Avaya Aura® Call Center Elite Multichannel - Software Development Kit

Release 6.6, Issue 1.0, Dated 04-Apr-2019

Contents

Chapter 1: About this document 4

Scope of this document 4

Intended audience.... 4

Chapter 2: Overview....5

Avaya Aura® Call Center Elite Multichannel overview 5

High Level Diagram 5

Key components 6

Software Development Kit (SDK) 6

EMC Desktop API Organization.... 7

Namespaces and classes 7

Dependencies used: 7

How the client application communicates with EMC.... 9

Sample Applications in EMC.... 11

Chapter 3: Installation Requirements 12

Software requirements.... 12

Chapter 4: Getting Started 13

EMC Work Flows 13

Voice Work Flow 13

Multimedia Work Flow 13

Steps to start building custom Agent Desktop application 14

Chapter 5: Sample Applications....18

5.1 Agent Login.... 18

Steps.... 18

Screenshot 23

Steps to run Agent Login Sample UI.... 24

5.2 Voice Call Handling.... 25

Steps.... 25

Screenshot 32

Steps to run Voice Call Handling Sample UI ... 33

5.3 Email Handling ... 36

Steps ... 36

Screenshot ... 47

Steps to run Email Handling Sample UI ... 48

5.4 SMS Handling ... 50

Steps ... 50

Screenshot ... 57

Steps to run SMS Handling UI ... 58

5.5 WebChat Handling ... 60

Steps ... 60

Screenshot ... 68

Steps to run Web Chat UI ... 69

5.6 PCMS Handling ... 71

5.7 Adding button to Ribbon on EMC Desktop ... 75

5.7.1 Existing Custom Plugins ... 75

5.7.2 Adding Buttons to Ribbon from Custom Plugins ... 76

5.7.3 Adding Buttons to Ribbon in New Custom Plugins ... 77

Chapter 6: Capacities ... 78

Chapter 7: References ... 78

Chapter 1: About this document

Scope of this document

This document provides information about Avaya Aura® Call Center Elite Multichannel (EMC) Software Development Kit (SDK). This document also includes the sample applications and provides the code that helps the customers, business partners, system integrators, and developers to develop their own customized agent desktop application using the **EMC Desktop API** for EMC.

Intended audience

This document is intended for customers, business partners, system integrators, and developers who want to develop their own customized agent desktop application for EMC.

Chapter 2: Overview

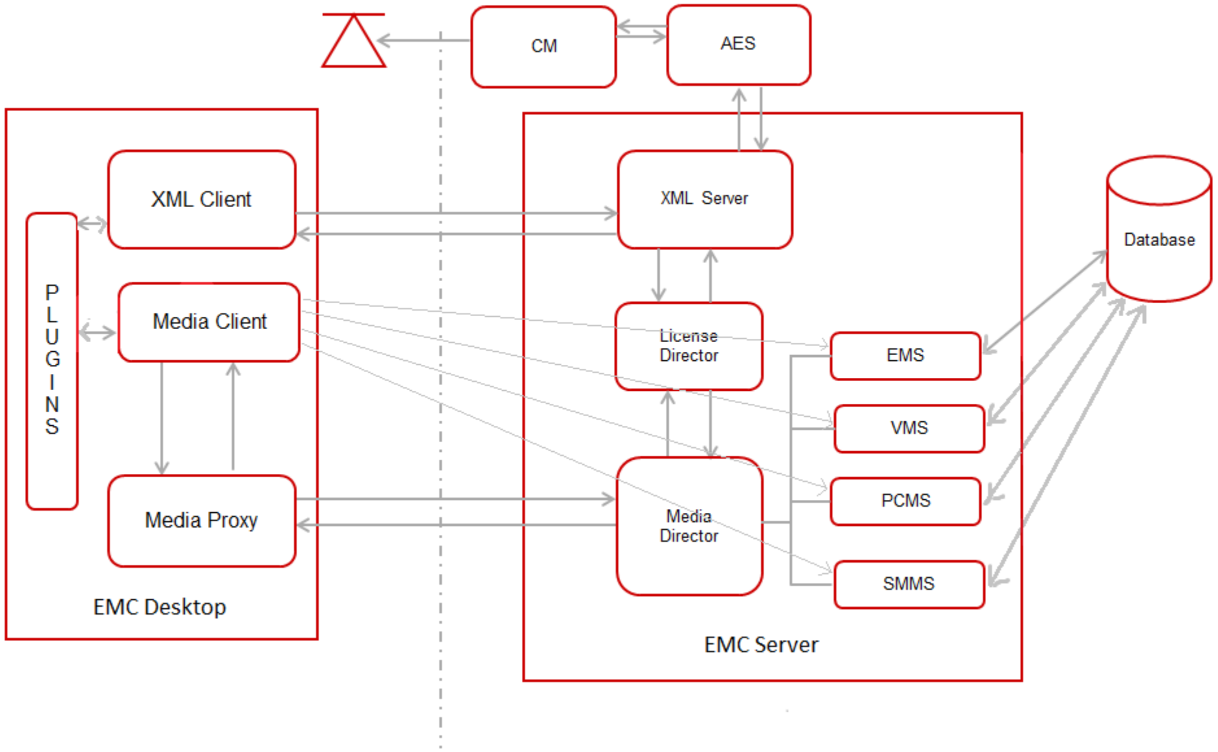
Avaya Aura® Call Center Elite Multichannel overview

EMC is a Microsoft Windows-based software product that converts traditional voice-based call centers into powerful multimedia contact centers. It supports inbound and outbound contacts.

EMC uses Avaya Aura® Call Center Elite so that the agents can communicate through various channels, such as phone, email, text, web chat, or instant messaging. Whether your customers prefer text messaging on mobile, sending emails or chatting over the Internet, their method of communication is treated as a phone call. The phone call is placed in a priority queue and distributed to an agent with relevant skills and knowledge. The agent can also reply using the same method of communication. Multimedia can take any form of communication for example – SMS, Email, Web Chat, and Gtalk. EMC supports integration of all the above.

High Level Diagram

The following diagram depicts the key components of EMC.



Key components

The following are the key components of EMC:

- **EMC Desktop Components**
  - **EMC Plugins -** for different groups or functionalities like handling voice and multimedia work items, updating customer information etc. EMC includes around 50 out-of-the-box plugins. These plugins aren't editable.
  - **EMC Desktop API (Interface to EMC Servers)**
    - XML Client - Connects EMC Desktop to XML Server for call routing
    - Media Client - Loads work items from Media Stores and the corresponding plug-ins
    - Media Proxy - Passes reference of Media Store to Media Client
- **EMC Server Components**
  - **XML Server** - Routes the call to appropriate XML Client of the agent
  - **License Director** - Provides licenses to all Server and Desktop components
  - **Media Director** - Media Director sends the Media Store work item reference to the corresponding Media Client of the agent desktop through Media Proxy
  - **Media Stores** - Act as interface points to corresponding external multimedia servers, such as line exchange server and chat web server, and also create work items for a new interaction and attach the customer contact history to the work item
    - Email Media Store (EMS)
    - Voice Media Store (VMS)
    - Simple Messaging Media Store (SMMS)
    - Preview Contact Media Store (PCMS)

The same **Desktop API** that EMC Desktop uses to connect to EMC Servers is also available for customized Desktop development.

Software Development Kit (SDK)

This EMC SDK Document includes the documentation and sample applications with source code to help the customers and developers start developing their customized agent desktop quickly.

The installer of EMC Developer installs the following components:

- XML Client
- Multimedia common libraries
- Plug-in common libraries
- Error logging

  Additionally, Desktop installer should be used to install addition required dlls and Media Proxy Service. (See Chapter 3 for more details on installation)

EMC Desktop API Organization

For EMC, there are two separate namespaces (Developer and Multimedia) that work together.

Namespaces and classes

The following namespaces are required for developing a customized desktop:

- **AgileSoftware.Developer**: The AgileSoftware.Developer namespace contains the classes and interfaces for creating applications that perform telephony operations. The major classes used are:
  - ASXMLClient
  - ASXMLStation

  ASXMLClient is the client component for communicating with Active XML Server.

  Binding with the ASXMLClient component, ASXMLStation performs the telephony operations on a voice station and manages the calls associated with it.

- **AgileSoftware.Multimedia:** The AgileSoftware.Multimedia namespace contains the classes and interfaces for creating applications that deliver multimedia work items. The major class used is:
  - ASMediaClient

  The ASMediaClient class allows the client applications to connect to Media Director through Media Proxy to receive the specified work items.

Dependencies used:

The following dependencies need to be added to the new applications:
(These need to be sourced from Desktop Installation folder and/or Developer Installation folder)

- Common dependencies
  - ASXMLClient.dll
  - CSTASchemas.dll
  - log4net.dll
    - This is open source library available from Apache Foundation. This is also included with the Sample Applications.

- Common Dependencies for Multimedia
  - ASGQEClientComm.dll
  - ASGQEDataStoreComm.dll
  - ASGQEGeneral.dll
  - ASMediaClient.dll
  - ASMediaControllerInterface.dll
  - ASMediaStorePluginInterface.dll
  - ASSimpleMessagingComm.dll
  - GenuineChannels.dll
- Specific Dependencies

  Email Handling Application

  - ASEmailMediaStore.Email.dll
  - ASEmailMediaStore.InboundEmail.dll
  - ASEmailMediaStore.Interface.dll
  - ASEmailMediaStore.WorkItem.dll
  - ASGUIHEmailControl.dll
  - GQEError.dll
  - ICSharpCode.SharpZipLib.dll

  SMS Handling Application

  - ASGQEDataStoreSimpleMessaging.dll
  - ASSimpleMessagingClientComm.dll
  - ASSimpleMessagingManagementCommon.dll
  - ASSimpleMessagingPlugin.dll
  - ASSMMediaServiceXmlUtilities.dll

  Web Chat Handling Application

  - ASSimpleMessagingClientComm.dll
  - ASSimpleMessagingManagementCommon.dll
  - ASSimpleMessagingPlugin.dll
  - ASSMMediaServiceXmlUtilities.dll
  - ASGQEDataStoreSimpleMessaging.dll
  - GQEError.dll
  - ICSharpCode.SharpZipLib.dll

How the client application communicates with EMC

Using EMC involves the following two logins:

- Login using the XML Client to login to XML server for Voice related behavior
- Login using the Media Client API to login to Media Director to retrieve the work items

For connecting the client applications to EMC, XML Client must connect to XML Server. Using a valid XML Server IP address and port number, an agent can browse through the available TServer links. The links returned can be secured or unsecured. The agent needs to choose one of the available links and connect to XML Server. Similar connectivity must be established with Media Director and Media Stores.

The following are the quick code snippets:

1. Initialize a new instance of AgileSoftware.Developer.ASXMLClient class.

   C# code:

```
//Create instance of ASXMLClient
ASXMLClient xmlClient;
xmlClient = newASXMLClient();
```

2. Browse for the available Server Telephony links.

   Connects to the XML Server name service using a valid XML Server IP address and port number to retrieve the available TServer link names and associated socket connection information. For XML Server, the default port number is 29096.

   C# code:

```
//Connects to the XML Server with Server IP address : "135.156.23.22" and port no. "29096" to
retrive available links
//Method used : XMLEnumerateServices()
//Positive acknowledgement : XMLEnumeratedServicesReturned event fires
//Negative acknowledgement : CSTAErrorReturned event fires
xmlClient.ServerIP = "135.156.23.22";
xmlClient.ServerPort = 29096;
xmlClient.XMLEnumerateServices();
```

   Connect to XML Server.

   Connects to the Active XML Server using the naming service. It establishes the TServer connection under the specified TServer link name.

   C# code:

```
//Connects to the XML Server using the specified TServer Link name returned by the
//XMLEnumeratedServices() method
//In the sample client development environment, the value returned is "AVAYA#CM150#CSTA-
S#XXAES195"
//Method used : Connect()
//Positive acknowledgement : SocketConnected and StreamConnected event fires
//Negative acknowledgement : CSTAErrorReturned event fires
xmlClient.TServerLinkName = "AVAYA#CM150#CSTA-S#XXAES195";
xmlClient.Connect();
```

3. Disconnect from XML Server.

   Closes the connection to Active XML Server.

   C# code:

```
//Disconnects from XML Server
//Method used : Disconnect()
xmlClient.Disconnect();
```

4. Connect to Media Director through Media Proxy.

   C# code:

```
//mediaClient is ASMediaClient used for all multimedia work items.
//Method used : Connect(mediaTypes,error)
//Parameters :
//mediaTypes(Int32[]) - specifies the media type of the work item it needs to receive from
Media Proxy. It will automatically register the specified media type after it connects to the
Media Director through the Media Proxy Server. Set its value to null to register all Media
Types
//error(GQEErrorArgs) - indicates the error when connecting to the server failed.
bool connToMediaDirector = mediaClient.Connect(null, out errorReturned);
//connToMediaStore = true if  connected successfully to Media Director.
```

5. Disconnect from Media Director.

   Closes the connection with Media Director.

   C# code:

```
//Disconnects from Media Director.
//Method used : Disconnect()
mediaClient.Disconnect();
```

6. Establish a remoting connection with Media Store based on the specified URL.

   C# code:

```
//Connects to the Email Media Store.
//Method used : ConnectToMediaStore(mediaStoreURL,errorText)
//Parameters :
//mediaStoreURL - specifies the URL of the Media Store that it connects to.
//errorText - returns the error text when it fails to connect to the Media Store.
//Get Media Store list

mediaStoreList = mediaClient.GetMediaStoreList();

//For Email Media type(1), connect to the media store using remoting URL


for (int i = 0; i < mediaStoreList.Count; i++)
{
    ems = mediaClient.ConnectToMediaStore(mediaStoreList[i].RemotingURL, out
    error_connectMS);
}
```

Sample Applications in EMC

The following is the list of sample applications that work with EMC and are available for download from Avaya DevConnect portal:

- Agent Login
- Voice Call Handling
- Email Handling
- SMS Handling
- Web Chat Handling

For detailed instructions, see *Chapter 5: Sample Applications*.

Chapter 3: Installation Requirements

To develop a contact center application using EMC Developer, the developers need to install EMC Developer and EMC Desktop on a client computer having Microsoft development environment, such as Visual Basic or Visual C#.

EMC Desktop is required as only the mandatory dlls are provided in the developer and any optional dlls not sourced though the developer install will need to be referenced from the EMC Desktop installation folder. The Desktop installer also installs the Media Proxy service which is required to connect to the Media Director.

For more information, see the following sections in *Installing Avaya Aura® Call Center Elite Multichannel:*

- *Installing and Configuring Desktop Components*
- *Installing Developer Components*

Software requirements

Application development using the **EMC Desktop API** is supported on the following platforms:

- Microsoft Windows 7 SP1:
  - Professional (32-bit and 64-bit)
  - Enterprise (32-bit and 64-bit)
  - Ultimate (32-bit and 64-bit)
- Microsoft Windows 8.0, windows 8.1 (32-bit or 64-bit) – Pro, Enterprise editions.
- Microsoft Windows 10 (32-bit or 64-bit).

Pre-requisites for application to work:

- Microsoft Internet Explorer 11
- Microsoft .Net Framework 3.5 SP1
- Microsoft .Net Framework 4.5.2 or above.
- Microsoft Visual Studio System 2013 or above.
- Avaya Aura® Call Center Elite MultiChannel Developer (described in "Installing Developer Components") should be installed.
- Avaya Aura® Call Center Elite MultiChannel Desktop should be installed.
- Softphones: Avaya one-X Communicator or Avaya one-X Agent should be installed. Otherwise having desk phone will be sufficient.

Chapter 4: Getting Started

EMC Work Flows

There are two primary work flows in EMC

- Voice Work Flow
- Multimedia Work Flow

The difference primarily stems from the fact that while Voice calls first arrive at the Avaya CM, which first does the agent selection for the call before passing the call details to EMC, Multimedia contacts as opposed to that, come first to EMC, which then raises an Agent selection request to CM.

Voice Work Flow

Voice work flow is as follows

- The voice call from a customer will first come to the Communication Manager Elite (CM Elite).
- The CM will then route it to appropriate agent.
- The call UCID / agent station details will be sent to EMC XML server via AES.
- The XML server will then send this to the corresponding agent desktop XML client.
- XML server will also inform Voice Media Store (VMS) about the call.
- The VMS will create a work item for that call.
- Media Client plug-in of Agent desktop will connect to the Voice Media Store using the Media Proxy and retrieve the Work item.

Multimedia Work Flow

Multimedia consists of email, text, web chat or instant messaging.

- A Multimedia request will first come to the corresponding Media Store,
  e.g. Email Media Store (EMS) or Simple Media Messaging Store (SMMS) for WebChat/IM
- The Media Store will create a work item for that.
- Media Store will also make a phantom call (phantom call is a pseudo call generated by CM using phantom stations) request to the CM through Media Director ->XML Server > AES.
- The CM will route to the appropriate agent and inform the selected station back to Media Director.
- The Media director then sends the Media Store work item reference to the corresponding Agent desktop Media Client via Media Proxy.
- The Agent desktop Media Client will then load the work item from corresponding Media Store.

Steps to start building custom Agent Desktop application

The steps to start building custom agent desktop application are as follows:

1. Start Microsoft Visual Studio and from the Visual Studio menu option, select File > New > Project.

2. Add the following dependencies:
   - ASXMLClient.dll
   - CSTASchemas.dll
   - log4net.dll

3. Include following namespaces in your main project file.

```
using AgileSoftware.Developer;
using AgileSoftware.Developer.CSTA;
using AgileSoftware.Developer.XML;
using System.Threading;
using System.Text.RegularExpressions;
using System.Numeric;
using log4net;
using log4net.Config;
using System.Drawing.Imaging;
```

4. The sample code for basic activities like Agent Login and various channel handling can be found in the next chapter describing the Sample Applications.

