Sending Data Between Views

Since both views are separate, how do we send the data from one view to other? Traditionally this can be made possible using Event handling mechanism.

The answer is to use the **MVVM Light Messenger**. This messenger provides a loosely-bound way to send message (data) from one ViewModel to other.

Diagrammatically the Messenger can be explained as below:

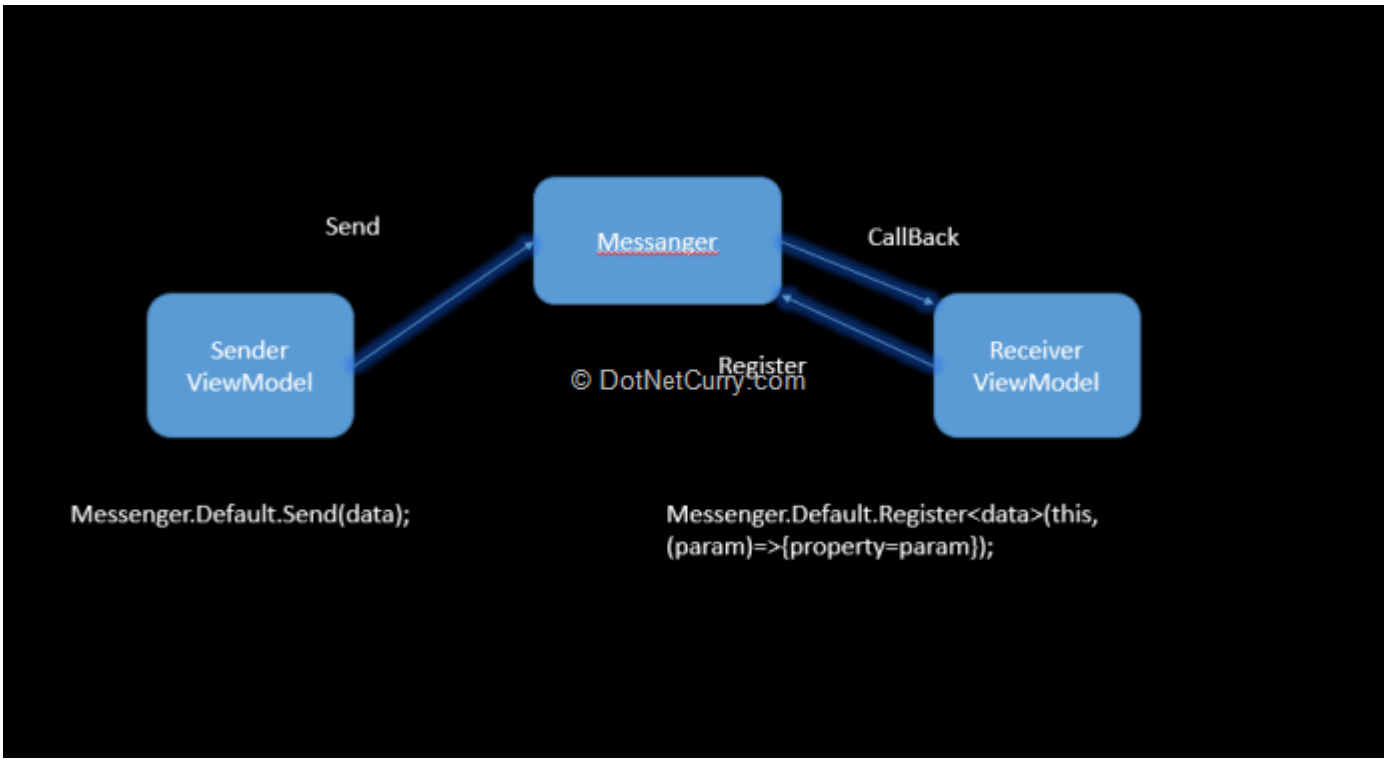

Figure 14: MVVM Light Messenger

Messenger is a singleton object which lives throughout the application. The sender ViewModel simply calls the static 'Send' method. The receiver ViewModel needs to register with the messenger to receive the object. It provides a call back function which gets called when the new message received. Let's see how this is done.

Step 1: In the project add a new folder and name it 'MessageInfrastructure'. In this folder, add a new class file:

```
using WPF_MVVMLight_CRUD.Model;

namespace WPF_MVVMLight_CRUD.MessageInfrastructure
{
    public class MessageCommunicator
```

```
    {
        public EmployeeInfo Emp { get; set; }
    }
}
```

The above class defines *Emp* property of the type EmployeeInfo. This will be used as a message (data) passed from one view to other.

Step 2: In the MainViewModel add the following method:

```
void SendEmployeeInfo(EmployeeInfo emp)
{
    if(emp!=null)
    {
        Messenger.Default.Send<messagecommunicator>(new MessageCommunicator()
{
          Emp = emp
        });
    }
}
```

Note: Please add the 'GalaSoft.MvvmLight.Messaging' namespace in the MainViewModel.

The above method accepts an EmployeeInfo object and calls the Send() method of the Messenger, which is typed to the MessageCommunicator class. This means that the View which calls the above method, must pass the EmployeeInfo object.

