

Finesse Desktop Edition and Client Library

**V2.2**

Table of Contents

[Over view 3](#_Toc508730305)

[Dependecies Injection 4](#_Toc508730306)

[FinAgent and FinView 4](#_Toc508730307)

[Build your first application (CSharp C#) 7](#_Toc508730308)

[Setup Library Object 10](#_Toc508730309)

[Login function 10](#_Toc508730310)

[Logout function 10](#_Toc508730311)

[Answer Call function 11](#_Toc508730312)

[Release Call function 11](#_Toc508730313)

[Hold Call function 11](#_Toc508730314)

[Resume Call function 11](#_Toc508730315)

[Signle Step Call Transfer function 12](#_Toc508730316)

[Consult Call function 12](#_Toc508730317)

[Conference Call function 12](#_Toc508730318)

[make Call function 13](#_Toc508730319)

[Update Call data function 13](#_Toc508730320)

[CHANGE status function 13](#_Toc508730321)

[Events Implementation 14](#_Toc508730322)

[New Event Function 15](#_Toc508730323)

[Error Message Function 16](#_Toc508730324)

[Loading Messages Events 17](#_Toc508730325)

[Relogin Message Event 18](#_Toc508730326)

[Fire Load Login Screen Message Event 19](#_Toc508730327)

[Fire USD Internal Event 20](#_Toc508730328)

# Over view

Finesse DE in the finesse desktop edition based on .Net Library of Finesse Client. The solution is provided in the following flavors

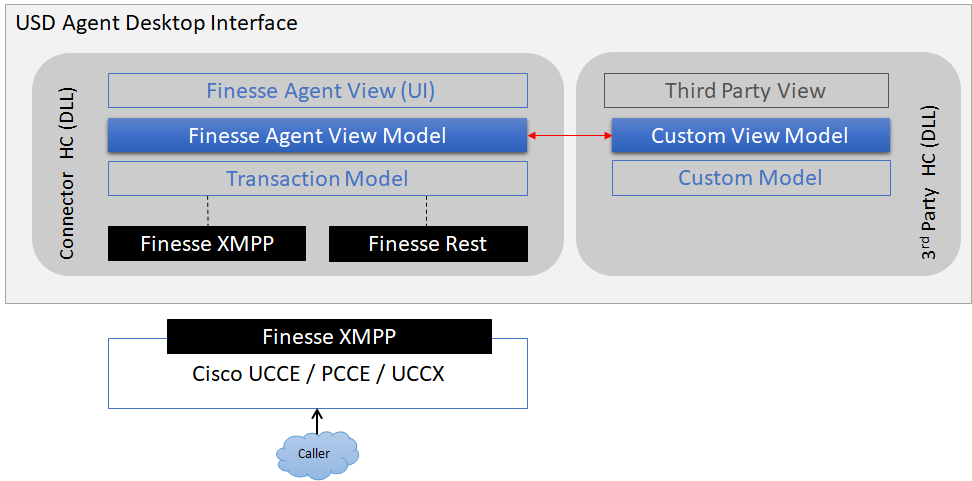
1. Finesse Gadget Desktop Platform Ver 2.2 supporting voice inbound
2. Finesse Client Library Ver 2.2 supporting voice inbound

On the other side, we have a third flavor of Finesse USD Connector based on finesse Client.

USD is a unified service desk for MS CRM. USD is designed for 3rd party customization and integration. IST developed the USD connector as hosted control based on the USD development standard.

The solution is designed based on MVVM

1. Model which is representing the transaction whether a call or chat or email or Task.
2. View is the main UI screen showing the transaction card details and possible functions and capabilities
3. View Model is the Finesse Agent View Model. And this view model is used by the UI for rendering the transaction data on the view.
4. Finesse XMPP and Finesse Rest are the main services adaptors that integrate with Finesse Server side.



3rd party customization is enabled, so you have the option to design your own GUI and link it to the Finesse Agent View Model.