5. Creating objects
   1. Initialize a new instance of AgileSoftware.Developer.ASXMLClient class.

      C# code:

```
//Create instance of ASXMLClient
ASXMLClient xmlClient;
xmlClient = new ASXMLClient();
```

   2. Subscribe to ASXMLClient events.

      C# code:

```
//ASXMLClient Events
 xmlClient.XMLEnumerateServicesReturned += new
 AgileSoftware.Developer.XML.XMLEnumerateServicesReturnedEventHandler(myClient_XMLEnumerat
 eServicesReturned);
 xmlClient.StreamConnected += new EventHandler(myClient_StreamConnected);
 xmlClient.SocketConnected += new EventHandler(myClient_SocketConnected);
 xmlClient.CSTAMonitorStartResponse += new
 CSTAMonitorStartResponseEventHandler(myClient_CSTAMonitorStartResponse);
xmlClient.CSTAMonitorStopResponse += new
CSTABasicResponseEventHandler(myClient_CSTAMonitorStopResponse);
 xmlClient.CSTAAgentLoggedOn += new
CSTAAgentLoggedOnEventHandler(myClient_CSTAAgentLoggedOn);
xmlClient.CSTAAgentLoggedOff += new
CSTAAgentLoggedOffEventHandler(myClient_CSTAAgentLoggedOff);
 xmlClient.CSTAAgentReady += new CSTAAgentReadyEventHandler(myClient_CSTAAgentReady);
xmlClient.CSTAAgentNotReady += new
CSTAAgentNotReadyEventHandler(myClient_CSTAAgentNotReady);
 xmlClient.CSTAAgentWorkingAfterCall += new
CSTAAgentWorkingAfterCallEventHandler(myClient_CSTAAgentWorkingAfterCall);
 xmlClient.CSTAGetAgentStateResponse += new
CSTAGetAgentStateResponseEventHandler(myClient_CSTAGetAgentStateResponse);
xmlClient.CSTAErrorReturned +=  new
CSTAErrorReturnedEventHandler(myClient_CSTAErrorReturned);
```

6. Recovery

   Recovery for XML Server Close.

   Try to handle the following event like

```
asxmlClient.ServerClosed += new EventHandler(asxmlClient_ServerClosed);
```

   On that event try to reconnect the XML Server

```
asxmlClient.Connect();
```

7. Debugging

   a. Implement logging for the application

   You can use `log4net` control for logging data in file. This is open source library available from Apache Foundation. This is also included with the Sample Applications.

   There are a three parts to `log4net`. There is the configuration, the setup, and the logging code.

   i. Configure `log4net`

   The standard way to set up a `log4net` logger is to utilize the *app.config* file in a desktop application. To use *app.config* you need to add a configuration section to this file

```
<configuration>
  <configSections>
    <section name="log4net"
    type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
  </configSections>
<configuration>
```

   In the *app.config* file add "<log4net></log4net>" under the <configSections/> element. Then add the following code in the <log4net /> element

```
<log4net>
<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
      <file value="AgentLoginApplication.log" />
      <appendToFile value="true" />
      <maximumFileSize value="1000KB" />
      <maxSizeRollBackups value="2" />
      <layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%date %-5level [%thread] - %message%newline" />
      </layout>
    </appender>
    <root>
      <level value="DEBUG" />
      <appender-ref ref="RollingFile" />
    </root>
</log4net>
```

ii. Setup log4net

Now that configuration is ready, we need to get log4net to read in this configuration. There are several ways to do this, but the main one is to insert the following line in your application startup code, in a method which executes this line before any Logging:

```
XmlConfigurator.Configure();

//Log File is generated that stores log entries of the application. It is stored in
the same directory with the name: AgentLoginApplication.log in this example
```

Now log4net is primed and ready to be used.

iii. Logging using log4net

In the class that you want to start logging information, put the following code.

```
using log4net;
public static readonly ILog log = LogManager.GetLogger(typeof(AgentLoginForm));
```

Then logs can be added anywhere in this class as needed, example:

```
public AgentLoginForm()
{
      //Initialize a new instance...
      InitializeComponent();
      buildEventHandlers();
      // etc ...

      log.Info("Initialized AgentLoginApplication");
}
```

b. To handle XML Client error situations, handle XMLClient_XMLServerError and xmlClient.CSTAErrorReturned and add errors to the logging file.

```
void myClient_CSTAErrorReturned(object sender, CSTAErrorReturnedEventArgs arg)
{
       rtbStatus.Text = "Error Occurred: " + " Category: " + arg.Error.Category + "
       Description: " + arg.Error.ValueDescription;
      log.Error("CSTA Error occurred" + arg.Error.ValueDescription);
}
```

Also when doing xml client connect or disconnect

```
err = xmlClient.Connect();
```

We get enASXMLClientError in return parameter; log these values in logs as below

```
//For connect
if (connectionState == Connection.Disconnect)
{
        xmlClient.TServerLinkName = cmbServerLink.Text.Trim();
        err = xmlClient.Connect();
        rtbStatus.Text = err.ToString();
       log.Fatal("Error occured while connection xml server : " + err.ToString());
}
```

Similarly handle media client error situations.

For example, when we connect to Media director we get status in Boolean return value. We can log status message accordingly as

```
//mediaClient is ASMediaClient used for all multimedia work items.
//Method used : Connect(mediaTypes,error)
//Parameters :
//mediaTypes(Int32[]) – specifies the media type of the work item it needs to receive
from Media Proxy. It will automatically register the specified media type after it
connects to the Media Director through the Media Proxy Server. Set its value to null to
register all Media Types
//error(GQEErrorArgs) – indicates the error when connecting to the server failed.

//RegisterMediaType: Registers a media type for which users need to receive work items.
// Parameters:
//    mediaType: Specifies the media type for which the work items will be received. Set
its value to 0 to register all the media types.
//    invokeRef: Returns a unqiue reference of this request in GUID format.
if (mediaClient.Connect(null, out errorReturned))
{
      log.Info("Connected to Media Director");
      rtbStatus.Text = "Connected to Media Director successfully";
      mediaClient.RegisterMediaType(0, out invoke_ref);
      btnMDConnect.Text = "Disconnect";
      mediaDir_state = MediaDirectorState.Connected;
      gbWebChat.Enabled = true;
}
else
{
       log.Error("Connection to Media Director failed " +
       errorReturned.ErrorDescription);
       rtbStatus.Text = "Connection to Media Director failed" +
       errorReturned.ErrorDescription;
}
```

c. Check corresponding EMC servers are up and running successfully.

8. Cleanup

C# code:

```
//ASXMLClient Events
 xmlClient.XMLEnumerateServicesReturned -= new
 AgileSoftware.Developer.XML.XMLEnumerateServicesReturnedEventHandler(myClient_XMLEnumerateSer
 vicesReturned);
 xmlClient.StreamConnected -= new EventHandler(myClient_StreamConnected);
 xmlClient.SocketConnected -= new EventHandler(myClient_SocketConnected);
 xmlClient.CSTAMonitorStartResponse -= new
 CSTAMonitorStartResponseEventHandler(myClient_CSTAMonitorStartResponse);
xmlClient.CSTAMonitorStopResponse -= new
CSTABasicResponseEventHandler(myClient_CSTAMonitorStopResponse);
 xmlClient.CSTAAgentLoggedOn -= new CSTAAgentLoggedOnEventHandler(myClient_CSTAAgentLoggedOn);
xmlClient.CSTAAgentLoggedOff -= new
CSTAAgentLoggedOffEventHandler(myClient_CSTAAgentLoggedOff);
 xmlClient.CSTAAgentReady -= new CSTAAgentReadyEventHandler(myClient_CSTAAgentReady);
 xmlClient.CSTAAgentNotReady -= new CSTAAgentNotReadyEventHandler(myClient_CSTAAgentNotReady);
 xmlClient.CSTAAgentWorkingAfterCall -= new
CSTAAgentWorkingAfterCallEventHandler(myClient_CSTAAgentWorkingAfterCall);
 xmlClient.CSTAGetAgentStateResponse -= new
CSTAGetAgentStateResponseEventHandler(myClient_CSTAGetAgentStateResponse);
 xmlClient.CSTAErrorReturned -= new CSTAErrorReturnedEventHandler(myClient_CSTAErrorReturned);
```

Chapter 5: Sample Applications

Following is the list of sample applications that are shipped along with EMC installer:

- Agent Login
- Voice Call Handling
- Email Handling
- SMS Handling
- Web Chat Handling

Details of these follow below.

5.1 Agent Login

The Agent Login Sample shows how the agent logs into the Elite MultiChannel (EMC). To login to EMC, the agent must connect to the XML Server. Using valid XML Server IP address and Port number, the agent can browse through available TServer Links. The links returned can be secured or unsecured links. The user needs to choose one of the available links and connect to the XML Server. Once connected to the XML Server, the agent can logon to a station using a valid station ID and start monitoring the events taking place on that particular station. After getting logged on to a particular station, the agent can handle the work items only when the agent logs on to the Communication Manager (CM) using a softphone or a deskphone. The agent logs onto the CM using valid agent ID and password. Once the agent has logged on to the CM, and is in the available state, the current work item is assigned to the agent. Once logged in, the agent may also change between states using the method shown in this sample application. The sample application maintains a log file named AgentLoginApplication.log that maintains all logs related to Agent Login and is stored where the application is running.

Steps

1. Start Microsoft Visual Studio.

2. From the Visual Studio menu option, select File > New > Project.

3. From the Visual Studio menu option, select Project > Add New Item > Application Configuration File.

4. Open the file and paste the following code between the <configuration></configuration> tags.

```
<configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"
    />
</configSections>
<log4net>
<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
<file value="AgentLoginApplication.log" />
<appendToFile value="true" />
<maximumFileSize value="1000KB" />
<maxSizeRollBackups value="2" />
<layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%date %-5level [%thread] - %message%newline" />
</layout>
</appender>
<root>
<level value="DEBUG" />
<appender-ref ref="RollingFile" />
</root>
</log4net>

Details on parameters above:

maxSizeRollBackups indicates property which Gets or sets the maximum number of backup files
that are kept before the oldest is erased.
```

The AgentLoginForm.cs file will implement the functionality for Agent Login Application.

5. Add the following namespaces:

```
using AgileSoftware.Developer;
using AgileSoftware.Developer.CSTA;
using AgileSoftware.Developer.XML;
using System.Threading;
using System.Text.RegularExpressions;
using System.Numeric;
using log4net;
using log4net.Config;
using System.Drawing.Imaging;
```

6. Initialize a new instance of AgileSoftware.Developer.ASXMLClient class.

C# code:

```
//Create instance of ASXMLClient
ASXMLClient xmlClient;
xmlClient = new ASXMLClient();
```

7. Subscribe to ASXMLClient events.

C# code:

```
//ASXMLClient Events
 xmlClient.XMLEnumerateServicesReturned += new
 AgileSoftware.Developer.XML.XMLEnumerateServicesReturnedEventHandler(myClient_XMLEnumerateSer
 vicesReturned);
 xmlClient.StreamConnected += new EventHandler(myClient_StreamConnected);
 xmlClient.SocketConnected += new EventHandler(myClient_SocketConnected);
 xmlClient.CSTAMonitorStartResponse += new
 CSTAMonitorStartResponseEventHandler(myClient_CSTAMonitorStartResponse);
 xmlClient.CSTAMonitorStopResponse += new
 CSTABasicResponseEventHandler(myClient_CSTAMonitorStopResponse);
 xmlClient.CSTAAgentLoggedOn += new CSTAAgentLoggedOnEventHandler(myClient_CSTAAgentLoggedOn);
 xmlClient.CSTAAgentLoggedOff += new
 CSTAAgentLoggedOffEventHandler(myClient_CSTAAgentLoggedOff);
 xmlClient.CSTAAgentReady += new CSTAAgentReadyEventHandler(myClient_CSTAAgentReady);
 xmlClient.CSTAAgentNotReady += new CSTAAgentNotReadyEventHandler(myClient_CSTAAgentNotReady);
 xmlClient.CSTAAgentWorkingAfterCall += new
 CSTAAgentWorkingAfterCallEventHandler(myClient_CSTAAgentWorkingAfterCall);
 xmlClient.CSTAGetAgentStateResponse += new
 CSTAGetAgentStateResponseEventHandler(myClient_CSTAGetAgentStateResponse);
 xmlClient.CSTAErrorReturned += new CSTAErrorReturnedEventHandler(myClient_CSTAErrorReturned);
```

8. Browse for available Server Telephony Links.

   Connects to the XML Server name service, using valid XML Server IP address and Port number, to retrieve the available TServer link names and associated socket connection information.

   C# code:

```
//Connects to the XML Server with Server IP address : “135.156.23.22” and port no. “29096” to
retrive available links
//Method used : XMLEnumerateServices()
//Positive acknowledgement : XMLEnumeratedServicesReturned event fires
//Negative acknowledgement : CSTAErrorReturned event fires
xmlClient.ServerIP = “135.156.23.22”;
xmlClient.ServerPort = 29096;
xmlClient.XMLEnumerateServices();
```

9. Connect to XML Server.

   Connects to the Active XML Server using naming service. It establishes the real TServer connection under the specified TServer link name.

   C# code:

```
//Connects to the XML Server using the specified TServer Link name returned by the
//XMLEnumeratedServices() method
//In the sample client development environment, the value returned is “AVAYA#CM150#CSTA-
S#XXAES195”
//Method used : Connect()
//Positive acknowledgement : SocketConnected and StreamConnected event fires
//Negative acknowledgement : CSTAErrorReturned event fires
xmlClient.TServerLinkName = “AVAYA#CM150#CSTA-S#XXAES195”;
xmlClient.Connect();
```

10. Disconnect from XML Server.

    Closes the connection to the Active XML Server.

    C# code:

```
//Disconnects from XML Server
//Method used : Disconnect()
xmlClient.Disconnect();
```

11. Start Monitor.

Starts monitor a device object. It takes as input a valid station ID.

C# code:

```
//Start to monitor a voice station 6511
//Method used : CSTAMonitorStart(DeviceObject,MonitorType,MonitorFilter)
//Parameters :
//DeviceObject(CSTADeviceID) – the monitor will be placed on.
//MonitorType(enMonitorType) – indicates the monitor tyoe when starting a monitor.
//MonitorFilter(CSTAMonitorFilter) – indicates which event will be filtered out when starting
//a monitor.
//Positive acknowledgement : CSTAMonitorStartResponse event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Create a CSTADeviceID object for voice station 6511
CSTADeviceID _thisDevice = new CSTADeviceID("6511", enDeviceIDType.deviceNumber);
//Use the CSTAMonitorStart method to start monitor this device.
xmlClient.CSTAMonitorStart(_thisDevice, enMonitorType.device, null);
```

12. Stop Monitor Station.

Used to cancel a previously initiated Monitor Start service.

C# code:

```
//Stops monitoring a device.
//Method used : CSTAMonitorStop(CSTAMonitor)
//Parameters :
//CSTAMonitorObject(CSTAMonitor) – indicates one of the CSTAMonitor object listed in the
//CSTAMonitorList
//Positive acknowledgement : CSTAMonitorStopResponse event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Get CSTAMonitorList
CSTAMonitorList monitorList = xmlClient.CSTAMonitorList;
//Stop monitoring the first CSTAMonitor objectxmlClient.CSTAMonitorStop(monitorList[0]);
```

13. Agent State Change.

Request a new agent state at the specified device.
The agent can be in one of the following states:

- agentLoggedOn
- agentLoggedOff
- agentReady
- agentNotReady
- agentWorkingAfterCall

C# code:

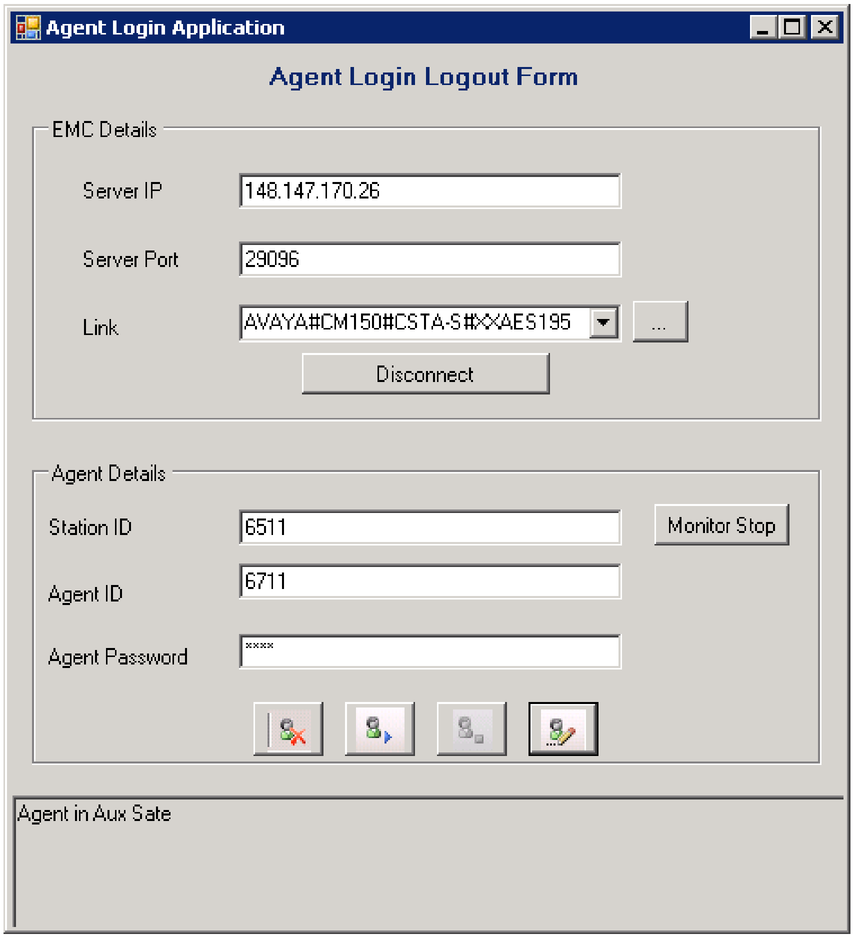
```
//Sets agent in requested state.
//Method used : CSTASetAgentState(DeviceID,AgentState,AgentID,AgentPwd,CSTADeviceID,
//PrivateDateSetAgentState)
//Parameters :
//DeviceID(CSTADeviceID) -  specifies the DeviceID for the agent
//AgentState(enReqAgentState) – specifies requested agent state
//AgentId(String) – specifies the agent identifier
//AgentPwd(String) – specifies the agent password
//ACDgroup(CSTADeviceID) – specifies the agent group
//PrivateData(PrivateDataSetAgentState) – specifies private data associated with this method.

//Create a CSTADeviceID object for voice station 6511
CSTADeviceID _thisDevice = new CSTADeviceID("6511", enDeviceIDType.deviceNumber);
//Set agent id
String agentID = “6711”;
//Set agent password
String agentPwd = “1234”;

//Following method logs on agent with agentID 6711 and password : 1234 onto voice station
6511.
xmlClient.CSTASetAgentState(_thisDevice, enReqAgentState.loggedOn, agentID, agentPwd, null,
null);
//Positive acknowledgement : CSTAAgentLoggedOn event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Following method logs off agent with agentID 6711 and password : 1234 from voice station
//6511.
xmlClient.CSTASetAgentState(_thisDevice, enReqAgentState.loggedOff, agentID,agentPwd, null,
null);
//Positive acknowledgement : CSTAAgentLoggedOff event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Following method sets agent state as Available for agent with agentID 6711 and password :
1234 on voice station 6511.
xmlClient.CSTASetAgentState(_thisDevice, enReqAgentState.ready, agentID, agentPwd, null,
null);
//Positive acknowledgement : CSTAAgentReady event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Following method sets agent state as Auxiliary for agent with agentID 6711 and password :
//1234 on voice station 6511.
xmlClient.CSTASetAgentState(_thisDevice, enReqAgentState.notReady, agentID, agentPwd, null,
null);
//Positive acknowledgement : CSTAAgentNotReady event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Following method sets agent state as Working After Call for agent with agentID 6711 and
//password : 1234 on voice station 6511.
xmlClient.CSTASetAgentState(_thisDevice, enReqAgentState.workingAfterCall, agentID, agentPwd,
null, null);
//Positive acknowledgement : CSTAAgentWorkingAfterCall event fires
//Negative acknowledgement : CSTAErrorReturned event fires
```

14. Clear events subscribed.

C# code:

```
//ASXMLClient Events
xmlClient.XMLEnumerateServicesReturned -= new
AgileSoftware.Developer.XML.XMLEnumerateServicesReturnedEventHandler(myClient_XMLEnumerateSer
vicesReturned);
xmlClient.StreamConnected -= new EventHandler(myClient_StreamConnected);
xmlClient.SocketConnected -= new EventHandler(myClient_SocketConnected);
xmlClient.CSTAMonitorStartResponse -= new
CSTAMonitorStartResponseEventHandler(myClient_CSTAMonitorStartResponse);
xmlClient.CSTAMonitorStopResponse -= new
CSTABasicResponseEventHandler(myClient_CSTAMonitorStopResponse);
xmlClient.CSTAAgentLoggedOn -= new CSTAAgentLoggedOnEventHandler(myClient_CSTAAgentLoggedOn);
xmlClient.CSTAAgentLoggedOff -= new
CSTAAgentLoggedOffEventHandler(myClient_CSTAAgentLoggedOff);
xmlClient.CSTAAgentReady -= new CSTAAgentReadyEventHandler(myClient_CSTAAgentReady);
xmlClient.CSTAAgentNotReady -= new CSTAAgentNotReadyEventHandler(myClient_CSTAAgentNotReady);
xmlClient.CSTAAgentWorkingAfterCall -= new
CSTAAgentWorkingAfterCallEventHandler(myClient_CSTAAgentWorkingAfterCall);
xmlClient.CSTAGetAgentStateResponse -= new
CSTAGetAgentStateResponseEventHandler(myClient_CSTAGetAgentStateResponse);
xmlClient.CSTAErrorReturned -= new CSTAErrorReturnedEventHandler(myClient_CSTAErrorReturned);
```

Screenshot


Agent Login Application

Agent Login Logout Form

EMC Details

Server IP: 148.147.170.26

Server Port: 29096

Link: AVAYA#CM150#CSTA-S#XXAES195

Agent Details

Station ID: 6511

Agent ID: 6711

Agent Password: ****

Agent in Aux Sate

Figure1. UI for Agent Login

Steps to run Agent Login Sample UI

1. Enter valid XML Server IP address and XML Server Port number. Click on the Browse Button to browse for available TServer Links.
   Expected Result:
   Available TServer Links are displayed in Combo Box.

2. Select one of the available TServer Links returned. Click on the Connect Button to Connect to the Active XML Server.
   Expected Result:
   "Stream Connected" is displayed in Status Bar.
   Agent Details Group Box enabled.

3. Enter a valid Station ID. Click on the Monitor Start Button to start monitoring events at the specified station.
   Expected Result:
   "Monitor Started" is displayed in Status Bar.
   If agent is already logged in, agent is set to current state. Else Agent ID and Agent Password Textbox are enabled.

4. Enter valid Agent ID and Password. Click Agent Login Button to logon Agent.
   Expected Result:
   Agent is logged on to the system and set to the current Agent State.
   The current Agent State is displayed in Status Bar.
   Agent State Change Buttons are enabled.

5. Click Available Button to set agent in Available (Ready) state.
   Expected Result:
   "Agent in Available State" is displayed in Status Bar.

6. Click Auxiliary Button to set agent in Auxiliary (Not Ready) state.
   Expected Result:
   "Agent in Aux State" is displayed in Status Bar.

7. Click After Call Work Button to set agent in ACW (Working After Call) state.
   Expected Result:
   "Agent in ACW State" is displayed in Status Bar.

8. Click Agent Logout Button to logout Agent.
   Expected Result:
   “Agent Logged Off” is displayed in Status Bar.
   Agent State Change Buttons are disabled.

9. Click on the Monitor Stop Button to stop monitoring events at the station.
   Expected Result:
   “Monitor Stopped” is displayed in Status Bar.
   Agent ID and Password Textbox are disabled.
   Agent State Change Buttons are disabled.

10. Click on the Disconnect Button to close connection to the Active XML Server.
    Expected Result:
    Agent Login Application Form is closed.

5.2 Voice Call Handling

The Voice Call Handling Sample shows how the agent will handle voice calls from the customer. To manage voice calls, the agent needs to login to the Communication Manager (CM) and be in the available state. Once the agent is logged in and in the available state, the incoming voice call on the CM is assigned to the agent. When a voice call is assigned to the agent, the agent can answer the call, end the call, place the call on hold, unhold the held call, transfer the call to another agent and take the call in conference with another agent. The agent may also make a call from the voice station. The sample application maintains a log file named CallHandlingApplication.log which stores all the logs related to the Voice Call Handling and is stored in the installation Directory.

Steps

1. Start Microsoft Visual Studio.

2. From the Visual Studio menu option, select File > New > Project.

3. From the Visual Studio menu option, select Project > Add New Item > Application Configuration File.

4. Open the file and paste the following code between the <configuration></configuration> tags.

```
<configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"
/>
</configSections>
<log4net>
<appender name="RollingFile"type="log4net.Appender.RollingFileAppender">
<file value="CallHandlingApplication.log" />
<appendToFile value="true" />
<maximumFileSize value="1000KB" />
<maxSizeRollBackups value="2" />
<layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%date %-5level [%thread] - %message%newline" />
</layout>
</appender>
<root>
<level value="DEBUG" />
<appender-ref ref="RollingFile" />
</root>
</log4net>

Details on parameters above:

maxSizeRollBackups indicates property which Gets or sets the maximum number of backup files
that are kept before the oldest is erased.
```

The VoiceCallHandlingForm.cs file will implement the functionality for handling voice calls.

5. Add the following namespaces:

```
using AgileSoftware.Developer;
using AgileSoftware.Developer.CSTA;
using AgileSoftware.Developer.XML;
using AgileSoftware.Developer.Station;
using AgileSoftware.Developer.Device;
using System.Threading;
using System.Text.RegularExpressions;
using System.Numeric;
using log4net;
using log4net.Config;
using System.Drawing.Imaging;
```

6. Initialize a new instance of AgileSoftware.Developer.ASXMLClient class. (Refer Step 7 of Agent Login)

   Initialize a new instance of AgileSoftware.Developer.ASXMLStation class.

   C# code:

```
//Create instance of ASXMLClient
ASXMLStation xmlStation;
xmlStation = new ASXMLStation();
```

7. Subscribe to the ASXMLClient and ASXMLStation events.
   (Refer Step 8 of Agent Login)

   In addition, subscribe to the following CSTA events:

   C# code:

```
//ASXMLClientevents
 xmlClient.CSTADelivered += new CSTADeliveredEventHandler(xmlClient_CSTADelivered);
 xmlClient.CSTAEstablished += new CSTAEstablishedEventHandler(xmlClient_CSTAEstablished);
 xmlClient.CSTAAnswerCallResponse += new
CSTABasicResponseEventHandler(xmlClient_CSTAAnswerCallResponse);
 xmlClient.CSTAMakeCallResponse += new
CSTAMakeCallResponseEventHandler(xmlClient_CSTAMakeCallResponse);
 xmlClient.CSTAConnectionCleared += new
CSTAConnectionClearedEventHandler(xmlClient_CSTAConnectionCleared);
 xmlClient.CSTAHeld += new CSTAHeldEventHandler(xmlClient_CSTAHeld);
 xmlClient.CSTAHoldCallResponse += new
CSTABasicResponseEventHandler(xmlClient_CSTAHoldCallResponse);
 xmlClient.CSTARetrieveCallResponse +=    new
CSTABasicResponseEventHandler(xmlClient_CSTARetrieveCallResponse);
 xmlClient.CSTARetrieved += new CSTARetrievedEventHandler(xmlClient_CSTARetrieved);
 xmlClient.CSTATransferCallResponse += new
CSTATransferCallResponseEventHandler(xmlClient_CSTATransferCallResponse);
 xmlClient.CSTATransfered += new CSTATransferedEventHandler(xmlClient_CSTATransfered);
 xmlClient.CSTAConferenceCallResponse += new
CSTAConferenceCallResponseEventHandler(xmlClient_CSTAConferenceCallResponse);
 xmlClient.CSTAConferenced += new CSTAConferencedEventHandler(xmlClient_CSTAConferenced);

//ASXMLStation events
 xmlStation.MonitorStarted += new EventHandler(xmlStation_MonitorStarted);
xmlStation.MonitorStopped += new EventHandler(xmlStation_MonitorStopped);
```

   The ConnectToXMLServer.cs file is used to connect to the XML Server.

8. Browse for available Server Telephony Links:
   (Refer Step 9 of Agent Login)

9. Connect to XML Server:
   (Refer Step 10 of Agent Login)

10. Disconnect from XML Server:
    (Refer Step 10 of Agent Login)

    The Monitor.cs file is used to start and stop monitor.

11. Start Monitor:

    (Refer Step 12 of Agent Login)

    OR

    C# code:

```
//Start to monitor a voice station 6511
//Set the following properties to place a monitor.
xmlStation.StationDN = "6511";
xmlStation.CSTASource = xmlClient;
```

12. Stop Monitor Station:

    (Refer Step 13 of Agent Login)

    The Agent.cs file is used for agent state change functionalities.

13. Agent State Change:

    (Refer Step 14 of Agent Login)

    The CallHandling_Class.cs file is used for handling voice calls.

14. Handling Voice Calls:

    The agent can:

    - Make a Call
    - Answer a Call
    - End a Call
    - Hold a Call
    - Unhold a Call
    - Transfer a Call
    - Conference a Call

C# code :

When a Call is being presented to a voice station.

```
//Positive acknowledgement : CSTADelivered event fires
//Negative acknowledgement : CSTAErrorReturned event fires
```

Make a Call: Places a new call on the voice station.

```
//Following method dials a call to phone number “1425”
//Method used : CallDial(calledNumber,userData)
//Parameters :
//calledNumber(String) : specifies the phone number which will be called
//userData(String) : specifies the user to user information (UUI) which is attached with the
call
//Positive acknowledgement : CSTAMakeCallResponse event fires
//Negative acknowledgement : CSTAErrorReturned event fires
string CallToNumber = “1425”;
xmlStation.CallDial(CallToNumber, "");
```

Answer a Call: Answers the first altering call found on the voice station.

```
//Following method answers the first altering call on the voice station
//Method used : CallAnswer()
//Positive acknowledgement : CSTAAnswerCallResponse event fires
//Negative acknowledgement : CSTAErrorReturned event fires
xmlStation.CallAnswer();
```

End a Call: Releases all connections associated with the call found on the voice station.

```
//Following method releases all connections associated with the call on voice station
//Method used : CallRelease()
//Positive acknowledgement : CSTAConnectionCleared event fires
//Negative acknowledgement : CSTAErrorReturned event fires
xmlStation.CallRelease();
```

Hold a Call: Places an active call connection on hold.

```
//Following method holds the specified call on the voice station
//Method used : CallHold(callToBeHeld)
//Parameters :
//callToBeHeld(StationCall) : specifies the call to be held
//Positive acknowledgement : CSTAHoldCallResponse or CSTAHeld event fires
//Negative acknowledgement : CSTAErrorReturned event fires
StationCall objStationCall = xmlStation.GetCallByCallID(make_call_id);
//make_call_id : Returns the call ID and is retrieved from args of CSTAMakeCallResponse event
xmlStation.CallHold(objStationCall);
```

Unhold a Call: Retrieves the held call found on the voice station to make it as the active call.

```
//Following method retrieves the held call to make it an active call
//Method used : CallUnhold()
//Positive acknowledgement : CSTARetrieveCallResponse or CSTARetrieved event fires
//Negative acknowledgement : CSTAErrorReturned event fires
xmlStation.CallUnhold();
```

Transfer a Call: Performs the transfer call operation on the current active call of the voice station.

There are two types of transfer operation: blind transfer and consult transfer.

For blind transfer, the component automatically completes the transfer operation.

For consult transfer, the component does not automatically complete the transfer operation. The application needs to complete the pending transfer.

```
//Method used : CallTransfer(transferType,transferToNumber,userData)
//Parameters :
//transferType(enTransferType) : specifies the type of transfer operation to be undertaken
//transferToNumber(String) : specifies the destination number the call will be transfered to
//userData(String) : specifies user to user information (UUI) attached with the transfering
//call
//Positive acknowledgement : CSTATransferCallResponse or CSTATransfered event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Following method blind transfers the call to phone number “6513”
string transfer_station_id = “6513”;
xmlStation.CallTransfer(enTransferType.BlindTransfer, transfer_station_id, "");

//Following method consult transfers the call to phone number “6513”
//Start Consult
string transfer_station_id = “6513”;
xmlStation.CallTransfer(enTransferType.StartConsult, transfer_station_id, "");
//Complete Consult
xmlStation.CallTransfer(enTransferType.CompleteConsult, "", "");
```

Conference a Call: Performs the conference call operation on the current active call of the voice station.

There are two types of conference operation: blind conference and consult conference.

For blind conference, the component automatically completes the conference operation.