In the MainViewModel define the following *RelayCommand* object:

```
public RelayCommand<employeeinfo> SendEmployeeCommand { get; set; }
</employeeinfo>
```

The RelayCommand is defined with the parameter of the type EmployeeInfo. This means that it will execute method having input parameter of the type EmployeeInfo.

In the constructor of the ViewModel, define an instance of the RelayCommand as shown here:

```
SendEmployeeCommand = new
RelayCommand<employeeinfo>(SendEmployeeInfo);</employeeinfo>
```

Step 2: Open the EmployeeInfoView and define the EventToCommand for the DataGrid. Since the DataGrid is bound with the EmployeeInfo collection, when the DataGridRow is selected, it will select the EmployeeInfo object. We will map the SelectionChanged event of the DataGrid to the EventToCommand as shown here:

```
<UserControl x:Class="WPF_MVVMLight_CRUD.Views.EmployeeInfoView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
             xmlns:mvvm="http://www.galasoft.ch/mvvmlight"
        xmlns:ignore="http://www.ignore.com"
        mc:Ignorable="d ignore"
        DataContext="{Binding Main, Source={StaticResource Locator}}"
Height="446.866" Width="617.015">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="51*"/>
            <RowDefinition Height="42*"/>
            <RowDefinition Height="283*"/>
            <RowDefinition Height="71*"/>
        </Grid.RowDefinitions>
        <TextBlock TextWrapping="Wrap"
                    TextAlignment="Center" Text="List of All Employees" FontWeight="Bold"
FontSize="30"/>
        <Button x:Name="btnloadallemployees" Content="List All Employees"
                Grid.Row="3" FontSize="30"
                FontWeight="Bold"
                 Command="{Binding ReadAllCommand}"/>
        <DataGrid x:Name="dgemp" Grid.Row="2" ItemsSource="{Binding Employees}"
                 ColumnWidth="*"  Margin="0,10,0,28" RenderTransformOrigin="0.5,0.5"
                   IsReadOnly="True" >
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="SelectionChanged">
                    <mvvm:EventToCommand
                    Command="{Binding  SendEmployeeCommand, Mode=OneWay}"
                    CommandParameter="{Binding ElementName=dgemp,Path=SelectedItem}"
                     />
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </DataGrid>
        <TextBlock HorizontalAlignment="Left" Margin="10,7,0,0" Grid.Row="1"
                   TextWrapping="Wrap" Text="EmpName to Search:" VerticalAlignment="Top"
                   Width="231"/>
        <TextBox HorizontalAlignment="Left" Height="30" Margin="262,7,0,0"
                 Grid.Row="1" TextWrapping="Wrap"  Text="{Binding EmpName,
UpdateSourceTrigger=PropertyChanged}"
                 VerticalAlignment="Top" Width="310">
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="TextChanged">
                    <mvvm:EventToCommand
                    Command="{Binding SearchCommand, Mode=OneWay}"
                     />
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </TextBox>
    </Grid>
</UserControl>
```

```
<datagrid grid.row='"2"' isreadonly='"True"' rendertransformorigin='"0.5,0.5"'
margin='"0,10,0,28"' columnwidth='"*"' employees}"="" itemssource='"{Binding'
x:name='"dgemp"'>
    <i:interaction.triggers>
        <i:eventtrigger eventname='"SelectionChanged"'>
            <mvvm:eventtocommand command='"{Binding'
commandparameter='"{Binding' mode='OneWay}"'
elementname='dgemp,Path=SelectedItem}"' sendemployeecommand,="">
        </mvvm:eventtocommand></i:eventtrigger>
    </i:interaction.triggers>
</datagrid>
```

The above XAML shows that the Command property is bound with the *SendEmployeeCommand* declared in the ViewModel. The parameter sent from the UI to ViewModel is the *SelectedItem*, which is an EmployeeInfo object. Since the SendEmployeeCommand executes the *SendEmployeeInfo* method, the EmployeeInfo object will be passed to this method.

Note

```
public class ViewModelLocator
    {
        /// <summary>
        /// Initializes a new instance of the ViewModelLocator class.
        /// </summary>
        public ViewModelLocator()
        {
            ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);


            SimpleIoc.Default.Register<MainViewModel>();
        }

        public MainViewModel Main
        {
            get
            {
                return ServiceLocator.Current.GetInstance<MainViewModel>();
            }
        }

        public static void Cleanup()
        {
            // TODO Clear the ViewModels
        }
    }
```

Step 3: Now we need to register for the messenger. To do so, in the MainViewModel add the following method:

```
void ReceiveEmployeeInfo()
{
```

```
    if (EmpInfo != null)
    {
        Messenger.Default.Register<messagecommunicator>(this,(emp)=>{
            this.EmpInfo = emp.Emp;
        });
    }
}
```

The above method registers to the messenger and accepts the Emp message received. This message is then set to the EmpInfo notifiable property defined in the ViewModel class. Call this method in the constructor of the MainViewModel. Since EmpInfo property is bound with the SaveEmployeeView, the Employee data will be displayed in it.

Step 4: Run the application, click on the Load All Employees button and the DataGrid will show all Employees. Select the Row from the DataGrid, the selected Employee Information will be displayed in the SaveEmployeeView as below: