Avaya Aura® Call Center
Elite Multichannel
Web Chat Web Service Definition

Table of Contents

Introduction **3**
High level overview 3
EMC Components for Web Chat 3
Chat Features implemented in the Sample Client 4
Additional features of Sample Client 4
The Sample Client Application (ASP.Net) 4
Web Service methods available in Web Chat Web Service **5**
Details of current Web Service methods 6
Web Methods consumed by Web Chat Web Client 7
Web Methods consumed by EMC Web Chat Gateway 13
Sample EMC Web Chat ASP Design **14**
Details of implementation of Sample Client 14
Web Chat Customizations **19**
Adding additional Text Boxes in Chat UI 19
Localization of Web Chat UI 25

Introduction

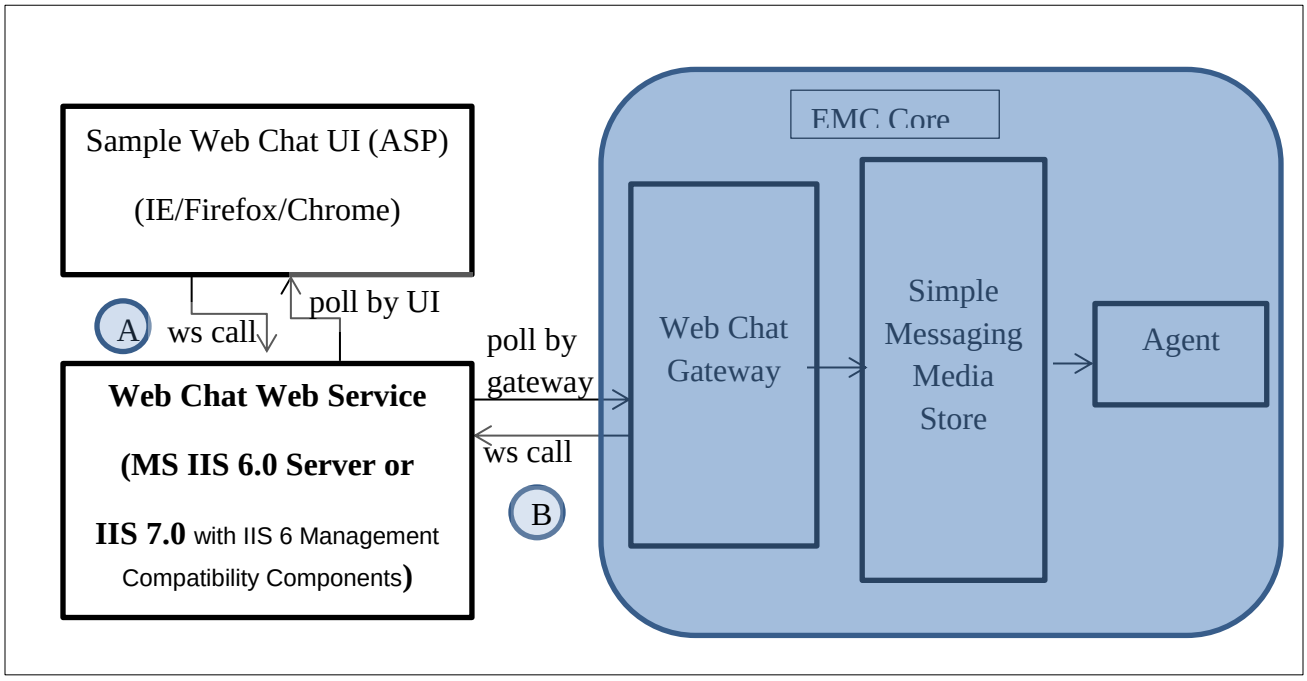
High level overview

Avaya CC Elite Multi-Channel Product (EMC) ships a sample implementation of web chat client along with the product. This document provides an overview of the underlying Web Chat service that this sample client consumes to implement end to end chat feature. This will help customized web chat interface developer to get started and understand the underlying web service usage.

EMC Components for Web Chat

There are 3 high level components in EMC that enable the web chat

1. A Sample Client Application (ASP.Net), which consumes the Web Service
2. The EMC Web Chat Web Service (runs off MS IIS 7.0) which provides chat service API documented below
3. The EMC Web Chat Gateway, which connects the chat sessions to agents



Sample Web Chat Implementation

Chat Features implemented in the Sample Client

1. Get service for customer
2. Check available service status
3. Start Conversation
4. Receive system messages for customer
5. Receive Customer messages
6. Receive and Send Typing Notifications
7. Receive reply from Agent
8. Close chat session

Additional features of Sample Client

1. Support for other languages in sample client
2. Save transcript on customer side

The Sample Client Application (ASP.Net)

The sample client shipped along with EMC is an ASP.Net based web chat application.