# Dependecies Injection

In order to custom the Finesse Desktop implementation, you have to do the following

* Start Visual Studio Project
* Add project reference to FinesseClient.dll
* Add project reference to System.ValueTuple.dll
* Add project reference to RestSharp.dll
* Add project reference to Matrix.dll

# FinAgent and FinView

The FinAgent is the main class representing all agent related details and actions. FinAgent is representing the solution ViewModel and it is responsible about mapping between the model and front end view.

The Model Consist of the following class

1. AgentInformation
2. Dialogs with all related details (MediaProperties , Participants , Actions)
3. VoiceStatus which is representing a status of voice
4. MessageEvent

FinAgent has the following members

1. AgentInformation .. which is taking care of the all agent related information including the AgentID, Extension , Password , Domain A , Domain B , Dialogs , Status and most of agent details… etc
2. LoadingMessage .. which is taking care of all loading messages during the agent login.
3. LogMessages … which is taking care of all log messages during the application usage.
4. TraceStatus … which is enabling the advanced log messages.
5. SaveLog … which is enabling the engine to save log file by default. The file is saved in your AppData folder with the name FinesseClient\_Date.log. sample “FinesseClient.3\_13\_2018.log”

FinView is the interface representing the signature of the main view screen. In order to build your own finesse agent view , you will need to implement the interface FinView.

FinView is extending the IView which is extension of Finesse DE implementation.

FinView is providing the signature of the following functions:

* FireNewEvent which is invoked whenever new event is resolved by the FinesseClient
* FireCallEvent which is invoked whenever a new call event is placed and resolved by the FinesseClient
* FireLoadingMessage which is invoked whenever a new loading message is available
* FireLogMessage which is invoked whenever a new log message is created
* FireReLoginEvent which is invoked whenever the system is logged in.
* FireLoadLoginScreen which is invoked whenever the system is logged out.
* FireErrorMessage which is invoked whenever the system error is in place
* FireDisconnectEvent which is invoked whenever a system disconnection happen during the agent login time.
* SetContext (Limited to Finesse DE only)

FinAgent is providing out of the box implementation for each of the above mentioned functions except the SetContext. So you may fire your own log message through finesse agent class.

The FireNewEvent could be implemented as the following sample

if (\_finAgent.AgentInformation.MessageEvent == null)

return;

if (\_finAgent.AgentInformation.MessageEvent.MessageType == null)

return;

if (\_finAgent.AgentInformation.MessageEvent.MessageType.Equals("user"))

{

}

else if (\_finAgent.AgentInformation.MessageEvent.MessageType.Equals("call"))

{

}

else if (\_finAgent.AgentInformation.MessageEvent.MessageType.Equals("error"))

{

}

The FireCallEvent could be implemented as the following sample

public void FireCallEvent(Dialog dialog)

{

if (dialog == null)

return;

bool isCallStart = false;

bool isCallEnd = false;

if(dialog.DialogEvent != null && dialog.State != null)

{

if(dialog.DialogEvent.Equals("POST") && (dialog.State.Equals("ALERTING") || dialog.State.Equals("INITIATING"))) // Call Ringing Event

{

\_finAgent.FireLogMessage("New Call started : Call From: " + dialog.FromAddress + " and Call To: " + dialog.ToAddress + " and dialog status is : "+dialog.State);

isCallStart = true;

}

else if (dialog.DialogEvent.Equals("DELETE") && dialog.State.Equals("ACTIVE")) // Call Terminated and call will be Transferred

{

\_finAgent.FireLogMessage("Call Termination event as you released the call : Call From: " + dialog.FromAddress + " and Call To: " + dialog.ToAddress);

isCallEnd = true;

}

else if(dialog.DialogEvent.Equals("DELETE") && dialog.State.Equals("DROPPED")) // Call Terminated and caller hangup

{

\_finAgent.FireLogMessage("Call Termination event and caller dropped the call : Call From: "+dialog.FromAddress+" and Call To: "+dialog.ToAddress);

isCallEnd = true;

}

else if (dialog.DialogEvent.Equals("RunningCall") && !dialog.State.Equals("DROPPED") && !dialog.State.Equals("FAILED")) // We found running call

{

\_finAgent.FireLogMessage("Call running event and call still active : Call From: " + dialog.FromAddress + " and Call To: " + dialog.ToAddress);

isCallStart = true;

}

} else

{

foreach(Dialog.Participant participant in dialog.Participants)

{

if(participant.Me) //Checking my status

{

if (participant.State.Equals("DROPPED")) // call is not active , and this is call terminate event

{

\_finAgent.FireLogMessage("We received message event without event discription. Seems system just started while call was active, we will terminate the call without firing end event");

}

else if (participant.State.Equals("INITIATING"))

{

\_finAgent.FireLogMessage("We received message event without event discription. Seems system just started while call is active, your status is: "+participant.State);

}

}

}

}

if(isCallEnd)

{

new Thread(delegate ()

{

System.Windows.Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.ApplicationIdle, new System.Action(() =>

{

}));

}).Start();

return;

}

if (isCallStart)

{

new Thread(delegate ()

{

System.Windows.Application.Current.Dispatcher.Invoke(DispatcherPriority.ApplicationIdle, new System.Action(() =>

{

}));

}).Start();

}

}

# Build your first application (CSharp C#)

Start a new WPF project on Visual Studio. Then add the project reference as defined in the section above.

In the main C# file which is linked to the mainWindow.XAML, please implement the interface FinView. Also you will need to add the following variable:

FinAgent finAgent = new finAgent(); as a class variable.

By adding this line, you have initialized the main finesse agent object which will be used in binding.

In the same file and before the Component Initialization “InitializeComponent” is invoked , please set the DataContext to the finAgent object reference. Also you will need to set the finAgent.FinView to your current window view. The object of setting the FinView is to receive the events later on.

In the main XAML file, please add the following fields

* Login Grid
* Agent ID text field
* Password field
* Extension field
* Domain A
* Domain B
* Login Button
* Main Grid
* Status Combo-Box
* Dialog List
* Loading Grid
* Label showing the progress.

Bind the Agent ID field to the AgentInformation.AgentID. Repeat the same step for extension considering the variable name is AgentInformation.Extension.

Repeat the same steps for Domain A, Domain B. ***­­Please note that password field is not blindable.***

Now, you will need to add Login\_Click implementation. In the login click, you will need to set the AgentInformation.Password from the password field. After setting the password, you may call finAgent.SignIn in order to start the sign-in process.

Once system is logged in an event will be fired at the function FireReLoginEvent(). At this function you may need to switch your view to the main Grid.

You need to bind the main Grid as the following

1. Call Dialogs are binded to Dialog List as the below sample

<ListBox ItemsSource="{Binding Path=AgentInformation.Dialogs}" SelectedValue="{Binding Path=AgentInformation.SelectedDialog,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" SelectionChanged="SelectionChanged">

<ListBox.ItemTemplate>

<DataTemplate>

<WrapPanel>

<TextBlock Text="(" Foreground="Black" Margin="0,5"/>

<Label Content="{Binding Path=DialogStateTimer.TimerLabel}"/>

<TextBlock Text=")" Foreground="Black" Margin="0,5"/>

<Label Content="{Binding Header}"/>

<Button Tag="{Binding ID}" ToolTip="Answer" Name="Answer" Visibility="{Binding Answer, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/answercall.png" Width="25" Height="25" />

</Button>

<Button Tag="{Binding ID}" ToolTip="Direct Transfer" Name="Transfer" Visibility="{Binding Transfer, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/transfercall.png" Width="25" Height="25"/>

</Button>

<Button Tag="{Binding ID}" ToolTip="Transfer" Name="CTransfer" Visibility="{Binding CTransfer, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/transfercall.png" Width="25" Height="25"/>

</Button>

<Button Tag="{Binding ID}" ToolTip="Consult" Name="Consult" Visibility="{Binding Consult, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/consultcall.png" Width="25" Height="25"/>

</Button>

<Button Tag="{Binding ID}" ToolTip="Conference" Name="Conference" Visibility="{Binding Conference, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/conferencecall.png" Width="25" Height="25"/>

</Button>

<Button Tag="{Binding ID}" ToolTip="Keypad" Name="Keypad" Visibility="{Binding SendDTMF, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/keypadcall.png" Width="25" Height="25"/>

</Button>

<Button Tag="{Binding ID}" ToolTip="Hold" Name="Hold" Visibility="{Binding Hold, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/holdcall.png" Width="25" Height="25"/>

</Button>

<Button Tag="{Binding ID}" ToolTip="Resume" Name="Resume" Visibility="{Binding Resume, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/resumecall.png" Width="25" Height="25"/>

</Button>

<Button Tag="{Binding ID}" ToolTip="Release" Name="Release" Visibility="{Binding Release, Converter={StaticResource BoolToVisConverter}}" Click="CallButton\_Click">

<Image Source="Images/releasecall.png" Width="25" Height="25"/>

</Button>

</WrapPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

1. You will need to bind the status combo box as the below sample

<ComboBox Name="StatusComboBox" ItemsSource="{Binding Path=AgentInformation.VoiceStatusList}" Margin="10,2" VerticalAlignment="Center" VerticalContentAlignment="Center" SelectedValue="{Binding Path=AgentInformation.SelectedVoiceStatus,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" SelectionChanged="VoiceStatus\_SelectionChanged">

<ComboBox.Resources>

<Style TargetType="ComboBox">

<Setter Property="Foreground" Value="#0075be" />

<Setter Property="BorderBrush" Value="#0075be" />

<Setter Property="Background" Value="White" />

<Setter Property="SnapsToDevicePixels" Value="true"/>

<Setter Property="OverridesDefaultStyle" Value="true"/>

<Setter Property="ScrollViewer.HorizontalScrollBarVisibility" Value="Auto"/>

<Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Auto"/>

<Setter Property="ScrollViewer.CanContentScroll" Value="true"/>

<Setter Property="FontWeight" Value="Normal" />

<Setter Property="MinWidth" Value="300"/>

<Setter Property="MaxHeight" Value="32"/>

<Setter Property="MinHeight" Value="32"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="ComboBox">

<Grid>

<ToggleButton

Name="ToggleButton"

BorderBrush="{TemplateBinding BorderBrush}"

Background="{TemplateBinding Background}"

Foreground="{TemplateBinding Foreground}"

Style="{StaticResource ComboBoxToggleButton}"

Grid.Column="2"

Focusable="false"

IsChecked="{Binding Path=IsDropDownOpen,Mode=TwoWay,RelativeSource={RelativeSource TemplatedParent}}"

ClickMode="Press">

</ToggleButton>

<ContentPresenter

Name="ContentSite"

IsHitTestVisible="False"

Content="{TemplateBinding SelectionBoxItem}"

ContentTemplate="{TemplateBinding SelectionBoxItemTemplate}"

ContentTemplateSelector="{TemplateBinding ItemTemplateSelector}"

Margin="10,3,30,3"

VerticalAlignment="Center"

HorizontalAlignment="Left" />

<TextBox x:Name="PART\_EditableTextBox"

Style="{x:Null}"

Template="{StaticResource ComboBoxTextBox}"

HorizontalAlignment="Left"

VerticalAlignment="Center"

Margin="3,3,23,3"

Focusable="True"

Visibility="Hidden"

IsReadOnly="{TemplateBinding IsReadOnly}"/>

<Popup

Name="Popup"

Placement="Bottom"

IsOpen="{TemplateBinding IsDropDownOpen}"

AllowsTransparency="True"

Focusable="False"

PopupAnimation="Slide">

<Grid

Name="DropDown"

SnapsToDevicePixels="True"

MinWidth="{TemplateBinding ActualWidth}"

MaxHeight="{TemplateBinding MaxDropDownHeight}">

<Border

x:Name="DropDownBorder"

Background="White"

BorderThickness="2"

BorderBrush="#0075be"/>

<ScrollViewer Margin="4,6,4,6" SnapsToDevicePixels="True">

<StackPanel IsItemsHost="True" KeyboardNavigation.DirectionalNavigation="Contained" />