For consult conference, the component does not automatically complete the conference operation. The application needs to complete the pending conference.

```
//Method used : CallConference(conferenceType,conferenceWithNumber,userData)
//Parameters :
//conferenceType(enConferenceType) : specifies the type of conference operation to be
//undertaken
//conferenceWithNumber(String) : specifies the destination number the call will be
//conferenced to
//userData(String) : specifies user to user information (UUI) attached with the conferencing
//call
//Positive acknowledgement : CSTAConferenceCallResponse or CSTAConferenced event fires
//Negative acknowledgement : CSTAErrorReturned event fires

//Following method blind conferences the call with phone number “6513”
String conference_station_id = “6513”;
xmlStation.CallConference(enConferenceType.BlindConference, conference_station_id, "");

//Following method consult conferences the call with phone number “6513”
//Start Consult
String conference_station_id = “6513”;
xmlStation.CallConference(enConferenceType.StartConsult, conference_station_id, "");
//Complete Consult
xmlStation.CallConference(enConferenceType.CompleteConsult, "", "");
```

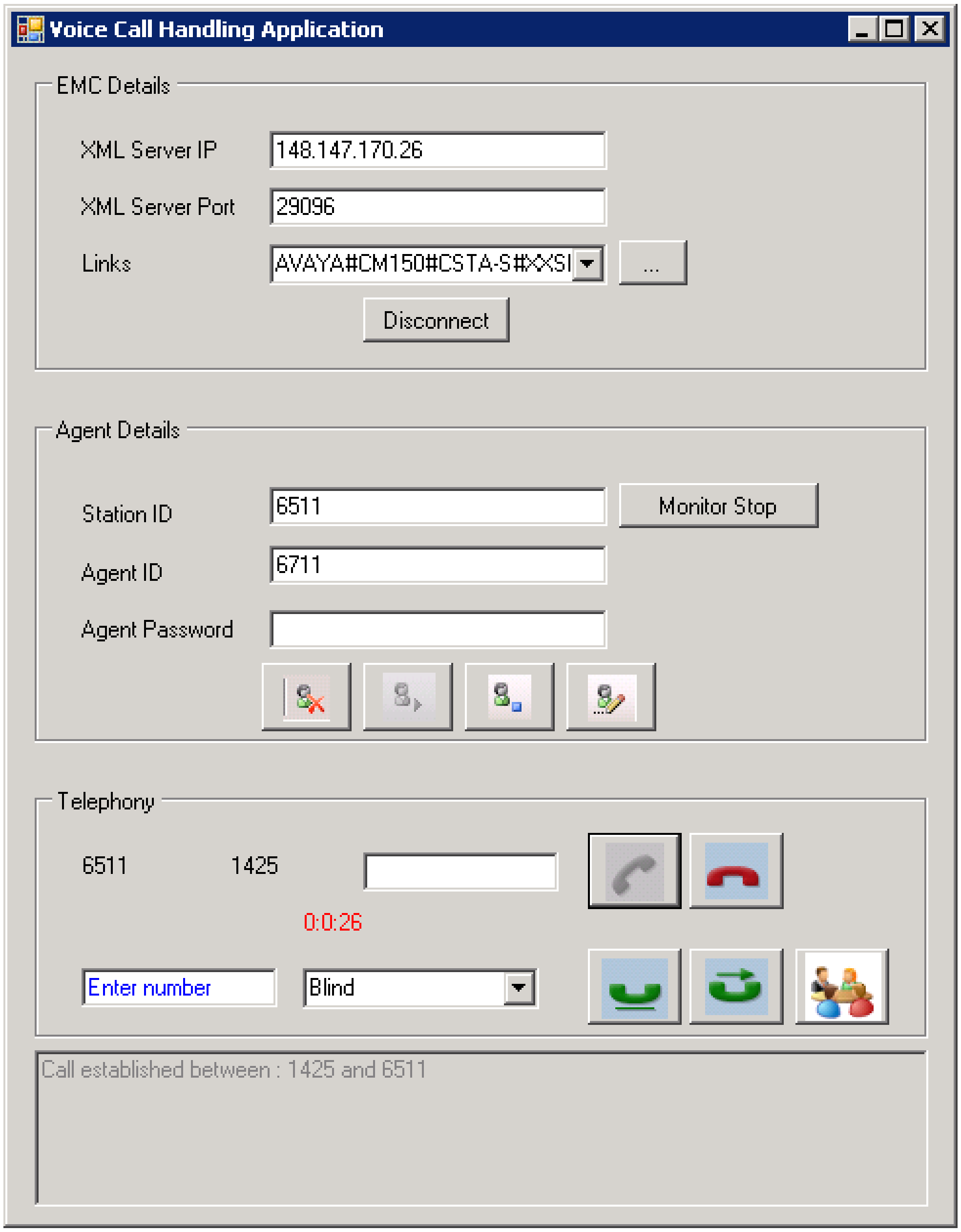
15. Clear events subscribed:
    (Refer Step 15 of Agent Login)

    In addition, clear the following CSTA events:

    C# code:

```
//ASXMLClient Events
 xmlClient.CSTADelivered -= new CSTADeliveredEventHandler(xmlClient_CSTADelivered);
 xmlClient.CSTAEstablished -= new CSTAEstablishedEventHandler(xmlClient_CSTAEstablished);
 xmlClient.CSTAAnswerCallResponse -= new
CSTABasicResponseEventHandler(xmlClient_CSTAAnswerCallResponse);
 xmlClient.CSTAMakeCallResponse -= new
CSTAMakeCallResponseEventHandler(xmlClient_CSTAMakeCallResponse);
 xmlClient.CSTAConnectionCleared -= new
CSTAConnectionClearedEventHandler(xmlClient_CSTAConnectionCleared);
 xmlClient.CSTAHeld -= new CSTAHeldEventHandler(xmlClient_CSTAHeld);
 xmlClient.CSTAHoldCallResponse -= new
CSTABasicResponseEventHandler(xmlClient_CSTAHoldCallResponse);
 xmlClient.CSTARetrieveCallResponse -= new
CSTABasicResponseEventHandler(xmlClient_CSTARetrieveCallResponse);
 xmlClient.CSTARetrieved -= new CSTARetrievedEventHandler(xmlClient_CSTARetrieved);
 xmlClient.CSTATransferCallResponse -= new
CSTATransferCallResponseEventHandler(xmlClient_CSTATransferCallResponse);
 xmlClient.CSTATransfered -= new CSTATransferedEventHandler(xmlClient_CSTATransfered);
 xmlClient.CSTAConferenceCallResponse -= new
CSTAConferenceCallResponseEventHandler(xmlClient_CSTAConferenceCallResponse);
 xmlClient.CSTAConferenced -= new CSTAConferencedEventHandler(xmlClient_CSTAConferenced);

//ASXMLStation events
 xmlStation.MonitorStarted -= new EventHandler(xmlStation_MonitorStarted);
 xmlStation.MonitorStopped -= new EventHandler(xmlStation_MonitorStopped);
```

Screenshot



EMC Details

XML Server IP: 148.147.170.26

XML Server Port: 29096

Links: AVAYA#CM150#CSTA-S#XXSI

Agent Details

Station ID: 6511

Agent ID: 6711

Agent Password:

Telephony

6511 1425

0:0:26

Enter number

Blind

Call established between : 1425 and 6511

Figure2. UI for Voice Call Handling

Steps to run Voice Call Handling Sample UI

1. Enter valid XML Server IP address and XML Server Port number. Click on the Browse Button to browse for available TServer Links.
   Expected Result:
   Available TServer Links are displayed in Combo Box.

2. Select one of the available TServer Links returned. Click on the Connect Button to connect to the Active XML Server.
   Expected Result:
   "Stream Connected" is displayed in Status Bar.
   Agent Details Group Box enabled.

3. Enter a valid Station ID. Click on the Monitor Start Button to start monitoring events at the station.
   Expected Result:
   "Monitor Started" is displayed in Status Bar.
   If agent is already logged in, agent is set to current state. Else Agent ID and Agent Password Textbox are enabled.
   Telephony Group Box is enabled.

4. Enter valid Agent ID and Password. Click Agent Login Button to logon Agent.
   Expected Result:
   Agent is logged on to the system and set to the current Agent State.
   The current Agent State is displayed in Status Bar.
   Agent State Change Buttons are enabled.

5. Click Available Button to set agent in Available (Ready) state.
   Expected Result:
   "Agent in Available State" is displayed in Status Bar.

6. Click Auxiliary Button to set agent in Auxiliary (Not Ready) state.
   Expected Result:
   "Agent in Aux State" is displayed in Status Bar.

7. Click After Call Work Button to set agent in ACW (Working After Call) state.
   Expected Result:
   "Agent in ACW State" is displayed in Status Bar.

8. Click Agent Logout Button to logout Agent.
   Expected Result:
   "Agent Logged Off" is displayed in Status Bar.
   Agent State Change Buttons are disabled.

9. To make a call to a specified station, enter a valid Station ID in given Textbox. The Place Call Button will be enabled. Click on the Place Call Button to dial a call to specified station.
   Expected Result:
   Call details like callID, callFrom and callTo are specified in Status Bar.
   Hang Up and Hold Buttons are enabled.
   Call timer started.

10. Call Delivered to a station.
    Expected Result:
    Answer Call and Ignore Buttons are enabled.
    Call details like callFrom and callTo are specified in Status Bar.

11. To answer an incoming call, Click on the Answer Button.
    Expected Result:
    Call details are displayed in Status Bar.
    Hang Up, Hold, Transfer and Conference Buttons will be enabled.
    Call timer is started.

12. To end a call, Click on the Hang Up Button.
    Expected Result:
    "Call cleared" is displayed in Status Bar.
    All call related buttons will be disabled.
    Call timer cleared.

13. To place an active call on Hold, Click on the Hold Button.
    Expected Result:
    "Call held" is displayed in Status Bar.
    Unhold Button is enabled.
    Answer Call, Hold, Transfer, Conference Button are disabled.
    Hold timer is started.

14. To unhold a held call to make it an active call, Click on the Unhold Button.
    Expected Result:
    "Call Retrieved" is displayed in Status Bar.
    Hold, Hang Up, Transfer and Conference Buttons are enabled.
    Hold timer cleared.

15. To blind transfer a call to a specified station, enter a valid station ID in the given textbox and choose Blind Transfer option. Click on the Transfer Button.
    Expected Result:
    "Call transferred" is displayed in Status Bar.
    All call related buttons will be disabled.

16. To consult transfer a call to a specified station, enter a valid station ID in the given textbox and choose Consult Transfer option. Click on transfer Button.
    Expected Result:
    Active call is put on hold until it is answered by the destination station.
    When the call is answered at the specified destination, you need to click on the Complete Transfer Button to complete the transfer operation.
    "Call transferred" is displayed in the Status Bar.

17. To blind conference a call to a specified station, enter a valid station ID in the given textbox and choose Blind Conference option. Click on the Conference Button.
    Expected Result:
    "Call conferenced" is displayed in Status Bar.

18. To consult conference a call to a specified station, enter a valid station ID in the given textbox and choose Consult Conference option. Click on Conference Button.
    Expected Result:
    Active call is put on hold until it is answered by the destination station.
    When the call is answered at the specified destination, you need to click on the Complete Conference Button to complete the conference operation.
    "Call Conferenced" is displayed in Status Bar.

19. Click on the Monitor Stop Button to stop monitoring events at the station.
    Expected Result:
    "Monitor Stopped" is displayed in Status Bar.
    Agent ID and Password Textbox are disabled.
    Agent State Change Buttons are disabled.
    Telephony Group Box is disabled.

20. Click on the Disconnect Button to close connection to the Active XML Server.
    Expected Result:
    Voice Call Handling Application Form is closed.

5.3 Email Handling

The Email Handling Sample shows how the agent handles email type work items. To handle email type work items, the agent must login to the Communication Manager (CM) and connect to the Email Media Store (EMS).Once the agent has logged onto the CM, the agent must connect to the Media Director through Media Proxy. Using valid Media Director IP address and Port number and Media Proxy IP address and Port number, the connection to the Media Director is established. When connected successfully to the Media Director, the agent must register to the EMS. When the EMS is registered successfully, the agent is assigned email type work items. Once the agent receives the email work item, the agent can open the incoming email and reply to the email. The agent also receives the phantom call on the voice station on which the agent is logged on. The sample application maintains a log file named EmailLogs.log which stores the logs related to the Email Handling. The log file is stored in the installation Directory.

Steps

1. Start Microsoft Visual Studio.

2. From the Visual Studio menu option, select File > New > Project.

3. Add the following dependencies:
    - ASXMLClient.dll
    - CSTASchemas.dll
    - ASEmailMediaStore.Email.dll
    - ASEmailMediaStore.InboundMail.dll
    - ASEmailMediaStore.Interface.dll
    - ASEmailMediaStore.WorkItem.dll
    - ASGQEClientComm.dll
    - ASGQEDataStoreComm.dll
    - ASGQEGeneral.dll
    - ASGUIEmailControl.dll
    - ASMediaControllerInterface.dll
    - ASMediaStorePluginInterface.dll
    - GQEError.dll
    - log4net.dll
    - GenuineChannels.dll
    - ICSharpCode.SharpZipLib.dll

4. From the Visual Studio menu option, select Project > Add New Item > Application Configuration File.

5. Open the file and paste the following code between the <configuration></configuration> tags.

```
<configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"
    />
</configSections>
<log4net>
<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
<file value="EmailLogs.log" />
<appendToFile value="true" />
<maximumFileSize value="1000KB" />
<maxSizeRollBackups value="2" />
<layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%date %-5level [%thread] - %message%newline" />
</layout>
</appender>
<root>
<level value="DEBUG" />
<appender-ref ref="RollingFile" />
</root>
</log4net>
<system.runtime.remoting>
<application>
    <channels>
            <channel type="Belikov.GenuineChannels.GenuineTcp.GenuineTcpChannel,
            GenuineChannels"
            InvocationTimeout = "180000"
            ConnectTimeout = "20000"
            MaxTimeSpanToReconnect = "10000"
            ReconnectionTries = "1"
            NoSizeChecking = "true"/>
    </channels>
</application>
</system.runtime.remoting>

Details on parameters above:

maxSizeRollBackups indicates property which Gets or sets the maximum number of backup files
that are kept before the oldest is erased.

InvocationTimeout, ConnectTimeout, MaxTimeSpanToReconnect are in milliseconds.
```

Select the Program.cs file from the Solution Explorer.

6. Add the following namespaces:

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels
```

7. Add the following code in the main() of Program.cs file:

```
RemotingConfiguration.Configure("EmailHandlingApplication.exe.config", false);
```

The EmailHandling.cs file is used to connect to the XML Server and start/stop monitor.

8. Add the following namespaces:

```
using AgileSoftware.Developer;
using AgileSoftware.Developer.CSTA;
using AgileSoftware.Developer.XML;
using AgileSoftware.Developer.Station;
using AgileSoftware.Developer.Device;
using System.Threading;
using System.Text.RegularExpressions;
using System.Numeric;
using log4net;
using log4net.Config;
using System.Drawing.Imaging;
using AgileSoftware.Multimedia;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Proxies;
using AgileSoftware.Multimedia.MediaStore.EMS.Email;
using AgileSoftware.CommonControl.Email;
using System.IO;
using System.Diagnostics;
using AgileSoftware.Multimedia.MediaStore.EMS.InboundEmail;
using AgileSoftware.Multimedia.MediaStore.EMS.Interface;
using AgileSoftware.Multimedia.WorkItem;
```

9. Initialize a new instance of AgileSoftware.Developer.ASXMLClient class. (Refer Step 7 of the Agent Login)

   Initialize a new instance of AgileSoftware.Developer.ASMediaClient class.

   C# code:

```
//Create instance of ASMediaClient
ASMediaClient mediaClient = null;
mediaClient = new ASMediaClient();
```

   Declare variables for email work item:

   C# code:

```
//List of registered Media Stores.
MediaStoreInfoList mediaStoreList = null;
//Interface that is used to establish a channel between Media Client and Media Store.
IASGQEMediaStore ems = null;
//Create work item object
ASGQEWorkItemBase2 workItem = null;
//Create object to store email work item data
EmailWorkItemData emailWID = null;
//Create object of Mime message
MimeMessage mimeMessage = null;
```

10. Subscribe to the ASXMLClient and ASMediaClient events. (Refer Step 8 of Agent Login)

    In addition, subscribe to the following CSTA and ASMediaClient events:

C# code:

```
//ASMediaClient Events
xmlClient.CSTADelivered += new CSTADeliveredEventHandler(xmlClient_CSTADelivered);
xmlClient.CSTAEstablished += new CSTAEstablishedEventHandler(xmlClient_CSTAEstablished);
xmlClient.CSTACallCleared += new CSTACallClearedEventHandler(xmlClient_CSTACallCleared);
xmlClient.CSTAConnectionCleared += new
CSTAConnectionClearedEventHandler(xmlClient_CSTAConnectionCleared);

//Media Client Events
mediaClient.GQEError += new GQEErrorEventHandler(mediaClient_GQEError);
mediaClient.MediaTypeRegistered += new
AgileSoftware.Multimedia.Client.MediaTypeRegisteredEventHandler(mediaClient_MediaTypeRegister
ed);
mediaClient.MediaTypeUnregistered += new
AgileSoftware.Multimedia.Client.MediaTypeUnregisteredEventHandler(mediaClient_MediaTypeUnregi
stered);
mediaClient.WorkItemRefDelivered += new
AgileSoftware.Multimedia.Client.WorkItemRefDeliveredEventHandler(mediaClient_WorkItemRefDeliv
ered);
mediaClient.WorkItemRefEstablished += new
AgileSoftware.Multimedia.Client.WorkItemRefEstablishedEventHandler(mediaClient_WorkItemRefEst
ablished);
mediaClient.WorkItemRefRemoved += new
AgileSoftware.Multimedia.Client.WorkItemRefRemovedEventHandler(mediaClient_WorkItemRefRemoved
);
```

11. Browse for available Server Telephony Links:
    (Refer Step 9 of Agent Login)

12. Connect to XML Server:
    (Refer Step 10 of Agent Login)

13. Disconnect from XML Server:
    (Refer Step 11 of Agent Login)

14. Start Monitor:
    (Refer Step 12 of Agent Login)

15. Stop Monitor Station:
    (Refer Step 13 of Agent Login)

16. Agent State Change:
    (Refer Step 14 of Agent Login)

    The MediaDirector.cs file connects to the Media Director and registers to the Media Store.

17. Connect to Media Director through Media Proxy:

Connects to the Media Director through the Media Proxy to receive media work items from the Media Client.

C# code:

```
//Following method connects the Media Director with IP address <135.156.23.22> and Port no.
29087 through Media Proxy with IP address <135.156.23.22> and Port no. 29079 on station 6511.
//mediaClient is ASMediaClient used for all multimedia work items.
//Method used : Connect(mediaTypes,error)
//Parameters :
//mediaTypes(Int32[]) – specifies the media type of the work item it needs to receive from
Media Proxy. It will automatically register the specified media type after it connects to the
Media Director through the Media Proxy Server. Set its value to null to register all Media
Types.
//error(GQEErrorArgs) – indicates the error when connecting to the server failed.
mediaClient.StationDN = “6511”;
 mediaClient.ServerIP = “135.156.23.22”;
 mediaClient.ServerPort = 29087;
 mediaClient.ProxyIP = “135.156.23.22”
 mediaClient.ProxyPort = 29079;
mediaClient.ChannelType = "gtcp";
bool connToMediaDirector = mediaClient.Connect(null, out errorReturned);
//connToMediaStore = true if connected successfully to Media Director.