- ASP.Net
- Uses SOAP based Web Services
- Published on Microsoft IIS Web Server

Out of the box, the sample client runs on same IIS server as the Web Chat Web Service.

Web Service methods available in Web Chat Web Service

The Web Chat service is SOAP based Web Service

(A) Web methods consumed by sample client:

1. CheckServiceStatus
2. OpenSession
3. CloseSession
4. ReceiveMessagesForUser
5. SendMessageToService
6. SendNotificationToService

Details of these methods can be found later in the document. These are the calls that any custom implementation will use.

(B) Web methods consumed by Web Chat Gateway:

1. RecieveMessagesForService
2. SendMessageToUser
3. serviceLogin
4. serviceLogout

Only high level information of these methods is documented later. These will not be required for custom implementations, but are included here only for completeness of flow.

Details of current Web Service methods

Introduction

Web Chat Web Client (WCWC) is an application that enables to connect to EMC Web Chat Servers. WCWC Application could be built on any platform supporting consumption of XML Web Services (ASP.Net, Java etc.). Web Chat Client consumes the EMC Web Chat Web Service (WCWS).

Web Methods available in Web Service:

Web Chat Web Service contains 2 groups of methods. Details of methods can be found next.

Web Methods consumed by Web Chat Web Client

Following Web Service methods are used on a WCWC:

- **`CheckServiceStatus`**
- **`OpenSession`**
- **`SendMessageToService`**
- **`RecieveMessagesForUser`**
- **`CloseSession`**
- **`SendNotificationToService*`**

*Notifications received using **`RecieveMessagesForUser`** method. See User Messages for details in Server Messages section.

1. **CheckServiceStatus:**

   The CheckServiceStatus Web Method checks the status for each of the service in the ServiceIDs list and returns a list of serviceStatuses.

```
string[] CheckServiceStatus(string userID,
                            string password,
                            string[] ServiceIDs)
```

string userID -	Currently not used
string password -	Currently not used
string[] ServiceIDs -	Array of Service IDs to check

Returns: Array of status messages for the services
(See Server Messages Section below)

2. **OpenSession:**
   Used by the Web Chat ASP to request a new session for a service.

```
string OpenSession(string serviceID,
            string userID,
            string password,
            string initialQuestion,
            string cultureID,
            int priority,
            int updateInterval)
```

string serviceID -	Service ID to open session on
string userID –	Valid text username i.e. name of customer as seen on Agent side. (can pass an empty string)
string password -	Currently not used.
string initialQuestion-	Plain text initial question or XML formatted string containing additional customer data (see Session Data below)
string cultureID -	Standard or Custom (defined in Simple Messaging Media Store) Culture Name. This setting is used to select language for Customer Progress Messages. E.g. specify fr-FR or fr-FR-CCE-SMS as CultureID will display French progress messages from Simple Messaging Media Store. Please refer the section "**Changing the language culture for SMMS Queue"** below for more information.
int priority -	Priority of services in EMC Valid values 1-* where 1 is the highest priority, 5 is normal
int updateInterval -	Session timeout on the Service side (in sec); if session is not polled during the period, it gets destroyed; default = 5 sec (if not returned in 5 sec, chat will be destroyed).

Returns: New session unique ID string
If session ID is null or empty, treat it as open session has failed and close the chat.

Session Data XML string.

Session data parameter could contain both plain text initial question and more complex XML formatted string. The format of customer parameters to pass to the EMC is as following:

```
<?xml version=\"1.0\" encoding=\"utf-8\" ?>
    <WebChatWebSite>
        <InitialQuestion>Hi there</InitialQuestion>
        <MimeType>text/plain</MimeType>
        <UserParameter1>User Parameter 1 Value</UserParameter1>
        <...>
        <UserParameterN>User Parameter N Value</UserParameterN>
    </WebChatWebSite>
```

where UserParameter1 is the XML compliant node name.

3. **SendMessageToService:**
   Used by WebChat ASP to send message to the Web Service.

```
void SendMessageToService(string SessionID,
                          string message)
```

string SessionID -	Session ID returned by OpenSession
string message -	Plain text message to send to the service, supports only text based messages

4. **RecieveMessagesForUser:**
   Used by WebChat ASP to receive messages for the client from the service.

```
string[] RecieveMessagesForUser(string ServiceID,
                                string SessionID)
```

string ServiceID -	Service ID
string SessionID -	Session ID returned by OpenSession

Returns: Array of User Messages (See Server Messages section below).

5. **CloseSession:**
   Used by WebChat ASP to close session.

```
void CloseSession(string serviceToken,
                  string SessionID,
                  string reason)
```

string serviceToken - Currently not used (can pass an empty string)
string SessionID - Session ID returned by OpenSession
string reason - Plain text reason why the session gets closed (currently not used)

6. **SendNotificationToService:**
   Used by WebChat ASP to send typing notification to the Web Service.

```
void SendNotificationToService(string SessionID,
                               string userID,
                               int notificationCode)
```

string SessionID - Session ID returned by OpenSession
string userID - Valid text username
i.e. name of the customer as seen on Agent side.
(can pass an empty string)
int notificationCode – 0 when Customer is typing
1 when Customer has stopped typing.
(Need to set a timer of 5 seconds wherein pass value 0 as notification code when customer is typing and pass value 1 five seconds after customer has stopped typing to notify the service that customer has stopped typing).

Server Messages:

Every message received from the WCWS is a string containing 2 or 3 fields separated by colon (':')

1st field is the session ID

2nd field is a command

3rd field (optional) is the message body

1) **Status Messages**: (returned by CheckServiceStatus method)

   For E.g.:

   092357316fe6469bad4f7af09b796c5d:QueryQueueStatusReturn:QueueID=Sales,OpenState=Open,UserMessage=

   Commands: *"WsError", "QueryQueueStatusReturn"*

   Message body:

   "WsError" - Plain text message

   "QueryQueueStatusReturn" - Contains Key-Value pairs delimited by comma (',').

   Keys present are:-

   - QueueID: Specifies Service IDs .
   - OpenState: Specifies state of the service.

     Values are: *"Open", "Offline", "No Services", "Disconnected", "OutOfHours", "Closed"*.

     If value is "*Open"* then session can be opened. For all other values session cannot be started.
   - UserMessage: Specifies service message.

2) User messages: (returned by RecieveMessagesForUser method)

Commands: *"SessionOpened", "SessionState", "StdMessage","TypingNotification", "CloseSession", "WsError"*

- **SessionOpened**:
  f7e5b89605194c2f95be37a49c1679ff:SessionOpened: Good Afternoon! Welcome <CustomerName>! We look forward to being of assistance.
  Specifies that the chat session is now opened.
  Message Body: System Message
- **SessionState**:
  f7e5b89605194c2f95be37a49c1679ff:SessionState: You are first in the queue.
  f7e5b89605194c2f95be37a49c1679ff:SessionState: The conversation request is being delivered to an agent. Please wait a second.
  f7e5b89605194c2f95be37a49c1679ff:SessionState: The conversation request has been accepted by an agent. Please start the conversation.
  Specifies that conversation can now be started.
  Message Body: System Message
- **StdMessage**:
  f7e5b89605194c2f95be37a49c1679ff:StdMessage:<Agent Message>
  Specifies that the message is sent from the Agent and not a system generated one. Message can contain URLs and Email Addresses.
  Message Body: Agent Typed Message
- **TypingNotification**:
  f7e5b89605194c2f95be37a49c1679ff:TypingNotification:TypingUser
  Specifies that the agent is typing something.
- **CloseSession**:
  f7e5b89605194c2f95be37a49c1679ff:CloseSession:The conversation session has been closed. Thank you.
  f7e5b89605194c2f95be37a49c1679ff:CloseSession
  Specifies that the chat conversation can now be closed.
  Message Body: System Message
- **WsError**:
  f7e5b89605194c2f95be37a49c1679ff:WsError
  Specifies that an error has occurred and chat must be destroyed.

Web Methods consumed by EMC Web Chat Gateway

- **`serviceLogin`**
- **`serviceLogout`**
- **`SendMessageToUser`**
- **`RecieveMessagesForService`**

1. **serviceLogin:**
   Used by Web Chat Gateway to login the services to Web Service.

   Input parameters:
   - String serviceID
   - String password
   - Int updateInterval

   Output:
   - String serviceToken

2. **serviceLogout:**
   Used by the WebChat Gateway to logout the service with the serviceToken.

   Input parameters:
   - String serviceToken

   Output:
   - Bool – true for successful. Otherwise false.

3. **SendMessageToUser:**
   Used by WebChat Gateway to send message to WebChat ASP client.

   Input parameters:
   - String serviceToken
   - String serviceID
   - String message

4. **RecieveMessageForService:**
   Used by the WebChat Gateway to receive messages.

   Input parameters:
   - String serviceToken
   - String status
   - String statusMessage

   Output
   - List<string> messageList

Sample EMC Web Chat ASP Design

Details of implementation of Sample Client

1. Get service for customer:

In Default.aspx.cs file,

- Create a Session for Client
    - Set all its properties from the WebChatClient.cs file in App_code folder and web.config file.
    - Here, we set different properties for Client like:
        - CultureId,
        - Priority,
        - Polling interval,
        - Selected service,
        - Chat service etc.

On the Page_load event,

- Call the PopulateDropDownList (ddlServices) method of the WebChatConfiguration.cs which gets the serviceName and serviceID of the services from the web.config file and adds the services to the dropdown list in the UI.

On Polling,

- The updateStatus () method of Default.aspx.cs file is called periodically.
- For the first time, we do not have any sessionID for our Client. Hence, it creates a list of available serviceID.

2. Check available service status:

- It then checks the status for each service, for this it calls the CheckServiceStatus (“”,””, servicesID.ToArray ()) API of the WebChatWS. It passes as parameters
    - username,
    - password,
    - array of serviceIDs which we get from the PopulateDropDownList (ddlServices) method.

  It returns array of messages with information like
    - queueID,
    - status,
    - user message etc.

  This array is passed to a function processStatus.

  The processStatus (messages) method sets the Client status to offline and calls the splitMessageFromWS (message) of WebChatUtility.cs which tells us of it is a WSError message of QueryQueueStatusReturn.

  For QueryQueueStatusReturn, we get information like
    - ServiceID,
    - ServiceStatus and
    - ServiceMessage.

  We then check the selectedServiceID and setStatus from the ServiceStatus returned from QueryQueueStatusReturn to Open, Close and Not Available. It also updates other information like clock, WCStatus Icon, statusMessage accordingly.

3. Start Conversation with Agent:

The customer UI then presents page to enter customer information like

- Username
- Select service
- Enter initial question and
- Launch Conversation button.

When clicked on the Launch Conversation button, It calls the CreateConversation () method.

The CreateConversation () opens the conversation window i.e. Conversation.aspx.cs file.

- Conversation.aspx.cs file has an onStart () method.
- The onStart () method gets the InitialQuestion by using getElementByID. And displays the initial question by using writeToScreen () method.
- It then calls the startChatSession method of the Default.aspx.cs and passes the initial question as parameter.

The startChatSession() takes the initial question and generates ssnData. It calls the addSessionData () method and passes initial question to this method. The addSessionData() adds information to ssnData like

- UserHostAddress,
- Browser version etc. and returns a new ssnData.

It stores the serviceID, cultureID, username from the UI and Client Session and calls the method startChatSession and passes parameters like serviceID, userID, initialQuestion and ssnData.

The startChatSession now sets Client's Status as Requesting Conversation.

It converts the ssnData to string format using the ConvertToBase64String () method. It then calls the OpenSession API of WebChatWS.

The OpenSession method takes as parameters the serviceID, userID, password, ssnData, cultureID, priority and poll interval. It returns a sessionID in response which is stored as Client.SessionID. We now get a SessionID which will be used for the chat session.

4. **Receive system messages for customer:**

When we start a chat session, we get system messages which are displayed to the customer.

The updateStatus () method is called periodically. It checks whether the chat has a sessionID. Now we have a sessionID, hence it polls the WebService to see whether there are any messages for the user. For this it calls the RecieveMessagesForUser API of the WebChatWS. It takes as parameter the selectedServiceID and sessionID of the Client. It returns an array of messages as response.

This array of messages is given to the processMessage () method. This method takes each message from the array and calls the splitMessageFromWS of WebChatUtility.cs which takes the message as parameter and gives command and body as output of the method. The command gives us information like whether it is sessionOpen, sessionClose, sessionStatus, stdmsg, closeSession or WSError. The body part will give us the message that is displayed to the customer.

5. **Receive Customer messages:**

When a customer wants to send another set of messages to the agent, he types the message and sends it by clicking on the Send Button of the Conversation.aspx.cs class. The onClick event of the Send Button calls the Send() method.

The Send() method gets the message using the value property of getElementByID of the page text box.

When it gets the message string, it performs a check to see the max message size. We set the maxMessageSize in Conversation.aspx.cs which gives us the maximum size of the message that can be sent. It the message size exceeds the maxMessageSize, we trim the extra message and send message with the length as max size.

Clear the sendMessage text box and make a call to the Send () method of Default.aspx.cs file. It takes as parameter the message to be sent.

The Send () method calls the SendMessage () method that takes as parameter the message to be sent. The SendMessage () method calls the BeginSendToService API of the WebChatWS. It takes as parameter sessionID and the message. It also displays the message in the chat window.

6. **Receive a reply from Agent:**

   The updateStatus() method is called periodically to check messages for the customer. It checks for the sessionID. If not empty or null it will call the RecieveMessageForUser API of the WebServiceWS. It takes as parameter the serviceID and sessionID of the client. It returns an array of message. This array of messages is passed to processMessage() method.

   The processMessage() method takes each message and passes this message to splitMessageFromWS() of WebChatUtility.cs which returns the command and body as reponse. Command gives the type of message. In this case it will be a stdmsg. The body gives the actual message which is then displayed to the customer in the Conversation window.

7. **Close chat session:**

   When there is no chat between the agent and the customer for a specified amount of time, the chat session is automatically closed. This is informed through system messages.

   The updateStatus() which is called periodically, checks if the sessionId is empty or null. If not, it calls the RecieveMessageForUser API of the WebChatWS. It passes as parameters the selectedServiceID and the sessionID. It returns an array of messages as a response.

   This array of messages is given to the processMessage() method. The processMessage() takes each message and passes it to the splitMessageFromWS() method of WebChatUtility.cs which gives the command and body as output of the method.

   In this case, we get the command as CloseSession. The sessionID is cleared and the client status is set to Conversation Closed. And the messages received in the body part are displayed to the customer.

   If the customer decides to close the chat session, she clicks on the Close button of the Conversation window. It calls the closeSession() method of the Default.aspx. We check the sessionID is null or empty. We have a sessionID, hence we call the BeginCloseSession() of the WebChatWS. It takes the sessionID as input parameter. It then clears the sessionID and sets the client chat status to Conversation Closed.

   Once the chat session is closed, it then registers all the services and displays the UI page to the customer.

Web Chat Customizations

Adding additional Text Boxes in Chat UI

Web Chat can be customized to request and send additional data from Customer to Agent. The additional data as part of all workitem data can be viewed /passed to an external application using external application configuration at Agent desktop or alternately displayed at the Agent desktop using custom plugins.

The example below illustrates how to add new Text Boxes and Labels to the default web chat page and pass those values to the desktop as Extra Parameters. The user Phone will be added in this example.

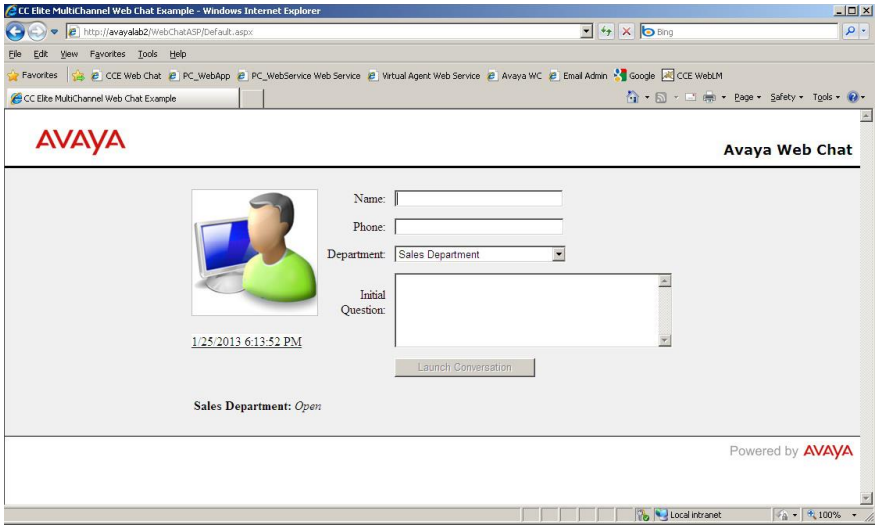
1. **Add additional Label and Textbox in UI – say "Phone"**

   Add Label in Resource file:
   Find the Default.aspx**.resx** file located (by default) in
   Drive:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Server\Media Gateways\Web Chat For IIS\WebChatASP\

```
<data name="DefaultLabelIntialQuestion" xml:space="preserve">
        <value>Initial Question: </value>
</data>
<data name="DefaultLabelName" xml:space="preserve">
        <value>Name:</value>
</data>
<data name="DefaultLabelPhone" xml:space="preserve">
        <value>Phone:</value>
</data>
```

   Adding Label and Textbox in UI:

```
<tr>
        <td style="width: 120px;text-align:right">
                <%=GetLocalResourceObject("DefaultLabelPhone")%>
        </td>
        <td>
                <asp:TextBox ID="txtPhone" runat="server" Width="211px"></asp:TextBox>
        </td>
</tr>
```



2. Add corresponding variable in the ASP file

Edit the Default**.aspx** page located in the *Drive*:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Server\Media Gateways\Web Chat For IIS\WebChatASP

Locate the function: function startChatSession(initialQuestion)
Add the highlighted custom code:

```
var serviceID = getSelectedService();
var cid = "en-US";
var uname = document.getElementById("txtName").value;

//Custom start
var uphone = document.getElementById("txtPhone").value;

ssnData = ssnData + formatXmlElement("UserPhone", uphone);
ssnData = ssnData + '</WebChatWebSite>';
//Customer End
```

Locate the function: function addSessionData(initialQuestion)
Comment or remove the line:

```
ssnData = ssnData + '</WebChatWebSite>';
```

3. Showing the Custom Data at Agent end

a. All work item data can be viewed / passed to an external application at Agent desktop

This is an external application container configuration in Agent desktop setting to popup work item data (xml format) in configured application.

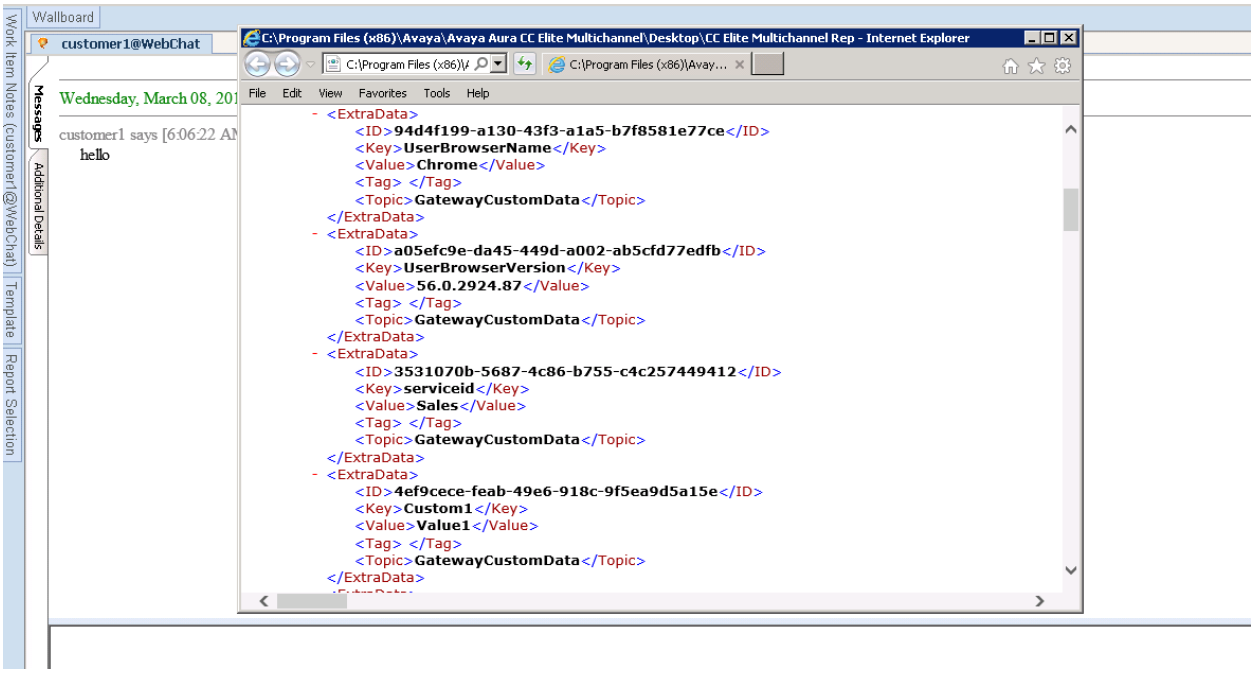
Steps to display data on Internet Explorer:

1) Close Desktop and open ASGUIHost.ini file from Desktop Directory
2) Configure the following sections as shown below:

[Simple Messaging]
Assembly File Name = ASSimpleMessagingPlugin.dll
Enable Error Logging = True
Active Window On Work Item Accepted = True
Enable External Application = True

[External Application Execute]
Assembly File Name = ASExternalApplicationExecutePlugin.dll
Enable Error Logging = True
Enable External Application = True
External Application File Name = iexplore.exe
XML File Path = C:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Desktop\CC Elite Multichannel Desktop\WorkItemXML
XML File Name = WorkItemData.xml
Delete XML Files On Exit = True

3) Open the Desktop.
4) Now when chat is invoked and reaches Agent Desktop, an IE Browser window will pop-up displaying the work item data in xml format as shown below.



b. Creating custom Plugin to Display additional data to Agent Desktop

Alternately, a custom plugin can be created, say CustomParamsPlugin.dll, to extract the custom Key “UserPhone” from the ExtraData and display on the Agent desktop.

The custom Key in the Extra Data is highlighted below:

```
DocumentWindowKey =64dcb5e2-09aa-4c0a-8789-138a2f3b0542
MediaType = 4
ExtraData Rows=[7]
QueueConfig Rows =[22]
myValue = [192.168.15.20]
myValue = [text/plain]
myValue = [IE]
myValue = [<?xml version="1.0" encoding="utf-16"?>
<GatewayDetails>
        <MediaGatewayID>1eb069e0-b6e9-4492-a3d1-1769e1ab0dae</MediaGatewayID>
        <MediaGatewayName>Web Chat Gateway@AVAYALAB2</MediaGatewayName>
        <MediaGatewayType>ASMG-WEB-CHAT</MediaGatewayType>
</GatewayDetails>]
myValue = [<?xml version="1.0" encoding="utf-16"?>
<InteractionData>
        <SessionData>
                <RemoteServiceID>Sales</RemoteServiceID>
                <RemoteServiceName>Sales</RemoteServiceName>
                <RemoteServiceType>Unknown</RemoteServiceType>
                <RemoteServiceTypeName>WebChat</RemoteServiceTypeName>
                <ChannelID>Default</ChannelID>
                <UserID>EMCUserID</UserID>
                <UserName> EMCUser </UserName>
                <CultureID>es-CO</CultureID>
                <QueueID>Sales</QueueID>
                <PriorityInQueue>5</PriorityInQueue>
                <ToAddress>Sales@localhost</ToAddress>
                <FromAddress>Martin</FromAddress>
                <GatewayCustomData>
                        <MimeType>text/plain</MimeType>
                        <UserIPAddress>192.168.15.20</UserIPAddress>
                        <UserBrowserName>IE</UserBrowserName>
                        <UserBrowserVersion>7.0</UserBrowserVersion>
                        <UserPhone>40002</UserPhone>
                </GatewayCustomData>
                <CustomAddressType>0</CustomAddressType>
        </SessionData>
</InteractionData>]
myValue = [40002]
myValue = [7.0]
```

For example, in the Custom Plugin, you can use the VoicePlugin.AddNumber method to populate the Dial pad with the value of the Key “UserPhone”.

```
Plugin.VoicePlugin.AddNumberToDialList(destNo, “WebChatCall”, true).ToString();
```

With this, the phone number and UUI (WebChatCall) can be automatically populated for the Agent to call the Web Chat client customer. The Agent can then simply click the Dialpad and a new outbound call will be initiated. (the existing phantom call for the Web Chat workitem will automatically be placed on hold).

Localization of Web Chat UI

The Web Chat interface is designed to support multiple languages, and includes the functionality to support custom languages. Resource files are used to store the displayed text, both from the Web Page itself, as well as the progress/canned messages that come from the Simple Messaging Media Store. The Resource files are in an XML format.

There are two locations where Resource files could be modified to suit a custom deployment:

1. WebChatForIIS (for ASP Web Chat Pages)

Each Web Chat application uses resource files which contain all the displayed text for the webpage interface. Modification of these resource files allows the appearance of the WebChat webpage to be altered.

There are two resource files that can be modified:

- default.aspx.resx: This contains the text that is displayed on the default webpage prior to initiating the conversation.
- conversation.aspx.resx: This contains the text that is displayed on the conversation web page while a conversation is in progress.

The Default Location for these resource files is

"C:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Server\Media Gateways\Web Chat For IIS\WebChatASP\App_LocalResources"

Out of the box, the WebChat for IIS installer installs English based resource files. These would need to be manually edited to support another language.

2. SimpleMessagingMediaStore (SMMS)

EMC Supports a predefined list of Languages. See EMC Install Guide for setting the culture for each SMMS Queue.

If the default translated messages need to be changed, this can be achieved using a Custom Language Culture.

Note: If this method is used then in every EMC release this Custom File will need to be merged manually otherwise the new localization updates will not work

You can change the Web Chat message by using the custom language resource file.

A Custom Language resource file is in a XML format, in which the individual strings can be customized. Each string is denoted by a data name parameter, which is a unique numeric identifier, and contains a value parameter, which is the string to display.

All the available localized strings are present in the following folder:-

C:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Desktop\CC Elite Multichannel Desktop\Custom Languages

Follow the below procedure to change the default strings:

1. Copy the corresponding language file from the above mentioned folder to C:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Server\Media Stores\Simple Messaging Media Store.

2. If we take an example of English it would be ASResource.custom_en.resx.

3. Edit the Resource file in the Simple Messaging Media Store directory. The sections of the resource file that are used by Simple Messaging as progress messages are from 103221 through to 103231, as well as 102838 through to 102858.

   Note: Only edit the text contained between the <value> and </value> identifiers. If any % variables are present in the string they should be kept intact.

   Eg:-

   ```
   <data name="_102841" xml:space="preserve">
     <value>The conversation request is going to be …</value>
   </data>
   ```

4. Save the file.

5. Edit the SMMS Queue and set the Culture for the Queue. You will find newly created entry “en-US” as the option.

6. When a new chat is invoked the custom language resource file would be used.

7. Restart SMMS.

Note: There are three locations from where culture for SMMS can be set. EMC Server will look for the culture from the following 3 files and in the following order:

1. web.config at *C:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Server\Media Gateways\Web Chat For IIS\WebChatASP*

```
<add key="ServicePriority" value="5" />
<add key="PollInterval" value="5" />
<add key="CultureID" value="custom_en" />
<add key="MaxMessageSize" value="1024" />
<add key="MimeType" value="text/plain" />
```

2. ASWebChatGateway.ini at *C:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Server\Media Gateways\Web Chat Gateway*

```
[Culture]
Culture = custom_en
```

3. ASSimpleMessagingMediaStore.ini at *C:\Program Files (x86)\Avaya\Avaya Aura CC Elite Multichannel\Server\Media Stores\Simple Messaging Media Store*

```
[QueueName]
Culture = custom_en
```

Language will be set to the first culture found.

If language needs to be set globally ie. for all SMMS Queues then either set the culture in web.config or ASWebChatGateway.ini file.

If different language needs to be set for different SMMS Queues then set the culture fields in SMMS Queues and set the culture field in web.config and ASWebChatGateway.ini to **empty** so that Server will pick it up from SMMS Queue.

For example if we want to set Custom English only for a particular SMMS Queue then the culture fields will look like below:

1. web.config:

```
<add key="ServicePriority" value="5" />
<add key="PollInterval" value="5" />
<add key="CultureID" value="" />
<add key="MaxMessageSize" value="1024" />
<add key="MimeType" value="text/plain" />
```

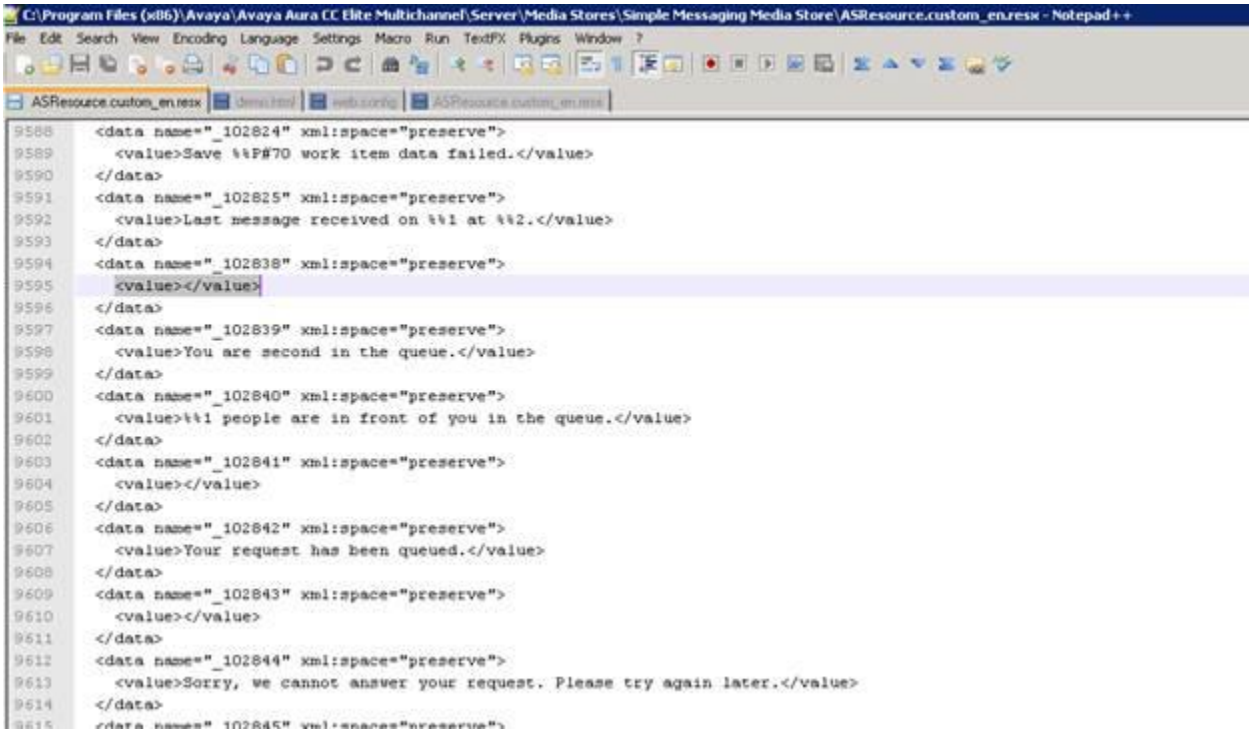
2. ASWebChatGateway.ini

   [Culture]
   Culture =

3. ASSimpleMessagingMediaStore.ini

   [QueueName]
   Culture = custom_en

Some of the text can be emptied out to remove some progress messages from the Chat UI. This can be achieved by using a Custom Resource file. Custom Resource file with default translations are already supplied above.



Note: Setting canned messages to empty from the Control Panel UI does not work. But the value can be changed in the database by editing the entries from the database table `[ASMSControl].[dbo].[CannedMessages]`.