</ScrollViewer>

</Grid>

</Popup>

</Grid>

<ControlTemplate.Triggers>

<Trigger Property="HasItems" Value="false">

<Setter TargetName="DropDownBorder" Property="MinHeight" Value="95"/>

</Trigger>

<Trigger Property="IsGrouping" Value="true">

<Setter Property="ScrollViewer.CanContentScroll" Value="false"/>

</Trigger>

<Trigger SourceName="Popup" Property="Popup.AllowsTransparency" Value="true">

<Setter TargetName="DropDownBorder" Property="CornerRadius" Value="0"/>

<Setter TargetName="DropDownBorder" Property="Margin" Value="0,2,0,0"/>

</Trigger>

<Trigger Property="IsEditable" Value="true">

<Setter Property="IsTabStop" Value="false"/>

<Setter TargetName="PART\_EditableTextBox" Property="Visibility" Value="Visible"/>

<Setter TargetName="ContentSite" Property="Visibility" Value="Hidden"/>

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

<Style.Triggers>

</Style.Triggers>

</Style>

</ComboBox.Resources>

<ComboBox.ItemTemplate>

<DataTemplate>

<WrapPanel>

<Image Source="{Binding StatusImage}" Width="22"/>

<TextBlock Text="{Binding StatusLabel}"/>

</WrapPanel>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

1. Next step is implementing the call actions
   1. Answer Click should call finAgent.AnswerCall
   2. Release Click should call finAgent.ReleaseCall
   3. Rest of functions are implementation with similar names
      1. HoldCall
      2. ResumeCall
      3. SSTransferCall
      4. TransferCall
      5. ConsultCall
      6. ConferenceCall
      7. SendDTMFCall
2. Last Step is the implementation of status change as the following sample

if (StatusComboBox.SelectedItem == null)

return;

\_finAgent.ChangeAgentVoiceStatus(StatusComboBox.SelectedItem as VoiceStatus);

# Setup Library Object

FinAgent(string \_agentID, string \_password, string \_extension, string \_domainA, string \_domainB, string \_traceStatus, IFinAgentView UI)

To create a new finesse agent object, you are required to define the following parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| \_agentID | String | The Agent ID will be logged in |
| \_password | String | The Password will be used by the agent |
| \_extension | String | The Agent Extension |
| \_domainA | String | The hostname / ip of finesse side A |
| \_domainB | String | The hostname / ip of finesse side B |
| \_traceStatus | String | “enabled / disabled” in order to enable the traces in USD debug window |
| UI | IFinAgentView | The view object instance of the view |

# Login function

public bool SignIn()

In order to sign in, you have to call SignIn function after creating the FinAgent Object.

Function will return whether true or false based on sign in outcome.

# Logout function

public bool SignOut(string status, string reasonCodeLabel)

In order to sign out, you have to call Signout function after creating the FinAgent Object and agent is signed on.

Function will return whether true or false based on sign out outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| Status | String | The new status which is “LOGOUT” |
| reasonCodeLabel | String | The reason code label selected by the end user. |

# Answer Call function

public bool AnswerCall(string dialogID)

In order to Answer a call, you have to call AnswerCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID to be answered |

# Release Call function

public bool ReleaseCall(string dialogID)

In order to Release a call, you have to call ReleaseCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID to be answered |

# Hold Call function

public bool HoldCall(string dialogID)

In order to put a call on hold, you have to call HoldCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID to be answered |

# Resume Call function

public bool ResumeCall(string dialogID)

In order to retrieve a call from hold, you have to call ResumeCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID to be answered |

# Signle Step Call Transfer function

public bool SSTransferCall(string dialogID, string dialedNumber)

In order to transfer a call on single step, you have to call SSTransferCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID to be answered |
| dialedNumber | String | The target phone number will be used for call transfer |

# Consult Call function

public bool ConsultCall(string dialogID, string dialedNumber)

In order to make a consult call, you have to call ConsultCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID to be answered |
| dialedNumber | String | The target phone number will be used for call transfer |

# Conference Call function

public bool ConferenceCall(string dialogID, string dialedNumber)