Details on parameters above:

“ChannelType” is a property of mediaClient which gets/sets .Net remoting configuration file.
If its value is left as empty, the component will automatically register a TCP channel on
port 0.

“gtcp” is genuine channel TCP protocol which uses .Net remoting for a socket connection.
```

18. Disconnect from Media Director:

Closes the connection with the Media Director.

C# code:

```
//Disconnects from Media Director.
//Method used : Disconnect()
mediaClient.Disconnect();
```

19. Register the Media Types:

Registers the Media type for which the users need to receive work items.

C# code:

```
//RegisterMediaType: Registers a media type for which users need to receive work items.
//Method used : RegisterMediaType(mediaType,involeRef)
//Parameters:
//mediaType(Int32): Specifies the media type for which the work items will be received. Set
its  value to 0 to register all the media types.
//invokeRef(String): Returns a unqiue reference of this request in GUID format.
//Positive acknowledgement : MediaTypeRegistered event will be fired.
//Negative acknowledgement : GQEError event fires when it fails.

mediaClient.RegisterMediaType(0, out invoke_ref);
```

20. Unregister the Media Types:

Unregisters the Media type for which the users do not need to receive work items anymore.

C# code:

```
//Unregister the media types.
//Method used : UnregisterMediaType(mediaType,involeRef)
//Parameters :
//mediaType(Int32) – specifies the media type user needs to unregister. Set its  value to 0
to unregister all the media types.
//invokeRef(String) – returns unique reference of the request in GUID format.
//Positive acknowledgement : MediaTypeUnregistered event will be fired.
//Negative acknowledgement : GQEError event fires when it fails.
mediaClient.UnregisterMediaType(0, out invoke_ref);
```

21. Connect to the Email Media Store:

Establishes a remoting connection with the Media Store based on specified URL.

C# code:

```
//Connects to the Email Media Store.
//Method used : ConnectToMediaStore(mediaStoreURL,errorText)
//Parameters :
//mediaStoreURL – specifies the URL of the Media Store that it connects to.
//errorText – returns the error text when it fails to connect to the Media Store.
//Positive acknowledgement : returns a "AgileSoftware.Multimedia.IASGQEMediaStore" object
when connects successfully to the Media Store
//Negative acknowledgement : returns null when failed to connect to Media Store
//Get Media Store list
mediaStoreList = mediaClient.GetMediaStoreList();
//For Email Media type(1), connect to the media store using remoting URL

for (int i = 0; i < mediaStoreList.Count; i++)
{

    if (mediaStoreList[i].MediaType == 1)
        {

            ems = mediaClient.ConnectToMediaStore(mediaStoreList[i].RemotingURL, out
            error_connectMS);

            ems =
            (AgileSoftware.Multimedia.IASGQEMediaStore)Activator.GetObject(typeof(AgileSoftwar
            e.Multimedia.IASGQEMediaStore), mediaStoreList[i].RemotingURL);

            ems.GetAutoTextKeySet("Dummy");

        }
}

Details of MediaType parameter above:

public const int WORK_ITEM_TYPE_EMAIL = 1;
public const int WORK_ITEM_TYPE_FAX = 2;
public const int WORK_ITEM_TYPE_PREVIEWCALL = 3;
public const int WORK_ITEM_TYPE_SIMPLE_MESSAGE = 4;
public const int WORK_ITEM_TYPE_VOICE = 5;
public const int WORK_ITEM_TYPE_CUSTOM_ACTION = 6;
```

The EmailWI.cs file is used to handle the Email work item and the phantom call.

22. Email Work Item Handling:

The ASMediaClientWorkItemRefDelivered event fires to notify the client that a work item (email work item) is delivered.

This event will contain the reference to the work item in the workItemRef parameter. The client will use this work item reference to get access to the remoting object. Communication about the work item will then take place directly between the client and the Data Store.

The client side application can present the content of the work item to users when it fires, but should not allow the users to manipulate the work item at this stage. The users should only be allowed to manipulate the work item when the work item has been accepted by the user - after the WorkItemRefEstablished event fires.

C# code:

```
//Check if the work item delivered is an email type work item.
//If it is an email work item store work item reference to retrieve various email properties.

void mediaClient_WorkItemRefDelivered(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefDeliveredArgs arg)
{

    //Check if Email type work item
    if (arg.WorkItemType != ASGQEWorkItemBase2.WORK_ITEM_TYPE_EMAIL)
    {
    //Not an email work item
    return;
    }
    //received an email work Item

    if (ems != null)
    {
    //Store work item reference
    workItem = arg.WorkItemRef as ASGQEWorkItemBase2;
        workItem.KeepAlive();

    if (workItem != null)
        {
            //Store work item Data as Email Work Item Data
            emailWID = workItem.GetWorkItemData() asEmailWorkItemData;
            //Store Email Work Item Data as Mime Message
            mimeMessage = ((emailWID == null) || (emailWID.EmailMessage == null)) ? null : new
            MimeMessage(emailWID.EmailMessage);
            //Store attachments from Email
            mimeattachments = mimeMessage.Attachments;
            //Used when sending reply email message.
            messageProcessor = workItem as IMessageProcessor;
        }

     }

  }
```

The ASMediaClientWorkItemRefEstablished event fires to notify the client that the work item (Email work item) has been accepted at the client side.

The system uses the phantom call to deliver the work items, this event fires when the phantom call is answered by the client.

C# code:

```
//Email work item accepted by the agent
void mediaClient_WorkItemRefEstablished(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefEstablishedArgs arg)
{
    if (arg.WorkItemType != ASGQEWorkItemBase2.WORK_ITEM_TYPE_EMAIL)
    {
            //Not email work item
            return;
    }
    //Email work item
    if (ems != null)
    {
            workItem = arg.WorkItemRef asASGQEWorkItemBase2;
            workItem.KeepAlive();
            if (workItem != null)
            {
            //Agent accepted Email work item
            }
    }
}
```

Display the email message fields:

The following properties of MimeMessage can be used to retrieve the email message fields:

C# code:

```
//Store email fields like To,From,Cc,Bcc,Subject,Body,Attachments.
mimeMessage.From;
mimeMessage.To;
mimeMessage.Cc;
mimeMessage.Bcc;
mimeMessage.Subject;
mimeMessage.Date;
//if email body is Text type.
mimeMessage.TextBody;
//if email body is HTML type.
mimeMessage.DecodedBody;
mimeMessage.Attachments;
```

Open attachment:

The following method opens an email attachment:

C# code:

```
//Get filename and store bytes
string filename;
filename = mimeMessage.Attachments.AttachmentFileNames;
Bytes = mimeAttachments[filename];
//Create a temporary file
string attachmentfilename = Path.Combine(Path.GetTempPath(), filename);
File.WriteAllBytes(attachmentfilename, Bytes);
//Open temporary file
Process attachmentProcess = new Process();
attachmentProcess.StartInfo = new ProcessStartInfo(attachmentfilename);
attachmentProcess.Start();
```

Reply to an email:

The following method can be used to send a reply mail to the incoming mail:

C# code:

```
//Create new mime message
MimeMessage reply_mimeMessage = new MimeMessage();
//Set the following properties
reply_mimeMessage.From = mimeMessage.To;
reply_mimeMessage.ReplyTo = mimeMessage.To;
reply_mimeMessage.To = mimeMessage.From;
reply_mimeMessage.Subject = "RE : " + mimeMessage.Subject;
//replyFromAgent contains the reply message body.
reply_mimeMessage.TextBody = replyFromAgent;

//Following method sends the reply email.
byte[] reply_Message = reply_mimeMessage.ToArray();
Guid reply_Guid = Guid.NewGuid();
messageProcessor.StartAction(reply_Guid);
messageProcessor.Send(reply_Guid, reply_Message);
messageProcessor.StopAction(reply_Guid);
```

Close WorkItem:

It is used from the client to the Media Store to indicate that the work item has been completed and can now be closed.

C# code:

```
//Close work item
//Method used : Close(invokeRef,completionStatus)
//invokeRef(String) -returns a unique reference for this method
//completionStatus(Int32) - used to indicate how the work item is completed. It can have
//two values 1 or 0 according to whether it was completed successfully or not respectively.
//workItem.Close(string.Empty, 1);
```

The ASMediaClientWorkItemRefRemoved event fires to notify the client that the agent did not accept the work item (email work item) by answering the call in expected time. The work item will be queued back to the server and try to deliver again.

When the system uses the phantom call to deliver the work items, this event fires when the phantom call is redirected back to the switch hunt group without being answered by the agent or the required time for the agent to answer the call has expired and this phantom call has been dropped by the server. This event does not mean the associated work item has been removed from the queue of the Media Director.

C# code:

```
//Removes the work item
void mediaClient_WorkItemRefRemoved(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefRemovedArgs arg)
{
    //The work item is removed and queued back to the server to be delivered again.
}
```

23. Handling Phantom call:

A phantom call is presented to the agent voice station along with the work item. This phantom call needs to be handled.

The CSTADelivered event will be fired when a phantom call is being presented to a device. The connection ID returned from this event can be used to answer the phantom call.

C# code:

```
//Store the connection ID
CSTAConnectionID connID;
connID = arg.Connection;
```

Answer the phantom call:

C# code:

```
//Following method answers the phantom call on the voice station
//Method used : CSTAAnswerCall(callToBeAnswered)
//callToBeAnswered(CSTAConnectionID) - specifies connection to be answered.
//Positive acknowledgement : CSTAEstablished event will be fired.
//Negative acknowledgement : CSTAErrorReturned event will be fired.
xmlClient.CSTAAnswerCall(connID);
```

Once the phantom call has been answered the agent can move to auxiliary state or ACW state before the phantom call is cleared.

Clear phantom call:

C# code:

```
//Following method clears the phantom call on the voice station
//Method used : CSTAClearCall(callToBeCleared,userData)
//callToBeAnswered(CSTAConnectionID) - specifies connection to be answered.
//userData(String) - specifies user data to be sent to the parties in call.
//Positive acknowledgement : CSTACallCleared and CSTAConnectionCleared event will be fired.
//Negative acknowledgement : CSTAErrorReturned event will be fired.
if (connID != null)
{
   xmlClient.CSTAClearCall(connID, "");
}
```

Set agent to Auxiliary state:

C# code:

```
//Following method sets agent state as Auxiliary for agent with agentID 6711 and password :
1234 on voice station 6511.
//Method used : CSTASetAgentState(DeviceID,AgentState,AgentID,AgentPwd,CSTADeviceID,
PrivateDateSetAgentState)
//Parameters :
//DeviceID(CSTADeviceID) -  specifies the DeviceID for the agent
//AgentState(enReqAgentState) - specifies requested agent state
//AgentId(String) - specifies the agent identifier
//AgentPwd(String) - specifies the agent password
//ACDgroup(CSTADeviceID) - specifies the agent group
//PrivateData(PrivateDataSetAgentState) - specifies private data associated with this method.
//Create a CSTADeviceID object for voice station 6511
//Positive acknowledgement : CSTAAgentNotReady event will be fired.
//Negative acknowledgement : CSTAErrorReturned event will be fired.
CSTADeviceID dev = newCSTADeviceID(“6511”, enDeviceIDType.deviceNumber);
xmlClient.CSTASetAgentState(dev, enReqAgentState.notReady,”6711”, “1234”, null, null);
```

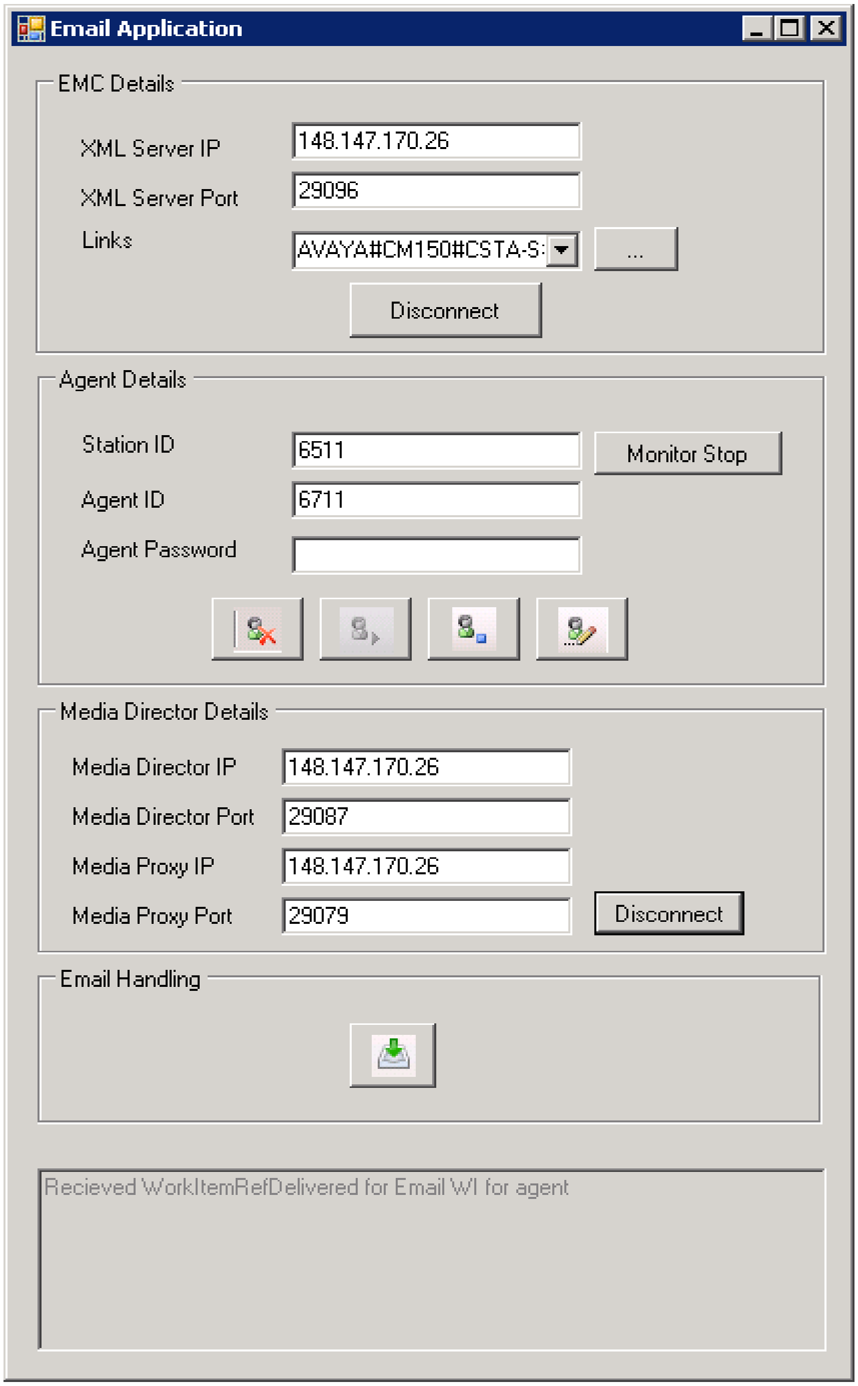
24. Clear events subscribed:
    (Refer Step 15 of Agent Login)

    In addition, clear the following CSTA events:

    C# code:

```
//ASXMLClient Events
CSTAAgentWorkingAfterCallEventHandler(xmlClient_CSTAAgentWorkingAfterCall);
xmlClient.CSTADelivered -= new CSTADeliveredEventHandler(xmlClient_CSTADelivered);
xmlClient.CSTAEstablished -= new CSTAEstablishedEventHandler(xmlClient_CSTAEstablished);
 xmlClient.CSTACallCleared -= new CSTACallClearedEventHandler(xmlClient_CSTACallCleared);
xmlClient.CSTAConnectionCleared -= new
CSTAConnectionClearedEventHandler(xmlClient_CSTAConnectionCleared);

