMessage. **from:** The sender participant **text:** text content of message.
Event Triggered	

7.0.10 endConversation

This event ends an active conversation. The garbage collection and reinitialization of session or context maintenance can be managed here.

Protocol	Websockets
Event Name	endConversation
Source	Server
Input/Output Format	JSON

Event Payload	{
	"conversationId": "7g5639ju987g",
	"requestId":"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"
	}

Payload Parameters	**conversationId:** Id of the conversation to end
Event Triggered	

7.0.11 Past Messages

To load message history of a conversation.

Protocol	HTTP
URL	*http://<FQDN>/api/conversation/past-messages*
Content Type	Application/JSON
HTTP Method	GET
Input/Output Format	JSON
Request Body	Following are the allowed parameters in the Request body. **conversationId:** represents a session on chat solution. The chat solution returns that on a conversation initialization. **count:** Number of messages to fetch

	time: The time in ISO 8601 format. The solution returns past messages older than this time specified. To get last 10 messages, specify the time as current time and count should be 10.

Sample Request Body	{
	"conversationId": "-OPoRo7eDS",
	"count":10,
	"time": "2018-05-07T06:52:21.661Z"
	}

HTTP Response	200: Success 500: Internal Server Error 503: Service Unavailable 404: Conversation Not found
Response Body	The response object contains conversationId, participant list for this conversation, and messages. The messages is an array of ChatMessage

Sample Response Body (in case of success)	{
	"conversationId": "5ae01d346c25be4319a9df95",
	"participants": [
	{
	"id": "d346c25",
	"name": "Kashif Sohail",
	"type": "Customer"
	},
	{
	"id": "88hfyw6",
	"name": "Awais Aslam",
	"type": "Agent"
	}
	],
	"messages": [
	{
	"conversationId": "7g5639ju987g",
	"type": "ChatMessage",
	"requestId":
	"99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z",
	"from": {
	"id": "d346c25",
	"name": "Kashif Sohail"
	},
	"to": [
	{
	"id": "88hfyw6",
	"name": "Awais Aslam"
	},
	{
	"id": "4ty678gu",
	"name": "Bob"
	}
	],
	"text": "Hello, what is the status of my
	request",
	"attachments": []
	},
	{

	{
	"conversationId": "7g5639ju987g",
	"type": "ChatMessage",
	"requestId": "99397bf7-a4fb-41c3-8292-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z",
	"from": {
	"id": "abcdedf",
	"name": "Kashif Sohail"
	},
	"to": [],
	"text": "Your request is awaiting approval from manager.s",
	"attachments": []
	}
	],
	"requestId": "8876553-a4fb-41c3-hg54-a021b44262a6",
	"timestamp": "2018-05-07T06:52:21.661Z"
	}

Sample Response Body (in case of failure)	

7.0.13 InitContext

The initContext REST API will be used to initiate a customer's context on the bot side. The bot connector will call this when it receives initConversation from the chat server. The bot may initialize a dialog/context in it's memory if this conversationId is not seen before or load if it already exists.

Protocol	HTTP
URL	*http://<Bot-FQDN>/api/initContext*
Content Type	Application/JSON
HTTP Method	POST
Input/Output Format	JSON
Request Body	Following are the allowed parameters in the Request body. **conversationId:** represents a session on chat solution. Unique per customer. **customerInfo:** A complete object contains customer information provided by chat initiating component.

Sample Request Body	{
	"conversationId": "-OPoRo7eDS",
	"customerInfo": {
	"accountId":"LK9VmDiwM",
	"name":"Ali",
	"email":"abc@gmail.com",
	"accountBalance":"$350",
	"phone":"12345",

	<pre>"channel":"web"
	}
	}</pre>

HTTP Response	200: Success 500: Internal Server Error 503: Service Unavailable