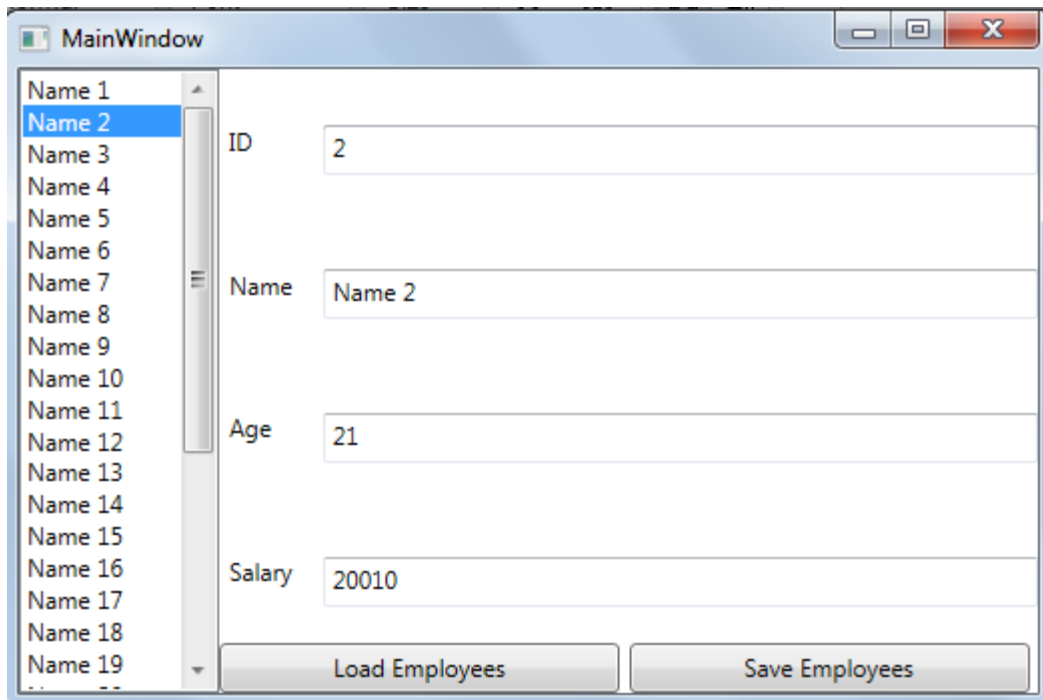


# Simple MVVM Light Toolkit Example.

In this example, we will create a sample application for loading and saving the employees details using MVVM Light Toolkit.

