Workspace Desktop Edition Developer's Guide

Customize Views and Regions

4/16/2025

Customize Views and Regions

Purpose: To provide information about customizable views and their regions.

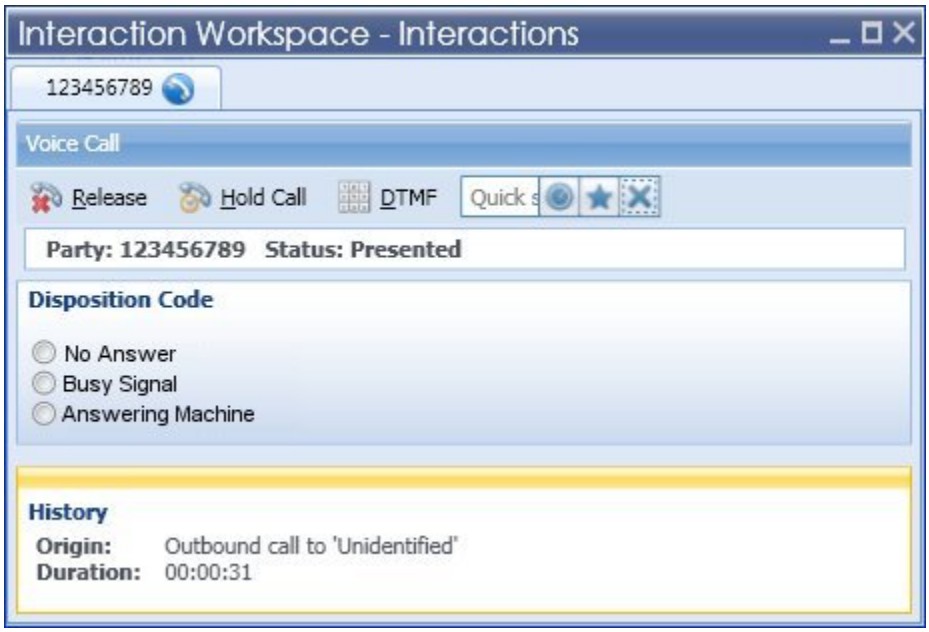
Contents

Before You Start

- All the code snippets in this page are extracted from the Genesyslab.Desktop.Modules.ExtensionSample source files. See About the Extension Samples for additional information about the samples.
- In addition to this page, read:
  - Creating a New Module
  - Deploying Your Custom Module into the Genesys Out-Of-The-Box Application

Replacing an Existing View

There are several ways to customize Workspace Desktop Edition. The most basic way is to change an existing behavior or appearance by changing the implementation of an existing interface. The code that is displayed after the figure demonstrates how to replace an existing view, `DispositionCodeView`, with the new view, `DispositionCodeExView`. You can replace the existing view with another by associating the existing `IDispositionCodeView` interface with the new `DispositionCodeExView` implementation.



Voice Interactions View Before Customization. The out-of-the-box application uses radio buttons in the Disposition Code View. The code sample that is displayed after the figure modifies this view.

1. Start by creating a new Windows Presentation Foundation (WPF) UserControl that is composed of the following two files:
   - `DispositionCodeExView.xaml`