In order to make a conference call, you have to call ConferenceCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID to be answered |
| dialedNumber | String | The target phone number will be used for call transfer |

# make Call function

public bool MakeCall(string dialedNumber)

In order to make call, you have to call MakeCall function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialedNumber | String | The extension will be dialed |

# Update Call data function

public bool UpdateCallData(string dialogID, Dictionary<string, string> callVariables, string wrapupReason)

In order to update call data, you have to call UpdateCallData function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| dialogID | String | The call dialog ID |
| callVariables | Dictionary | The call variables map representation the new data |
| wrapupReason | String | The wrapup reason |

# CHANGE status function

public bool ChangeStatus(string status, string reasonCodeLabel)

In order to change agent status, you have to call ChangeStatus function.

Function will return whether true or false based on outcome.

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| Status | String | The new agent status |
| reasonCodeLabel | String | The reason code in case of Logout or not Ready |
| wrapupReason | String | The wrapup reason |

# Events Implementation

In order to receive Finesse Server events , the following functions should be implemented in your View side or according to the design model

# New Event Function

public void FireNewEvent()

Fire New Event is the method receiving the Finesse Events. The view side has to implement this function in order to receive incoming event. The event could be one of the following types

1. User Message
2. Call Message
3. Error Message

The FinAgent Model is handling the agent information through a main class of agent information. In the agent information, the message event is an instance representing the new received message.

Below sample code of FireNewEvent Implementation. Each event received on XMPP will be sent through this function to the front end.

{

if (finAgent.\_agentInformation.\_MessageEvent == null)

return;

if (finAgent.\_agentInformation.\_MessageEvent.MessageType == null)

return;

if (finAgent.\_agentInformation.\_MessageEvent.MessageType.Equals("user"))

{

//PopulateStatusDropDown();

//AdjustScreen();

}

else if (finAgent.\_agentInformation.\_MessageEvent.MessageType.Equals("call"))

//PopulateCallInformation()));

else if (finAgent.\_agentInformation.\_MessageEvent.MessageType.Equals("error"))

{

//PopulateErrorMessage(null);

//PopulateCallInformation();

}

}

# Error Message Function

public void FireErrorMessage(string msg)

Fire Error message function has to be implemented also through the front end. This method is called by the model in case of any error occurred during any activity.

Below is sample implementation.

{

if (msg != null)

{

try

{

//PopulateErrorMessage(msg);

//PopulateCallInformation()));

}

catch (Exception)

{

}

}

}

# Loading Messages Events

public void FireLoadingMessage(string msg)

Fire loading message is a function has to be implemented by the front end. The function is receiving the loading messages during the sign on. It should be used for during the signin process.

Below is sample implementation.

{

if (msg != null)

{

try

{

//PopulateLoadingMessage(msg);

}

catch (Exception)

{

}

}

}

# Relogin Message Event

public void FireReLoginEvent()

Front end has to implement the function FireReLoginEvent. The function will be called after the agent getting connected after many connection trials.

Below is sample implementation of the FireReloginEvent.

{

try

{

//MessageBar.Visibility = Visibility.Hidden;

//LoginMessageBar.Visibility = Visibility.Hidden;

//finAgent.LoadAgentInformation(null);

//if(finAgent.\_agentInformation.Dialogs != null)

// finAgent.\_agentInformation.Dialogs.Clear();

//finAgent.LoadCallInformation();

//PopulateAgentInformation();

//PopulateCallInformation();

//PopulateStatusDropDown();

}

catch (Exception)

{

}

}

# Fire Load Login Screen Message Event

public void FireLoadLoginScreen()

Front end has to implement the function FireLoadLoginScreen. The function will be called in case agent failed to sign in.

Below is sample implementation of the FireReloginEvent.

{

LoadScreen(ScreenName.LoginScreen);

}

# Fire USD Internal Event

public void FireCallEvent(Dialog dialog)

Front end has to implement the function FireCallEvent. The function will be called in cased of any call message is received through XMPP. The Dialog messages are coming as updates. So each update is fired through a new dialog object. The dialog Object is handing all the call details such as call variables .. etc