 //Media Client Events
 mediaClient.GQEError -= new GQEErrorEventHandler(mediaClient_GQEError);
mediaClient.MediaTypeRegistered -= new
AgileSoftware.Multimedia.Client.MediaTypeRegisteredEventHandler(mediaClient_MediaTypeRegister
ed);
mediaClient.MediaTypeUnregistered -= new
AgileSoftware.Multimedia.Client.MediaTypeUnregisteredEventHandler(mediaClient_MediaTypeUnregi
stered);
mediaClient.WorkItemRefDelivered -= new
AgileSoftware.Multimedia.Client.WorkItemRefDeliveredEventHandler(mediaClient_WorkItemRefDeliv
ered);
mediaClient.WorkItemRefEstablished -= new
AgileSoftware.Multimedia.Client.WorkItemRefEstablishedEventHandler(mediaClient_WorkItemRefEst
ablished);
mediaClient.WorkItemRefRemoved -= new
AgileSoftware.Multimedia.Client.WorkItemRefRemovedEventHandler(mediaClient_WorkItemRefRemoved
);
```

Screenshot



EMC Details

XML Server IP: 148.147.170.26

XML Server Port: 29096

Links: AVAYA#CM150#CSTA-S:

Agent Details

Station ID: 6511

Agent ID: 6711

Agent Password:

Media Director Details

Media Director IP: 148.147.170.26

Media Director Port: 29087

Media Proxy IP: 148.147.170.26

Media Proxy Port: 29079

Recieved WorkItemRefDelivered for Email WI for agent

Figure3. UI for Email Handling

Steps to run Email Handling Sample UI

1. Enter valid XML Server IP address and XML Server Port number. Click on the Browse Button to browse for available TServer Links.
   Expected Result:
   Available TServer Links are displayed in Combo Box.

2. Select one of the available TServer Links returned. Click on the Connect Button to connect to the Active XML Server.
   Expected Result:
   "Stream Connected" is displayed in Status Bar.
   Agent Details Group Box enabled.

3. Enter a valid Station ID. Click on the Monitor Start Button to start monitoring events at the station.
   Expected Result:
   "Monitor Started" is displayed in Status Bar.
   If agent is already logged in, agent is set to current state. Else Agent ID and Agent Password Textbox are enabled.
   Media Director Group Box is enabled.

4. Enter valid Agent ID and Password. Click Agent Login Button to logon Agent.
   Expected Result:
   Agent is logged on to the system and set to the current Agent State.
   Agent State Change Buttons are enabled.

5. To receive an email work item, the agent must be in available state. Click Available Button to set agent in Available (Ready) state.
   Expected Result:
   "Agent in Available State" is displayed in Status Bar.

6. Enter valid Media Director IP address and Port Number and Media Proxy IP address and Port Number to connect to the Media Director through Media Proxy. After Connecting successfully to the Media Store, register the Media types and connect to the Email Media Store (EMS) to receive email work items.
   Expected Result:
   "Connected to EMS Media Store" is displayed in Status Bar.
   Email Handling Group Box is enabled.

7. Email work item is delivered to agent.
   Expected Result:
   Accept Mail Button is enabled.
   Phantom call delivered to the voice station.

8. Click on the Accept Button to open incoming mail.
   Expected Result:
   Incoming mail will be displayed in a new window.
   Phantom call is cleared on voice station.
   Agent is set to Auxiliary state.

9. Agent may reply to the incoming mail. Click the Reply to Mail button to reply to the incoming mail.
   Expected Result:
   Reply to Customer window will open.

10. Enter the reply message in the given field and Click on the Send Button to send the reply.
    Expected Result:
    Reply mail is sent and the Email Handling Application form is displayed.
    “Work item closed” is displayed in Status Bar.

11. Click on the Disconnect button in the Media Director Group Box to disconnect from the Media Director.
    Expected Result:
    “Media Director disconnected” is displayed in Status Bar.
    Email Handling Group box is disabled.

12. Click on the Monitor Stop button to stop monitoring events at the voice station.
    Expected Result:
    Media Director Details Group Box is disabled.
    Agent State change buttons are disabled.
    “Monitor stopped” is displayed in Status Bar.

13. Click on the Disconnect button to disconnect from the Active XML Server.
    Expected Result:
    Email Handling Application Form is closed.

5.4 SMS Handling

The SMS Handling Sample shows how the agent handles sms type work items. To handle sms type work items, the agent must login to the Communication Manager (CM) and connect to the Simple Messaging Media Store (SMMS). Once the agent has logged onto the CM, the agent must connect to the Media Director through Media Proxy. When connected successfully to the Media Director, the agent must register to the SMMS. When the SMMS is registered successfully, the agent is assigned sms type work items when the agent is in available state. Once the agent receives the sms work item, the agent can open the incoming sms and send an acknowledgement. The agent also receives a phantom call on the voice station on which the agent is logged on. The sample application maintains a log file named smsLogs.log which stores the logs related to the SMS Handling. The log file is stored in the installation Directory.

Steps

1. Start Microsoft Visual Studio.

2. From the Visual Studio menu option, select File > New > Project.

3. Add the following dependencies:
   - ASGQEClientComm.dll
   - ASGQEDataStoreComm.dll
   - ASGQEDataStoreSimpleMessaging.dll
   - ASGQEGeneral.dll
   - ASMediaClient.dll
   - ASMediaControllerInterface.dll
   - ASMediaStorePluginInterface.dll
   - ASSimpleMessagingClientComm.dll
   - ASSimpleMessagingComm.dll
   - ASSimpleMessagingManagementCommon.dll
   - ASSimpleMessagingPlugin.dll
   - ASSMMediaServiceXMLUtilities.dll
   - ASXMLClient.dll
   - CSTASchemas.dll
   - GQEError.dll
   - log4net.dll
   - GenuineChannels.dll
   - ICSharpCode.SharpZipLib.dll

4. From the Visual Studio menu option, select Project > Add New Item > Application Configuration File.

5. Open the file and paste the following code between the <configuration></configuration> tags.

```
<configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"
    />
</configSections>
<log4net>
<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
<file value="smsLogs.log" />
<appendToFile value="true" />
<maximumFileSize value="1000KB" />
<maxSizeRollBackups value="2" />
<layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%date %-5level [%thread] - %message%newline" />
</layout>
</appender>
<root>
<level value="DEBUG" />
<appender-ref ref="RollingFile" />
</root>
</log4net>
<system.runtime.remoting>
<application>
    <channels>
            <channel type="Belikov.GenuineChannels.GenuineTcp.GenuineTcpChannel,
            GenuineChannels"
            InvocationTimeout = "180000"
            ConnectTimeout = "20000"
            MaxTimeSpanToReconnect = "10000"
            ReconnectionTries = "1"
            NoSizeChecking = "true"/>
    </channels>
</application>
</system.runtime.remoting>

Details on parameters above:

maxSizeRollBackups indicates property which Gets or sets the maximum number of backup files
that are kept before the oldest is erased.

InvocationTimeout, ConnectTimeout, MaxTimeSpanToReconnect are in milliseconds.
```

Select the Program.cs file from the Solution Explorer.

6. Add the following namespaces:

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels
```

7. Add the following code in the main() of Program.cs file:

```
RemotingConfiguration.Configure("SMSHandlingApplication.exe.config", false);
```

The SMSHandling.cs file is used to connect to the XML Server and start/stop monitor.

8. Add the following namespaces:

```
using AgileSoftware.Developer;
using AgileSoftware.Developer.CSTA;
using AgileSoftware.Developer.Station;
using AgileSoftware.Developer.XMLClient;
using AgileSoftware.Developer.XML;
using log4net;
using log4net.Config;
using System.Threading;
using System.Numeric;
using System.Text.RegularExpressions;
using System.Drawing.Imaging;
using AgileSoftware.Multimedia.MediaStore;
using AgileSoftware.Multimedia;
using AgileSoftware.Multimedia.MediaStore.SimpleMessaging.MediaServiceXmlUtilities;
using AgileSoftware.Multimedia.MediaStore.SimpleMessaging.Client;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Proxies;
```

9. Initialize a new instance of AgileSoftware.Developer.ASXMLClient class. (Refer Step 7 of the Agent Login)

   Initialize a new instance of AgileSoftware.Developer.ASMediaClient class. (Refer Step 9 of the Email Handling)

   Declare variables for sms work item:

   C# code:

```
//List of registered Media Stores.
MediaStoreInfoList mediaStoreList = null;
//Interface that is used to establish a channel between Media Client and Media Store.
IASGQEMediaStore smms = null;
//Create work item object
ASGQEWorkItemBase2 workItem = null;
//Create a Simple Messaging Client
Public ISimpleMessagingClient smmsclient;
```

10. Subscribe to ASXMLClient and ASMediaClient events: (Refer Step 10 of the Email Handling)

11. Browse for available Server Telephony Links: (Refer Step 9 of the Agent Login)

12. Connect to XML Server: (Refer Step 10 of Agent Login)

13. Disconnect from XML Server: (Refer Step 11 of Agent Login)

14. Start Monitor: (Refer Step 12 of Agent Login)

15. Stop Monitor Station:
    (Refer Step 13 of Agent Login)

    The Agent.cs file is used to change agent states

16. Agent State Change:
    (Refer Step 14 of Agent Login)

    The MediaDirector.cs file connects to the Media Director and registers to the Media Store.

17. Connect to Media Director through Media Proxy:
    (Refer Step 17 of the Email Handling

18. Disconnect from Media Director:
    (Refer Step 18 of the Email Handling)

19. Register the Media Types:
    (Refer Step 19 of the Email Handling)

20. Unregister the Media Types:
    (Refer Step 20 of the Email Handling)

21. Connect to the Simple Messaging Media Store:

    Establishes a remoting connection with the Media Store based on specified URL.

    C# code:

```
//Connects to the Simple Messaging Media Store.
//Method used : ConnectToMediaStore(mediaStoreURL,errorText)
//Parameters :
//mediaStoreURL - specifies the URL of the Media Store that it connects to.
//errorText - returns the error text when it fails to connect to the Media Store.
//Positive acknowledgement : returns a “AgileSoftware.Multimedia.IASGQEMediaStore” object
when connects successfully to the Media Store
//Negative acknowledgement : returns null when failed to connect to Media Store
//Get Media Store list
mediaStoreList = mediaClient.GetMediaStoreList();
//For sms Media type(4), connect to the media store using remoting URL
for (int i = 0; i < mediaStoreList.Count; i++)
 {
 if (mediaStoreList[i].MediaType == 4)
     {

        smms = mediaClient.ConnectToMediaStore(mediaStoreList[i].RemotingURL, out
        error_connectMS);

        smms =
        (AgileSoftware.Multimedia.IASGQEMediaStore)Activator.GetObject(typeof(AgileSoftware.Mu
        ltimedia.IASGQEMediaStore), mediaStoreList[i].RemotingURL);

      smms.GetAutoTextKeySet("Dummy");

     }
 }
```

```
Details of MediaType parameter above:

public const int WORK_ITEM_TYPE_EMAIL = 1;
public const int WORK_ITEM_TYPE_FAX = 2;
public const int WORK_ITEM_TYPE_PREVIEWCALL = 3;
public const int WORK_ITEM_TYPE_SIMPLE_MESSAGE = 4;
public const int WORK_ITEM_TYPE_VOICE = 5;
public const int WORK_ITEM_TYPE_CUSTOM_ACTION = 6;
```

22. SMS Work Item Handling:

The ASMediaClientWorkItemRefDelivered event fires to notify the client that a work item (sms work item) is delivered.

This event will contain the reference to the work item in the workItemRef parameter. The client will use this work item reference to get access to the remoting object. Communication about the work item will then take place directly between the client and the Data Store.

The client side application can present the content of the work item to users when it fires, but should not allow the users to manipulate the work item at this stage. The users should only be allowed to manipulate the work item when the work item has been accepted by the user - after the WorkItemRefEstablished event fires.

C# code:

```
//Check if the work item delivered is an sms type work item.
//If it is an sms work item store work item reference.

void mediaClient_WorkItemRefDelivered(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefDeliveredArgs arg)
{
if (arg.WorkItemType != ASGQEWorkItemBase2.WORK_ITEM_TYPE_SIMPLE_MESSAGE)
    {
     //Received work item is not an sms.
     return;
     }
     //Received sms work item.
}
```

The ASMediaClientWorkItemRefEstablished event fires to notify the client that the work item (sms work item) has been accepted at the client side.

When the system uses the phantom call to deliver the work items, this event fires when the phantom call is answered by the client.

Following code displays the sms work item and sends an acknowledgement to the customer.

C# code:

```
void mediaClient_WorkItemRefEstablished(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefEstablishedArgs arg)
{
    if (arg.WorkItemType != ASGQEWorkItemBase2.WORK_ITEM_TYPE_SIMPLE_MESSAGE)
    {
            //Received work item is not an sms.
            return;
    }

    //Received sms work item.
    if (smms != null)
     {
            workItem = arg.WorkItemRef as ASGQEWorkItemBase2;
            workItem.KeepAlive();

            if (workItem != null)
            {
                    //SMS client initialization
                    string messages = String.Empty;
                    string mediaTypeName = String.Empty;
                    bool mediaconnected = false;
                    bool available2send = false;
                    List<SMSimpleMessage> messageList = null;
                    string customername;
                    string initialMessage;

                    //Create simple messaging client
                    smmsclient = (ISimpleMessagingClient)workItem;

                    //Recieve simple message sent by the client
                    smmsclient.SMClientWorkItemLoaded(out messages, outmediaTypeName,
                                                    outavailable2send, out mediaconnected);

                    SMXmlParser.deserializeSimpleMessages(messages, out messageList);

                    customername = ((SMSimpleUserMessage)messageList[0]).Author;
                    initialMessage = messageList[0].Message;

                    //customername and initialMessage return Customer name and the sms message
                    send by the customer.

                    //Reply to the customer
                    string reply_message = "Welcome to EMC support";
                    SMSimpleUserMessage replyMessage = new SMSimpleUserMessage(reply_message,
                    "text/plain", txtAgentID.Text.Trim());
                    replyMessage.Direction = SMSimpleMessageDirection.OutboundFromAgent;

                    string xmlText = String.Empty;
                    SMXmlWriter.serializeSimpleMessage(replyMessage, out xmlText);
                    smmsclient.SMClientSendMessage(xmlText);

                    //Close SMMS WorkItem
                    //Method used : Close(invokeRef,completionStatus)
                    //invokeRef(String) - returns a unique reference for this method
                    //completionStatus(Int32) - used to indicate how the work item is
                    completed. It can have two values 1 or 0 according to whether it was
                    completed successfully or not respectively.
                    workItem.Close(String.Empty, 1);

            }

     }
}
```

The ASMediaClientWorkItemRefRemoved event fires to notify the client that the agent did not accept the work item (sms work item) by answering the call in expected time. The work item will be queued back to the server and try to deliver again.

When the system uses the phantom call to deliver the work items, this event fires when the phantom call is redirected back to the switch hunt group without being answered by the agent or the required time for the agent to answer the call has expired and this phantom call has been dropped by the server. This event does not mean the associated work item has been removed queue of the Media Director.

C# code:

```
//Removes the work item
void mediaClient_WorkItemRefRemoved(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefRemovedArgs arg)
{
    //The work item is removed and queued back to the server to be delivered again.
}
```

Close WorkItem:

It is used from the client to the Media Store to indicate that the work item has been completed and can now be closed.

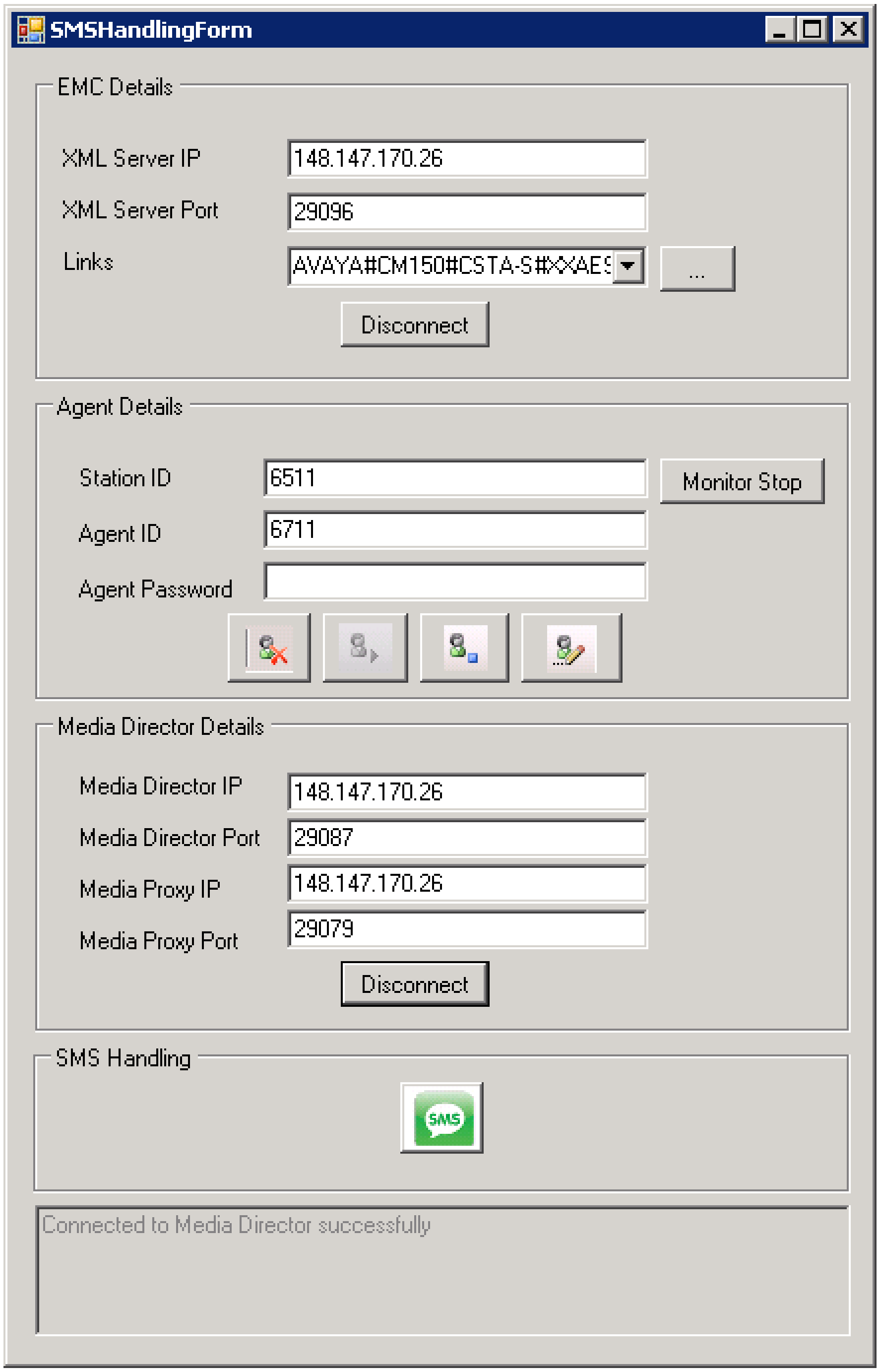
C# code:

```
//Close work item
//Method used : Close(invokeRef,completionStatus)
//invokeRef(String) - returns a unique reference for this method
//completionStatus(Int32) - used to indicate how the work item is completed. It can have
//two values 1 or 0 according to whether it was completed successfully or not respectively.
workItem.Close(string.Empty, 1);
```

23. Handling Phantom call:
    (Refer Step 23 of the Email Handling)

24. Clear subscribed events:
    (Refer Step 24 of the Email Handling)

Screenshot



EMC Details

XML Server IP: 148.147.170.26

XML Server Port: 29096

Links: AVAYA#CM150#CSTA-S#XXAES

Agent Details

Station ID: 6511

Agent ID: 6711

Agent Password:

Media Director Details

Media Director IP: 148.147.170.26

Media Director Port: 29087

Media Proxy IP: 148.147.170.26

Media Proxy Port: 29079

SMS Handling

Connected to Media Director successfully

Figure4. UI for SMS Handling.