- DispositionCodeExView.xaml.cs

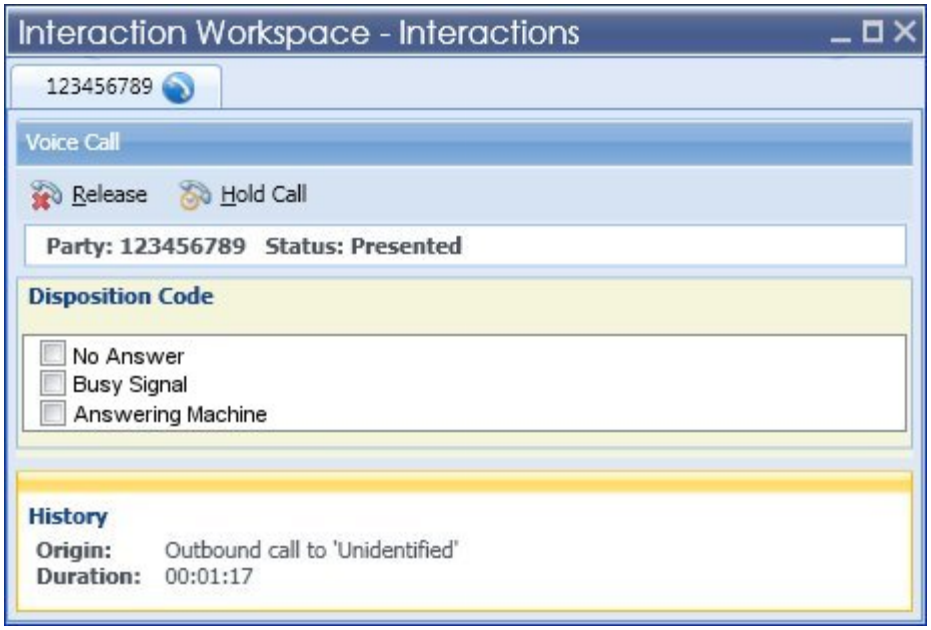
2. Ensure that your custom-built view named, DispositionCodeExView implements the genuine IDispositionCodeView interface:
[C#]

```
// File: DispositionCodeExView.cs
public partial class DispositionCodeExView : UserControl, IDispositionCodeView
{
  public DispositionCodeExView(IDispositionCodeViewModel dispositionCodeViewModel)
  {
    this.viewManager = viewManager;
    this.Model = dispositionCodePresentationModel;
    InitializeComponent();
    Width = Double.NaN;
    Height = Double.NaN;
  }
  #region IDispositionCodeView Members
  public IDispositionCodeViewModel Model
  {
    get { return this.DataContext as IDispositionCodeViewModel; }
    set { this.DataContext = value; }
  }
  #endregion
  ...
}
```

3. Register the new view in your module to make it replace the former view when the module is loaded. Do this by calling the IObjectContainer.RegisterType method must be used to register the new implementation in the initialization method of the ExtensionSampleModule:
[C#]

```
// File: ExtensionSampleModule.cs
public class ExtensionSampleModule : IModule
{
  readonly IObjectContainer container;
  ...
  public void Initialize()
  {
    container.RegisterType<IDispositionCodeView, DispositionCodeExView>();
  }
}
```

You can replace any view with your own custom-built view by using the preceding examples. The figure below shows the view for Workspace Desktop Edition Voice Interactions after customization. In the customized view, the radio buttons for disposition codes are replaced with check boxes.



Disposition Code View After Customization

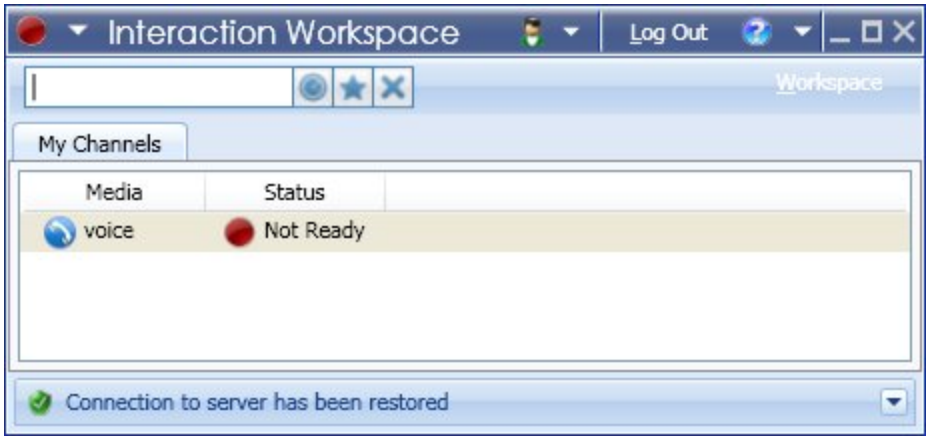
> **Important**
>
> Although the application has a different appearance, it retains its previous behavior.

Creating a New View

Advanced customization provides the IViewManager interface to add a new view to an existing region (which is embedded in an existing view). Regions which embed multiple views, tabs, or buttons, can be enriched with new views. To use the Model-View-ViewModel (MVVM) pattern, you must create both the view (for instance, `MySampleView`) which extends the IView interface and the view-model, for instance `MySampleViewModel`. The following subsections detail the creation for two new views through the customization samples.

Adding a Tab to the ToolbarWorkplaceRegion

The Genesyslab.Desktop.Modules.ExtensionSample example creates a new view in the Voice Interaction panel. In the following figure, the out-of-the-box application has a single tab called `My Channels`, which is part of the `ToolbarWorkplaceRegion` region. The customization adds a new tab called `My Sample Header` which contains a button and a time counter.



Voice Interactions View before customization. A single tab 'My Channels' is available in the ToolbarWorkplaceRegion

The following steps details the content of the Genesyslab.Desktop.Modules.ExtensionSample.

1. To create the view-model, create a new interface named `IMySampleViewModel` which manages a time counter and the header label for the new tab:
   [C#]

```
// File: IMySamplePresentationModel.cs
public interface IMySampleViewModel
{
  string Header { get; set; }
  TimeSpan Counter { get; set; }
  void ResetCounter();
}
```

2. Implement this interface by creating the `MySampleViewModel` class, as follows:
   [C#]

```
// File: MySamplePresentationModel.cs
public class MySampleViewModel : IMySampleViewModel, INotifyPropertyChanged
{
  // Field variables
  string header = "My Sample Header";
  TimeSpan counter = TimeSpan.Zero;
  public MySampleViewModel()
  {
    // Start the counter timer
    DispatcherTimer dispatcherTimer = new DispatcherTimer();
    dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
    dispatcherTimer.Tick += new EventHandler(delegate(object sender, EventArgs e)
      {
        Counter += TimeSpan.FromSeconds(1.0);
      });
    dispatcherTimer.Start();
  }
  #region IMySamplePresentationModel Members
  public string Header
  {
    get { return header; }
    set { if (header != value) { header = value; OnPropertyChanged("Header"); } }
  }
  public TimeSpan Counter
  {
```

```
    get { return counter; }
    set { if (counter != value) { counter = value; OnPropertyChanged("Counter"); } }
  }
  public void ResetCounter()
  {
    Counter = TimeSpan.Zero;
  }
  #endregion
  #region INotifyPropertyChanged Members
  public event PropertyChangedEventHandler PropertyChanged;
  protected void OnPropertyChanged(string propertyName)
  {
    if (PropertyChanged != null)
      PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
  }
  #endregion
}
```

3. Create the the view interface you want to implement, named `IMySampleView`, for your built-in component:
   [C#]

```
// File: IMySampleView.cs
public interface IMySampleView : IView
{
  IMySampleViewModel Model { get; set; }
}
```

4. Create a content for the new view with a new WPF UserControl that is composed of the following two files:

   - MySampleView.xaml

   [XAML]

```
<UserControl x:Class="Genesyslab.Desktop.Modules.ExtensionSample.MySample.MySampleView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Height="220" Width="279" MinHeight="90">
  <Grid>
    <Ellipse Margin="12" Name="ellipse1" Stroke="Black" />
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
      <Button Click="Button_Click">Button</Button>
      <TextBlock Text="{Binding Counter}"/>
    </StackPanel>
  </Grid>
</UserControl>
```

   - MySampleView.xaml.cs which contains your custom-built class named `MySampleView` implementing the IMySampleView interface created previously:

   [C#]

```
// File: MySampleView.xaml.cs
public partial class MySampleView : UserControl, IMySampleView
{
  public MySampleView(IMySampleViewModel mySampleViewModel)
  {
    this.Model = mySampleViewModel;
    InitializeComponent();
    Width = Double.NaN;
    Height = Double.NaN;
  }
```

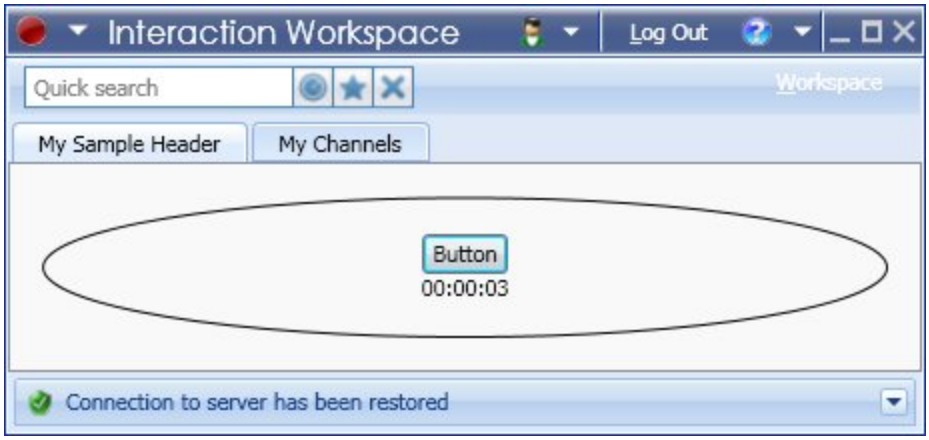
```
  #region IMySampleView Members

  public IMySampleViewModel Model
  {
    get { return this.DataContext as IMySampleViewModel; }
    set { this.DataContext = value; }
  }
  #endregion
  #region IView Members
  public object Context { get; set; }
  public void Create()
  {
  }
  public void Destroy()
  {
  }
  #endregion
  private void Button_Click(object sender, System.Windows.RoutedEventArgs e)
  {
    Model.ResetCounter();
  }
}
```

5. In the Initialize() method of your module (see Creating a New Module):
   - Register your new view and model with the `IObjectContainer.RegisterType` method.
   - Insert the view in the appropriate view or region.

   [C#]

```
// File: ExtensionSampleModule.cs
public class ExtensionSampleModule : IModule
{
  public void Initialize()
  {
    container.RegisterType<IMySampleView, MySampleView>();
    container.RegisterType<IMySampleViewModel, MySampleViewModel>();
    // Add the MySample view to the region "ToolbarWorkplaceRegion" (The TabControl in
the main toolbar)
    viewManager.ViewsByRegionName["ToolbarWorkplaceRegion"].Insert(0,
            new ViewActivator() { ViewType = typeof(IMySampleView), ViewName =
"MySample" });
    ...
  }
}
```

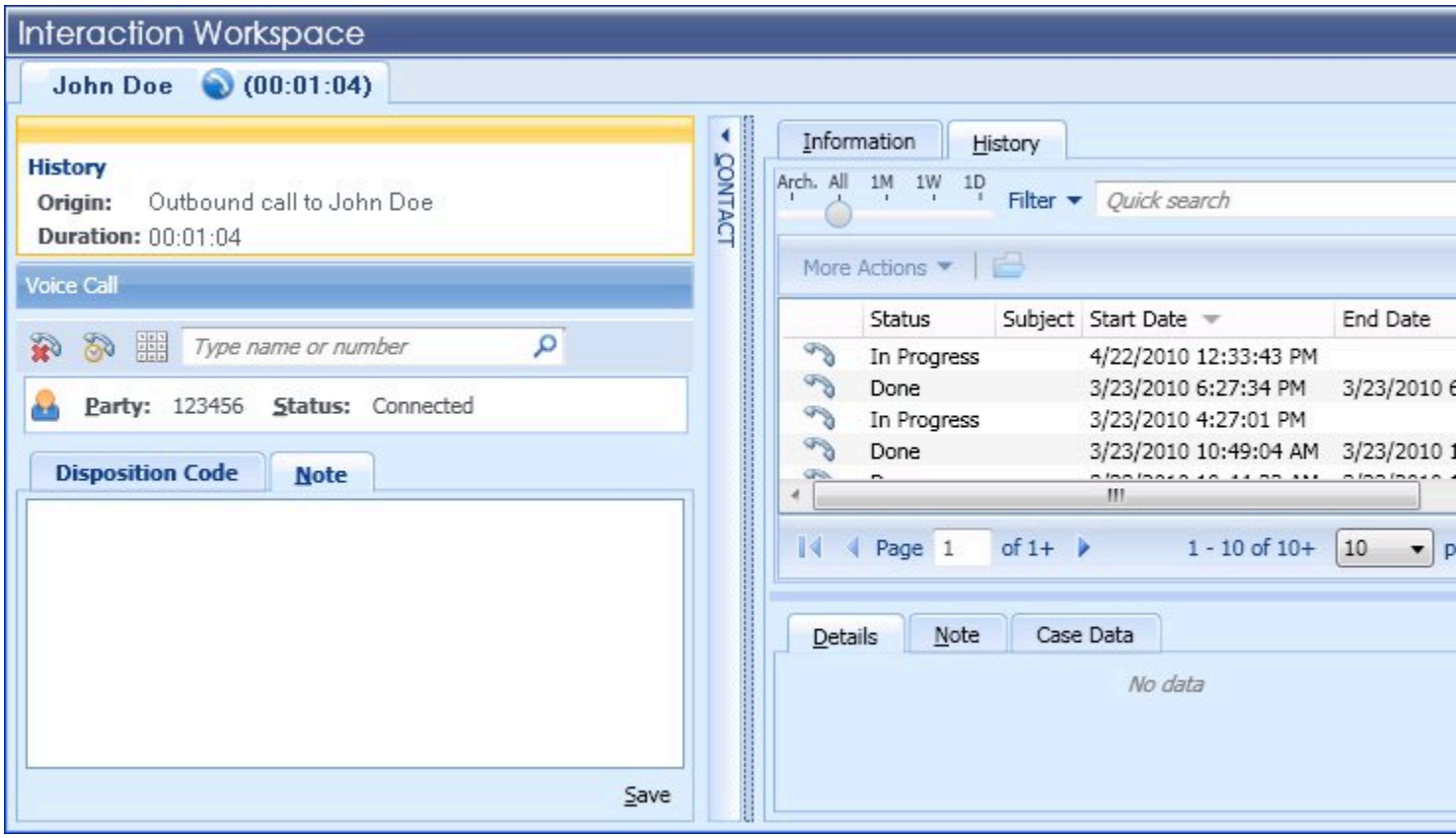
The figure below depicts the `MySampleView` after customization with the second tab (`My Sample Header`) included in the view. In the following example, the `ToolbarWorkplaceRegion` of the view is modified. For a complete list of views and regions, see How to Customize Views and Their Regions Reference for Windows.



Voice Interactions View After Customization: A new tab 'MySampleHeader' is available.

Adding a View to the Interaction Window

Similar to the Genesyslab.Desktop.Modules.ExtensionSample is the Genesyslab.Desktop.Modules.InteractionExtensionSample which adds a view to the right panel of the Interaction Window.



The Interaction Window before customization. In the middle bar which separates the windows in two parts, a "CONTACT" expand bu displays the current contact view to the right side of the window.

1. Create a new interface named IMySampleViewModel.
   The view model implemented includes a case for the interaction management, in addition to the counter and the header.
   [C#]

```
// File: IMySamplePresentationModel.cs
namespace Genesyslab.Desktop.Modules.InteractionExtensionSample.MySample
{
  public interface IMySampleViewModel
  {
    string Header { get; set; }
    TimeSpan Counter { get; set; }
    ICase Case { get; set; }
    void ResetCounter();
  }
}
```

2. the implementation of the interface includes the management of the case.
   [C#]

```
// File: MySamplePresentationModel.cs
namespace Genesyslab.Desktop.Modules.InteractionExtensionSample.MySample
{
  public class MySampleViewModel : IMySampleViewModel, INotifyPropertyChanged
  {
    // Field variables
    string header = "My Sample Header";
    TimeSpan counter = TimeSpan.Zero;
    ICase @case;
    public MySampleViewModel()
    {
      // Start the counter timer
      DispatcherTimer dispatcherTimer = new DispatcherTimer();
      dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
      dispatcherTimer.Tick += new EventHandler(delegate(object sender, EventArgs e)
        {
          Counter += TimeSpan.FromSeconds(1.0);
        });
      dispatcherTimer.Start();
    }
    #region IMySamplePresentationModel Members

        ...

        public ICase Case
    {
      get { return @case; }
      set { if (@case != value) { @case = value; OnPropertyChanged("Case"); } }
    }

    #endregion
    #region INotifyPropertyChanged Members
        ...
    #endregion
  }
}
```

3. Then, you create the view interfaces you want to implement, named IMySampleView and IMySampleButtonView.cs, for your built-in components:
   [C#]

```
// File: IMySampleButtonView.cs
namespace Genesyslab.Desktop.Modules.InteractionExtensionSample.MySample
```

```
public interface IMySampleButtonView : IView
{
  IMySampleViewModel Model { get; set; }
}
// File: IMySampleView.cs
namespace Genesyslab.Desktop.Modules.InteractionExtensionSample.MySample
{
  // Interface matching the MySampleView view
  public interface IMySampleView : IView
  {
    // Gets or sets the model.
    IMySampleViewModel Model { get; set; }
  }
}
```

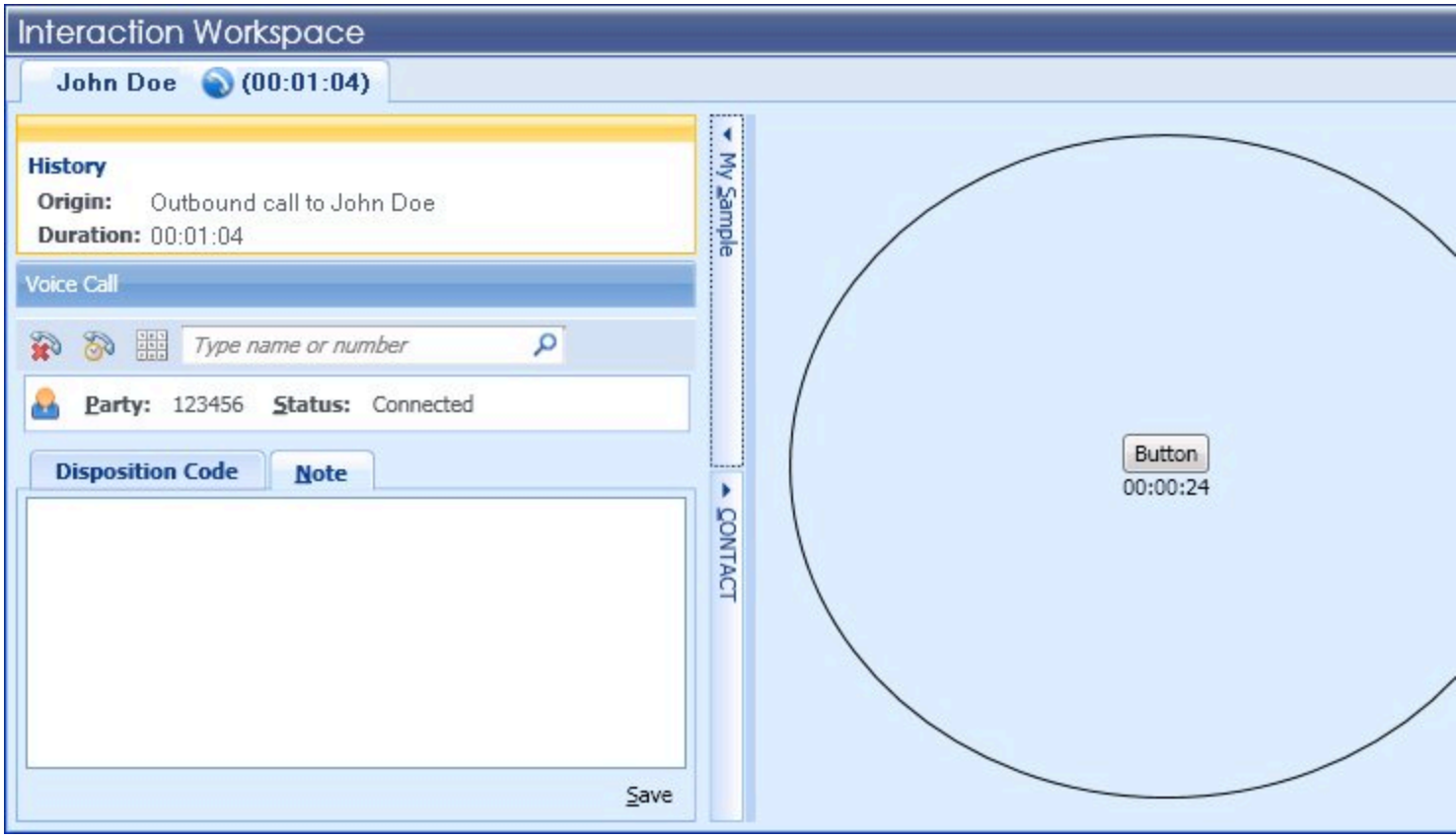
4. Create a content for the new view with a new WPF UserControl that is composed of the following two files:
   - MySampleButtonView.xaml
   - MySampleButtonView.xaml.cs