Steps to run SMS Handling UI

1. Enter valid XML Server IP address and XML Server Port number. Click on the Browse Button to browse for available TServer Links.
   Expected Result:
   Available TServer Links are displayed in Combo Box.

2. Select one of the available TServer Links returned. Click on the Connect Button to connect to the Active XML Server.
   Expected Result:
   "Stream Connected" is displayed in Status Bar.
   Agent Details Group Box enabled.

3. Enter a valid Station ID. Click on the Monitor Start Button to start monitoring events at the station.
   Expected Result:
   "Monitor Started" is displayed in the Status Bar.
   If agent is already logged in, agent is set to current state. Else Agent ID and Agent Password Textbox are enabled.
   Media Director Group Box is enabled.

4. Enter valid Agent ID and Password. Click Agent Login Button to login the agent.
   Expected Result:
   Agent is logged on to the system and set to the current Agent State.
   The current Agent State is displayed in Status Bar.
   Agent State Change Buttons are enabled.

5. To receive a sms work item, the agent must be in available state. Click Available Button to set agent in Available (Ready) state.
   Expected Result:
   "Agent in Available State" is displayed in Status Bar.

6. Click Agent Logout Button to logout Agent.
   Expected Result:
   "Agent Logged Off" is displayed in Status Bar.
   Agent State Change Buttons are disabled.

7. Enter valid Media Director IP address and Port Number and Media Proxy IP address and Port Number to connect to the Media Director through Media Proxy. After connecting successfully to the Media Store, register the Media types and connect to the Simple Messaging Media Store (SMMS) to receive sms work items.
   Expected Result:
   "Connected to SMMS Media Store" is displayed in the Status Bar.
   SMS Handling Group Box is enabled.

8. SMS work item is delivered to the agent.
   Expected Result:
   Accept Button is enabled.
   Phantom call delivered to the voice station.

9. Click on the Accept Button to open the sms.
   Expected Result:
   SMS will be displayed in a new window.
   Phantom call is cleared on the voice station.
   Agent is set to the Auxiliary state.

10. Agent may send an acknowledgement to the SMS. Click the Send Acknowledgement button to reply to the incoming mail.
    Expected Result:
    Reply sms is sent to the Customer.

11. Click on the Disconnect button in the Media Director Group Box to disconnect from the Media Director.
    Expected Result:
    "Media Director disconnected" is displayed in the Status Bar.
    SMS Handling Group box is disabled.

12. Click on the Monitor Stop button to stop monitoring events at the voice station.
    Expected Result:
    Media Director Details Group Box is disabled.
    Agent State change buttons are disabled.
    "Monitor stopped" is displayed in Status Bar.

13. Click on the Disconnect button to disconnect from the Active XML Server.
    Expected Result:
    SMS Handling Application Form is closed.

5.5 WebChat Handling

The Web Chat Handling Sample shows how the agent handles web chat type work items. To handle the web chat type work items, the agent must login to the Communication Manager (CM) and connect to the Simple Messaging Media Store (SMMS). Once the agent has logged onto the CM, the agent must connect to the Media Director through Media Proxy. When connected successfully to the Media Director, the agent must register to the SMMS. When the SMMS is registered successfully, the agent is assigned web chat type work items when the agent is in available state. Once the agent receives the web chat work item, the agent can start a conversation with the customer. The agent also receives a phantom call on the voice station on which the agent is logged on. The sample application maintains a log file named webChatLogs.log which stores the logs related to the Web Chat Handling. The log file is stored in the installation Directory.

Steps

1. Start Microsoft Visual Studio.

2. From the Visual Studio menu option, select File > New > Project.

3. Add the following dependencies:
    - ASGQEClientComm.dll
    - ASGQEDataStoreComm.dll
    - ASGQEDataStoreSimpleMessaging.dll
    - ASGQEGeneral.dll
    - ASMediaClient.dll
    - ASMediaControllerInterface.dll
    - ASMediaStorePluginInterface.dll
    - ASSimpleMessagingClientComm.dll
    - ASSimpleMessagingComm.dll
    - ASSimpleMessagingManagementCommon.dll
    - ASSimpleMessagingPlugin.dll
    - ASSMMediaServiceXMLUtilities.dll
    - ASXMLClient.dll
    - CSTASchemas.dll
    - GQEError.dll
    - log4net.dll
    - GenuineChannels.dll
    - ICSharpCode.SharpZipLib.dll

4. From the Visual Studio menu option, select Project > Add New Item > Application Configuration File.

5. Open the file and paste the following code between the <configuration></configuration> tags.

```
<configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"
    />
</configSections>
<log4net>
<appender name="RollingFile"type="log4net.Appender.RollingFileAppender">
<file value="webChatLogs.log" />
<appendToFile value="true" />
<maximumFileSize value="1000KB" />
<maxSizeRollBackups value="2" />
<layout type="log4net.Layout.PatternLayout">
<conversionPattern value="%date %-5level [%thread] - %message%newline" />
</layout>
</appender>
<root>
<level value="DEBUG" />
<appender-ref ref="RollingFile" />
</root>
</log4net>
<system.runtime.remoting>
<application>
    <channels>
           <channel type="Belikov.GenuineChannels.GenuineTcp.GenuineTcpChannel,
           GenuineChannels"
           InvocationTimeout = "180000"
           ConnectTimeout = "20000"
           MaxTimeSpanToReconnect = "10000"
           ReconnectionTries = "1"
           NoSizeChecking = "true"/>
    </channels>
</application>
</system.runtime.remoting>

Details on parameters above:

maxSizeRollBackups indicates property which Gets or sets the maximum number of backup files
that are kept before the oldest is erased.

InvocationTimeout, ConnectTimeout, MaxTimeSpanToReconnect are in milliseconds.
```

Select the Program.cs file from the Solution Explorer.

6. Add the following namespaces:

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels
```

7. Add the following code in the main() of Program.cs file:

```
RemotingConfiguration.Configure("WebChatHandlingApplication.exe.config", false);
```

The WebChatApplication.cs file is used to subscribe to the CSTA and ASMediaClient events.

8. Add the following namespaces:

```
using AgileSoftware.Developer;
using AgileSoftware.Developer.CSTA;
using AgileSoftware.Developer.Station;
using AgileSoftware.Developer.XMLClient;
using AgileSoftware.Developer.XML;
using log4net;
using log4net.Config;
using System.Threading;
using System.Numeric;
using System.Text.RegularExpressions;
using System.Drawing.Imaging;
using AgileSoftware.Multimedia.MediaStore;
using AgileSoftware.Multimedia;
using AgileSoftware.Multimedia.MediaStore.SimpleMessaging.MediaServiceXmlUtilities;
using AgileSoftware.Multimedia.MediaStore.SimpleMessaging.Client;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Proxies;
```

9. Initialize a new instance of AgileSoftware.Developer.ASXMLClient class. (Refer Step 7 of the Agent Login)

   Initialize a new instance of AgileSoftware.Developer.ASMediaClient class. (Refer Step 9 of the Email Handling)

   Declare variables for web chat work item:

   C# code:

```
//List of registered Media Stores.
MediaStoreInfoList mediaStoreList = null;
//Interface that is used to establish a channel between Media Client and Media Store.
IASGQEMediaStore smms = null;
//Create work item object
ASGQEWorkItemBase2 workItem = null;
//Create a Simple Messaging Client
Public ISimpleMessagingClient smmsclient;
//Web chat variables
string messages = String.Empty;
string mediaTypeName = String.Empty;
bool mediaconnected = false;
bool available2send = false;
List<SMSimpleMessage> messageList = null;
string customername;
string sendingText;
```

10. Subscribe to ASXMLClient and ASMediaClient events: (Refer Step 10 of the Email Handling)

The ConnectToXMLServer.cs file is used to connect to the XML Server.

11. Browse for available Server Telephony Links: (Refer Step 9 of Agent Login)

12. Connect to XML Server:
    (Refer Step 10 of Agent Login)

13. Disconnect from XML Server:
    (Refer Step 11 of Agent Login)

14. Start Monitor:
    (Refer Step 12 of Agent Login)

15. Stop Monitor Station:
    (Refer Step 13 of Agent Login)

    The Agent.cs file is used to change agent states.

16. Agent State Change:
    (Refer Step 14 of the Agent Login)

    The MediaDirector.cs file connects to the Media Director and registers to the Media Store.

17. Connect to Media Director through Media Proxy:
    (Refer Step 17 of the Email Handling)

18. Disconnect from Media Director:
    (Refer Step 18 of the Email Handling)

19. Register the Media Types:
    (Refer Step 19 of the Email Handling)

20. Unregister the Media Types:
    (Refer Step 20 of the Email Handling)

21. Connect to the Simple Messaging Media Store:
    (Refer Step 21 of the SMS Handling)

22. Web Chat Work Item Handling:

    The ASMediaClientWorkItemRefDelivered event fires to notify the client that a work item (web chat work item) is delivered.

    This event will contain the reference to the work item in the workItemRef parameter. The client will use this work item reference to get access to the remoting object. Communication about the work item will then take place directly between the client and the Data Store.

The client side application can present the content of the work item to users when it fires, but should not allow the users to manipulate the work item at this stage. The users should only be allowed to manipulate the work item when the work item has been accepted by the user - after the WorkItemRefEstablished event fires.

C# code:

```
//Check if the work item delivered is an sms type work item.
//If it is an sms work item store work item reference.

void mediaClient_WorkItemRefDelivered(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefDeliveredArgs arg)
{
    if (arg.WorkItemType != ASGQEWorkItemBase2.WORK_ITEM_TYPE_SIMPLE_MESSAGE)
   {
            //Received work item is not an sms.
            return;
    }
    //Received sms work item.
}
```

The ASMediaClientWorkItemRefEstablished event fires to notify the client that the work item (web chat work item) has been accepted at the client side.

When the system uses the phantom call to deliver the work items, this event fires when the phantom call is answered by the client.

Following code displays the web chat work item and sends an acknowledgement to the customer.

C# code:

```
void mediaClient_WorkItemRefEstablished(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefEstablishedArgs arg)
{
    if (arg.WorkItemType != ASGQEWorkItemBase2.WORK_ITEM_TYPE_SIMPLE_MESSAGE)
         {
            //Not an web chat work item
            return;
         }
    //Web chat work item recieved
    if (smms != null)
    {
             workItem = arg.WorkItemRef asASGQEWorkItemBase2;
             workItem.KeepAlive();

            if (workItem != null)
            {
                   //Code for handling Web Chat

            }
    }
}
```

Code for handling web chat:

```
//Create simple messaging client
smmsClient = (ISimpleMessagingClient)workItem;

//Get ClientInterfaceVersion
// Interface version will depend on the value defined for any extended interface and will be
typically defined as an integer value in the interface. This can be over ridden

if (2 == getClientInterfaceVersion(smmsClient))
{
    //ClientInterfaceVersion - 2
    smClientEventHandler = new SimpleMessagingClientEventHandler(smmsClient as
    ISimpleMessagingClient2);

    smClientEventHandler.SMReceivedMessage += new
    SMReceivedMessageEventHandler(onSMReceivedMessage);

}
else
{
    //ClientInterfaceVersion - 1
    smClientEventHandler = new SimpleMessagingClientEventHandler(smmsClient);

    smClientEventHandler.SMReceivedMessage += new
    SMReceivedMessageEventHandler(onSMReceivedMessage);

}

customername = "";
messageList = null;

//Recieve simple message sent by the client
smmsClient.SMClientWorkItemLoaded(out messages, out mediaTypeName, out available2send, out
mediaconnected);

SMXmlParser.deserializeSimpleMessages(messages, out messageList);

//Gets Client Version
privateint getClientInterfaceVersion(ISimpleMessagingClient simpleMessagingClient)
{
    int ver = 0;
    ver = (simpleMessagingClient asISimpleMessagingClient).Version;
    return ver;
}

//Message recieved
privatevoid onSMReceivedMessage(object sender, SMReceivedMessageArgs arg)
{
    string message = arg.Message;
    if (true == String.IsNullOrEmpty(message))
    {
        return;
    }

    SMSimpleMessage msg = null;
    bool res = SMXmlParser.deserializeSimpleMessage(message, out msg);

    preProcessSimpleMessage(msg);
    AppendMessage(msg);
}

//Pre-Process Simple Message to identify as System Message
protectedstaticvoid preProcessSimpleMessage(SMSimpleMessage msg)
{
    if (msg.Type == SMSimpleMessageType.SystemMessage)
    {
        preProcessSimpleSystemMessage(msg as SMSimpleSystemMessage);
    }
}
```

```
//Pre-Process System Message
publicstaticvoid preProcessSimpleSystemMessage(SMSimpleSystemMessage msg)
{
do
{
    if (false == String.IsNullOrEmpty(msg.Message))
    {
           break;
    }
    string strMessageID = String.Empty;
    strMessageID = msg.MessageID;
    int msgID = 0;
    if (false == int.TryParse(strMessageID, out msgID))
    {
           break;
    }
    else
    {
           msgID = Convert.ToInt32(strMessageID);
           log.Info("Getting string for :" + msgID);
           string message_string = getMessageString(msgID);

           msg.Message = message_string;

     }
} while (false);
}

//Identifies as User Message or System Message
publicvoid AppendMessage(SMSimpleMessage msg)
{
try
{
   if (null != msg)
   {
           //User Message
          if (SMSimpleMessageType.UserMessage == msg.Type)
           {
                   AppendUserMessage((SMSimpleUserMessage)msg);
           }

           //System Message
          if (SMSimpleMessageType.SystemMessage == msg.Type)
           {
                   if (((SMSimpleSystemMessage)msg).SubType != SMSystemMessageSubtype.Status)
                   {
                   AppendSystemMessage((SMSimpleSystemMessage)msg);
                   }
           }
    }
    else
    {
    //Received null message
    }
}
catch (Exception e)
       {
           //Print exception
       }
}

//Appends User Message recieved
publicvoid AppendUserMessage(SMSimpleUserMessage msg)
{
    //Retrieve User Message from msg.Message
}
```

```
//Append System Message
privatevoid AppendSystemMessage(SMSimpleSystemMessage msg)
{
    //Retreive System Message from msg.Message
}

//To reply to Customer chat
if (smmsClient != null)
{
    SMSimpleUserMessage sm = newSMSimpleUserMessage(sendingText, "text/plain","6711”);
    sm.Direction = SMSimpleMessageDirection.OutboundFromAgent;
    string xmlText = String.Empty;
    SMXmlWriter.serializeSimpleMessage(sm, out xmlText);
    smmsClient.SMClientSendMessage(xmlText);
}
Details on parameters above:
```

The ASMediaClientWorkItemRefRemoved event fires to notify the client that the agent did not accept the work item (web chat work item) by answering the call in expected time. The work item will be queued back to the server and try to deliver again.

When the system uses the phantom call to deliver the work items, this event fires when the phantom call is redirected back to the switch hunt group without being answered by the agent or the required time for the agent to answer the call has expired and this phantom call has been dropped by the server. This event does not mean the associated work item has been removed queue of the Media Director.

C# code:

```
//Removes the work item
void mediaClient_WorkItemRefRemoved(object sender,
AgileSoftware.Multimedia.Client.WorkItemRefRemovedArgs arg)
{
    //The work item is removed and queued back to the server to be delivered again.
}
```

Close WorkItem:

It is used from the client to the Media Store to indicate that the work item has been completed and can now be closed.

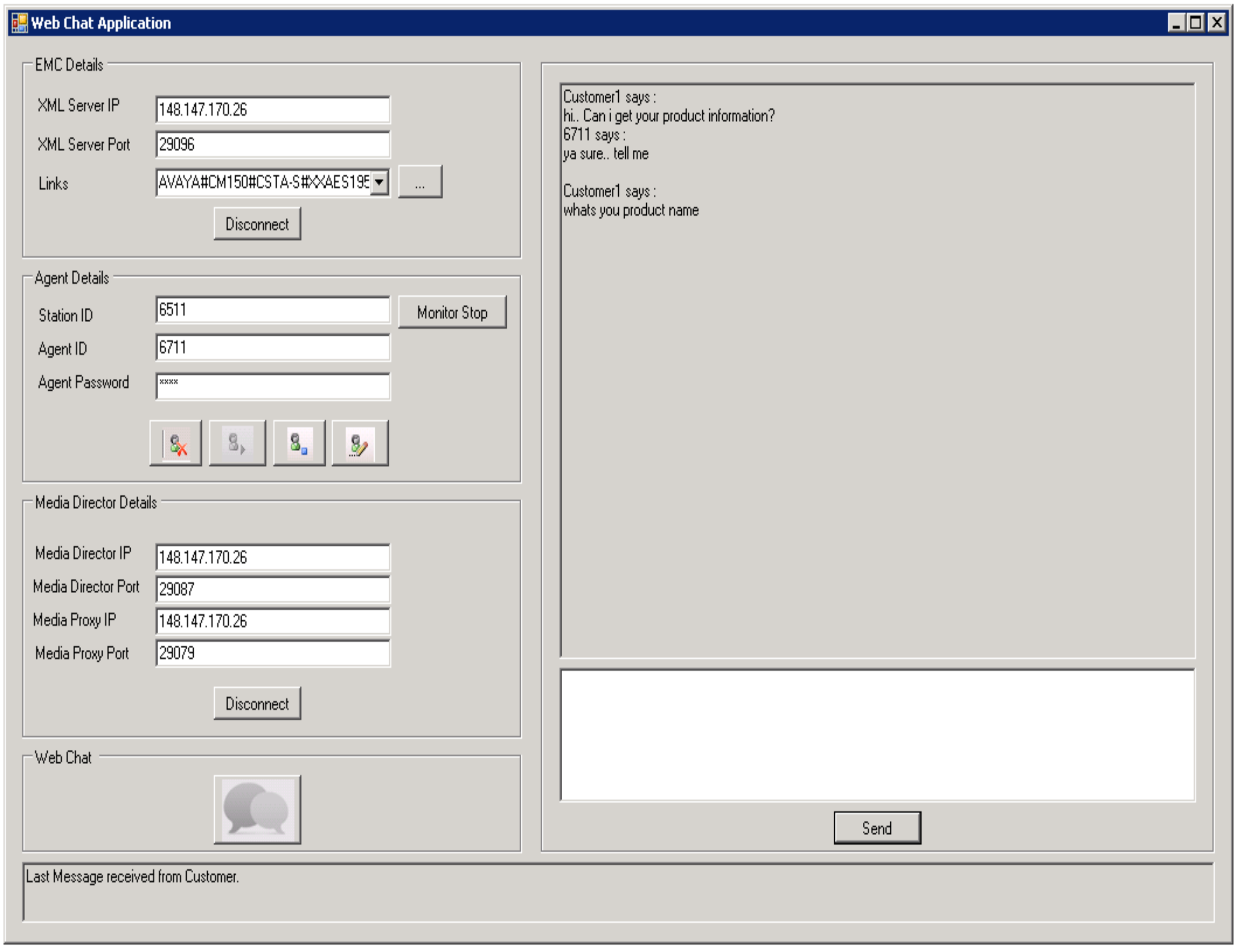
C# code:

```
//Close work item
//Method used : Close(invokeRef,completionStatus)
//invokeRef(String) - returns a unique reference for this method
//completionStatus(Int32) - used to indicate how the work item is completed. It can have
//two values 1 or 0 according to whether it was completed successfully or not respectively.
workItem.Close(string.Empty, 1);
```

23. Handling Phantom call:
    (Refer Step 23 of the Email Handling)

24. Clear subscribed events:
    (Refer Step 24 of the Email Handling)

Screenshot



EMC Details

XML Server IP 148.147.170.26

XML Server Port 29096

Links AVAYA#CM150#CSTA-S#XXAES195

Agent Details

Station ID 6511

Agent ID 6711

Agent Password ****

Media Director Details

Media Director IP 148.147.170.26

Media Director Port 29087

Media Proxy IP 148.147.170.26

Media Proxy Port 29079

Web Chat

```
Customer1 says :
hi.. Can i get your product information?
6711 says :
ya sure.. tell me

Customer1 says :
whats you product name
```

Last Message received from Customer.

Figure5. UI for Web Chat Handling.

Steps to run Web Chat UI

1. Enter valid XML Server IP address and XML Server Port number. Click on the Browse Button to browse for available TServer Links.
   Expected Result:
   Available TServer Links are displayed in Combo Box.

2. Select one of the available TServer Links returned. Click on the Connect Button to connect to the Active XML Server.
   Expected Result:
   "Stream Connected" is displayed in Status Bar.
   Agent Details Group Box enabled.

3. Enter a valid Station ID. Click on the Monitor Start Button to start monitoring events at the station.
   Expected Result:
   "Monitor Started" is displayed in the Status Bar.
   If agent is already logged in, agent is set to current state. Else Agent ID and Agent Password Textbox are enabled.
   Media Director Group Box is enabled.

4. Enter valid Agent ID and Password. Click Agent Login Button to login the agent.
   Expected Result:
   Agent is logged on to the system and set to the current Agent State.
   The current Agent State is displayed in Status Bar.
   Agent State Change Buttons are enabled.

5. To receive a web chat work item, the agent must be in available state. Click Available Button to set agent in Available (Ready) state.
   Expected Result:
   "Agent in Available State" is displayed in Status Bar.

6. Click Agent Logout Button to logout Agent.
   Expected Result:
   "Agent Logged Off" is displayed in Status Bar.
   Agent State Change Buttons are disabled.

7. Enter valid Media Director IP address and Port Number and Media Proxy IP address and Port Number to connect to the Media Director through Media Proxy. After connecting successfully to the Media Store, register the Media types and connect to the Simple Messaging Media Store (SMMS) to receive web chat work items.
   Expected Result:
   “Connected to SMMS Media Store” is displayed in the Status Bar.
   Web Chat Handling Group Box is enabled.

8. Web Chat work item is delivered to the agent.
   Expected Result:
   Accept Button is enabled.
   Phantom call delivered to the voice station.

9. Click on the Accept Button to start conversation with the customer.
   Expected Result:
   Customer message is displayed to the agent.
   Phantom call is cleared on the voice station.
   Agent is set to the Auxiliary state.

10. Agent may reply to the web chat. Click the Send button to reply to the customer.
    Expected Result:
    Reply message is sent to the Customer.

11. Click on the Disconnect button in the Media Director Group Box to disconnect from the Media Director.
    Expected Result:
    “Media Director disconnected” is displayed in the Status Bar.
    Web Chat Handling Group box is disabled.

12. Click on the Monitor Stop button to stop monitoring events at the voice station.
    Expected Result:
    Media Director Details Group Box is disabled.
    Agent State change buttons are disabled.
    “Monitor stopped” is displayed in Status Bar.

13. Click on the Disconnect button to disconnect from the Active XML Server.
    Expected Result:
    Web Chat Handling Application Form is closed.

5.6 PCMS Handling

The mechanism to connect to PCMS is same as Email / Web-Chat. The media type for Preview Contact is PreviewContact (= 3).

Once connected to the Media Director and the PCMS work item is delivered to Agent

1. PCMS Work Item Handling:

   To retrieve the PCMS Contact details, do the following when the Refdelivered/RefEstablished event is fired

   C# code:

```
void MediaController_WorkItemRefDelivered(object sender,
                        AgileSoftware.Multimedia.Client.WorkItemRefDeliveredArgs arg)
{
    try
    {
        if (arg.WorkItemType !=
          AgileSoftware.Multimedia.ASGQEWorkItemBase2.WORK_ITEM_TYPE_PREVIEWCALL)
                return;

        LaunchWorkItemDocumentWindow(arg.WorkItemType,
                    (AgileSoftware.Multimedia.ASGQEWorkItemBase2)arg.WorkItemRef,
                               arg.WorkItemRef.GetCustomerData(), arg.WorkItemID,
                                             arg.UniqueID, arg.KeepAliveInterval);
    }
    catch (Exception ex)
    {
          //log error.
    }
 }
```

   The LaunchWorkItemDocumentWindow is an EMC Desktop specific function which retrieves the contact details and the Queue information

   All the information about the Preview Contact work item details is present in the Customer data dataset. This is the class for that.

```
AgileSoftware.Multimedia.WorkItem.CustomerData
```

To extract information from this data-structure of the Customer contact details do the following

C# code:

```
GetCustomerData("CustomerName",string.Empty, out customerName);
GetCustomerData("CustomerAddress", string.Empty, out customerAddress);
GetCustomerData("CustomerNotes", string.Empty, out customerNotes);
GetCustomerData("FirstContactOption", "First Contact",out firstContactOption);
GetCustomerData("FirstContactOptionDetails", string.Empty, out
                                                       firstContactOptionDetails);
GetCustomerData("SecondContactOption","Second Contact", out secondContactOption);
GetCustomerData("SecondContactOptionDetails", string.Empty, out
                                                       secondContactOptionDetails);
GetCustomerData("ThirdContactOption", "Third Contact",out thirdContactOption);
GetCustomerData("ThirdContactOptionDetails", string.Empty, out
                                                       thirdContactOptionDetails);
GetUserDefinedAdditionalData("UserDefined");
```

The GetCustomerdata and getUserDefinedAdditionaldata is

C# code:

```
private void GetCustomerData(string key, string defaultValue, out string value)
{
     try
     {
         value = defaultValue;
         DataRow[] drs = extraData.Select("Key ='" + key.Trim() + "'");

         if (drs != null && drs.Length > 0)
                 value = Global.ToString(drs[0]["Value"]);
     }
     catch (Exception ex)
     {
           //log error.
     }
 }

 private void GetUserDefinedAdditionalData(string Topic)
 {
     try
     {
         drsUserDefined = extraData.Select("Topic ='" + Topic + "'", "Key");
     }
     catch (Exception ex)
     {
           //log error.
     }
 }
```

Once the contact is clicked by Agent, you can perform the necessary outbound action

The first/second/third contact option can be any of three values

```
            "Simple Messaging";
            "Voice"
            "Email";
```

The contact details contain the value of either phone no / email / SMS outbound no. Depending on the option you can create the corresponding outbound work item.

To create a Voice work item, you need to dial the number selected using the XMLClient / XMLStation CallDial API call. To create the Email and SMMS outbound work items you would need further information present in the Customer data

C# code:

```
private WorkItemSimpleMessagingConfig GetSimpleMessagingConfig(CustomerData cd)
{
        try
        {
              if (cd == null)
                     return null;
              string programID = GetCustomerDataValue("Outbound Program ID",
                                   Topic_Simple_Messaging, cd.QueueConfiguration, false);
              string message = GetCustomerDataValue("Message", Topic_Simple_Messaging,
                                                              cd.ExtraData, false);
              string serviceName = GetCustomerDataValue("Service Name",
                                  Topic_Simple_Messaging, cd.QueueConfiguration, false);
              string contactID = GetContactID(cd, Topic_Simple_Messaging);
              if (programID == string.Empty)
                     Global.PIMBroker.ErrorLogging.AddErrorToListInformation("ASPreviewCo
                     ntactPlugin.QueueConfiguration.GetSimpleMessagingConfig(): No
                     outbound program ID set up on the work item.");
              if (serviceName == string.Empty)
                     Global.PIMBroker.ErrorLogging.AddErrorToListInformation("ASPreviewCo
                     ntactPlugin.QueueConfiguration.GetSimpleMessagingConfig(): No
                     service name set up on the work item.");
             if (contactID == string.Empty)
                     Global.PIMBroker.ErrorLogging.AddErrorToListInformation("ASPreviewCo
                     ntactPlugin.QueueConfiguration.GetSimpleMessagingConfig(): No
                     contact details set up on the work item.");
             if (programID == string.Empty ||
                     serviceName == string.Empty ||
                     contactID == string.Empty)
                    return null;
             else
                    return new WorkItemSimpleMessagingConfig(programID, contactID,
                                  (int)WorkType.SimpleMessaging, message, serviceName);
        }
        catch (Exception ex)
        {
              //log error.
              return null;
        }
}
```

```
private WorkItemEmailConfig GetEmailConfig(CustomerData cd)
{
        try
        {
            if (cd == null)
                return null;
            string programID = GetCustomerDataValue("Outbound Program ID", Topic_Email,
                                                        cd.QueueConfiguration, false);
            string message = GetCustomerDataValue("Message", Topic_Email,
                                                        cd.QueueConfiguration, false);
            string contactID = GetContactID(cd, Topic_Email);
            if (programID == string.Empty)
                     Global.PIMBroker.ErrorLogging.AddErrorToListInformation("ASPreviewCo
                     ntactPlugin.QueueConfiguration.GetEmailConfig(): No outbound program
                     ID set up on the work item.");
            if (contactID == string.Empty)
                     Global.PIMBroker.ErrorLogging.AddErrorToListInformation("ASPreviewCo
                     ntactPlugin.QueueConfiguration.GetEmailConfig(): No contact details
                     set up on the work item.");
            if (programID == string.Empty || contactID == string.Empty)
                    return null;
            else
                    return new WorkItemEmailConfig(programID, contactID,
                                                        (int)WorkType.Emal, message);
        }
        catch (Exception ex)
        {
                 return null;
        }
}
```

The GetCustomerDataValue can be written like this

C# code:

```
private string GetCustomerDataValue(string key, string topic, DataTable dt,
                                                               bool retryWithoutTopic)
{
            try
            {
                if (dt == null)
                     return string.Empty;

                string value = string.Empty;
                System.Data.DataRow[] drs = dt.Select("Key = '" + key + "' AND Topic = '"
                                                                      + topic + "'");
                if ((drs == null || drs.Length <= 0) && retryWithoutTopic)
                    drs = dt.Select("Key = '" + key + "'");
                if ((drs != null) && (drs.Length > 0))
                    value = Global.ToString(drs[0]["Value"]);
                return value.Trim();
            }
            catch (Exception ex)
            {
                return string.Empty;
            }
}
```

5.7 Adding button to Ribbon on EMC Desktop

Pre-requisites for Ribbon implementation:

1. Add references of following libraries

-From EMC Desktop installation folder
- ASRibbonPlugin.dll

-From .Net framework
- PresentationCore.dll
-PresentationFramework.dll
-System.Windows.Controls.Ribbon.dll
-System.Xaml.dll
-WindowsBase.dll
-WindowsFormsIntegration.dll

2. Update interface reference of `IASGUIHost` to `IASGUIHost6`

5.7.1 Existing Custom Plugins

Suppose we have existing custom plugin which is adding some buttons to old EMC Desktop Toolbar. We can reuse the button's click handling for the Ribbon Menu /Ribbon button's click too, so that we can continue to use the same functionality.

Note that ASRibbonPlugin.dll has class `RibbonItem` with following constructor:

```
public RibbonItem(string name, RibbonItemType type, string label,
                  string header, BitmapImage smallImageSource,
                  BitmapImage largeImageSource, string toolTip,
                  string keyTip, bool isVisible, EventHandler click,
                  EventHandler dropdownOpened, EventHandler
                  dropdownClosed, ToolStripItem toolStripItem);
```

This will return an object of `class RibbonItem` (say `Obj_RibbonItem`).
`Obj_RibbonItem.RibbonElement` in turn will give you objects of `RibbonItemType` like

- `RibbonButtom`
- `RibbonMenu`
- `RibbonSplitButton`

based on Ribbon item that you add in the `RibbonTab`

We can reuse current toolbar button’s click handler functionality.

Here is an example:
Let’s say we have an old toolbar button

```
private ToolStripButton m_btnAnswer = new ToolStripButton();
```

And it has a click event handler, like

```
m_btnAnswer.Click += new EventHandler(m_btnAnswer_Click);
```

We need a corresponding button for Ribbon, associated with same handler, so we will declare

```
RibbonButton ribbonAnswerbutton;
var ribbonitem = new RibbonItem("btnVoiceAnswer",
                      RibbonItemType.RibbonButton,
                      “Answer”,  //Label
                      null,      //Needed in case of RibbonMenu
                      RibbonSmallImageAnswer, //Small when collapsed
                      RibbonBigImageAnswer, //Large button if not null
                      “Answers the Call”, //Tooltip
                      "",
                      true,
                      m_btnAnswer_Click, //Reusing old handler
                      null,
                      null,
                      m_btnAnswer);
ribbonAnswerbutton = ribbonitem.RibbonElement as RibbonButton;
```

Note: Ribbon Button will be Large size if Large image is provided above. If only Small Image is provided then Button is small by default.

Ribbon Buttons dynamic resizing will be determined by Microsoft Ribbon Library behavior when ribbon is expanded or collapsed. Please see Microsoft Ribbon documentation to set group definitions, to set allowable Button sizes (like no collapse, Large only, Small only etc).

5.7.2 Adding Buttons to Ribbon from Custom Plugins

This section is applicable to reusing older toolbar button functionality or creating new Ribbon based custom plugins. We will see how to add this Ribbon Button to the Ribbon.

First we need to create `RibbonGroup` and add the Ribbon button to this group:

```
private RibbonGroup ribbonGroup = new RibbonGroup();
ribbonGroup.Name = “answerRibbonGroup”;
ribbonGroup.Header = “AnswerGroup”;
ribbonGroup.Items.Add(ribbonAnswerbutton);
```

Now get the tab where we add the Button Group:

```
RibbonTab customRibbonTab;
customRibbonTab = Singleton.GUIHostPlugin.GetRibbonTab(enTab.Custom)
as RibbonTab;
customRibbonTab.Items.Add(ribbonGroup);
```

Check if the Custom tab is already enabled, if not we need to enable it as follow:

```
if (!customRibbonTab.IsEnabled)
  customRibbonTab.IsEnabled = true;
```

5.7.3 Adding Buttons to Ribbon in New Custom Plugins

After you done with basic steps for custom plugins mentioned at the starting of this section, you need to create new Ribbon button with desired functionality and follow the steps of section **5.7.2** to add these newly created buttons to the EMC Desktop Ribbon.

Chapter 6: Capacities

For capacity information of Avaya Aura® Call Center Elite Multichannel see *Avaya Aura® Call Center Elite Multichannel Overview*. *(Available at Avaya Support Site)*

Chapter 7: References

1. *Avaya Aura® Call Center Elite Multichannel Developer Reference (Available as .chm and .html at DevConnect portal)*
2. *Avaya Aura® Call Center Elite Multichannel Installation Guide (Available at Avaya Support Site)*