   See the files in the InteractionExtensionSample.
5. In the Initialize() method of your module (see Creating a New Module):
   - Register your views and models with the IObjectContainer.RegisterType method.
   - Insert the views in the appropriate view or region, as shown here:

[C#]

```
// File: InteractionExtensionSampleModule.cs
public void Initialize()
{
  // Add a view in the right panel in the interaction window
  // Here we register the view (GUI) "IMySampleView" and its behavior counterpart
"IMySampleViewModel"
  container.RegisterType<IMySampleView, MySampleView>();
  container.RegisterType<IMySampleViewModel, MySampleViewModel>();
  // Put the MySample view in the region "InteractionWorksheetRegion"
  viewManager.ViewsByRegionName["InteractionWorksheetRegion"].Add(
    new ViewActivator() { ViewType = typeof(IMySampleView),
              ViewName = "MyInteractionSample", ActivateView = true }
  );
  // Here we register the view (GUI) "IMySampleButtonView"
  container.RegisterType<IMySampleButtonView, MySampleButtonView>();
  // Put the MySampleMenuView view in the region "CaseViewSideButtonRegion"
        // (The case toggle button in the interaction windows)
  viewManager.ViewsByRegionName["CaseViewSideButtonRegion"].Add(
    new ViewActivator() { ViewType = typeof(IMySampleButtonView),
              ViewName = "MySampleButtonView", ActivateView = true }
  );
}
```

The figure below depicts the Interaction window after customization.



The Interaction Window after customization. In the middle bar which separates the windows in two parts, a "MySample" expand bu displays the sample view.

Hiding and Showing Custom Views

Available since: 8.1.100.14

You can display a custom view according to specific parameters by using the concept of conditions. To do this, include a `Condition` when you register your view with the `IViewManager`. This condition will be executed each time that the application framework instantiates the region. For instance, if you wish to change the displayed tabs in the interaction window based on the context of the displayed interaction, then you can include a condition when adding your customized `IMySampleView` to the `InteractionDetailsRegion` region:

[C#]

```
viewManager.ViewsByRegionName["InteractionDetailsRegion"].Add(new ViewActivator() {
        ViewType = typeof(IMySampleView), ViewName =   "MyInteractionSample", ActivateView =
true,
        Condition = MySampleViewModel.MySampleViewModelCondition});
```

Next, implement this condition somewhere in your code. In the following example, this method returns `true` to show the custom view or `false` to hide it:

[C#]

```
        public static bool MySampleViewModelCondition(ref object context)
        {
            IDictionary<string, object> contextDictionary = context as IDictionary<string,
object>;
            object caseView;
            contextDictionary.TryGetValue("CaseView", out caseView);
            object caseObject;
            contextDictionary.TryGetValue("Case", out caseObject);
            ICase @case = caseObject as ICase;
            if (@case != null)
            {
                if (@case.MainInteraction != null)
                {
                    IInteraction i = @case.MainInteraction;
                    return (i.HasBeenPresentedIn);
                }
            }
            return false;
        }
```