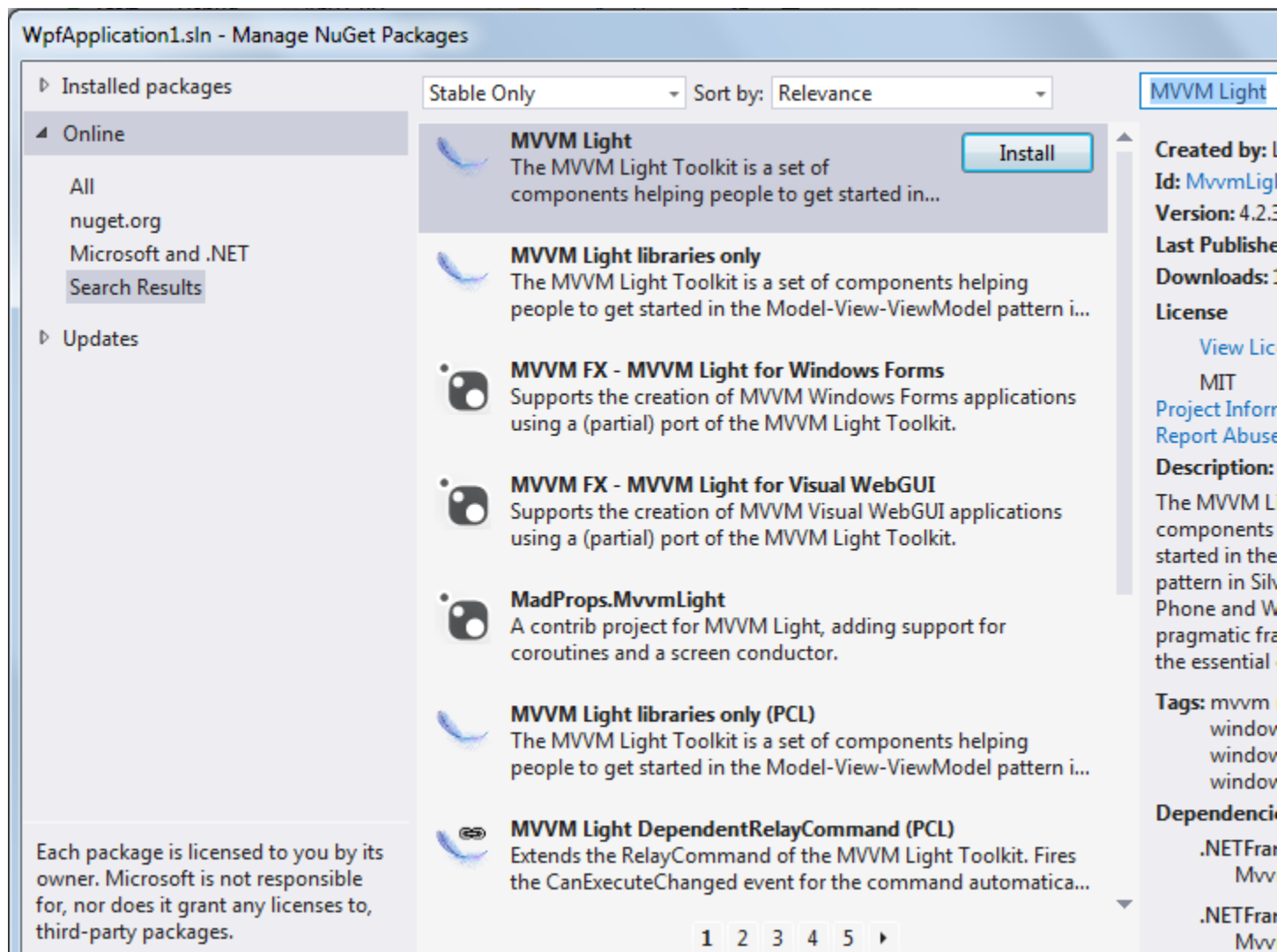


We will do below tasks one by one:

1. Add a reference of MVVM Light Toolkit into Visual Studio Project
2. Design UI of MainWindow.xaml
3. Create a Model class named Employee
4. Create a ViewModel class named MainViewModel
5. Create a ViewModelLocator class for binding the ViewModel classes to View classes
6. Build the project and test application

## Add reference of MVVM Light Toolkit

Open Visual Studio -> Go to Tools Menu-> Library Package Manager -> Manage NuGet Packages for Solution -> Click on Install.



This will install MVVM Light Toolkit into your project.

## Design UI of MainWindow.xaml

For designing the above UI, I have taken a DockPanel as parent control. In the left side, we have ListBox which show all the name of the employees. On the right side, I have taken the Grid for Labels and TextBoxes for showing the Employee details and in below, I have taken two buttons for loading and saving the employees details.

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="clr-namespace:WpfApplication1.ViewModel"
        Title="MainWindow" Height="350" Width="525">
    <DockPanel LastChildFill="True" DataContext="{Binding
MainViewModel, Source={StaticResource Locator}}"> //LINE 6
```

```

        <ListBox DockPanel.Dock="Left" ItemsSource="{Binding EmployeeList}"
        DisplayMemberPath="Name" SelectedItem="{Binding SelectedEmployee}" Width="100" />

        <DockPanel DockPanel.Dock="Bottom">
            <Button DockPanel.Dock="Left" Content="Load Employees" Width="200"
            Height="25" Command="{Binding LoadEmployeesCommand}" />
            <Button DockPanel.Dock="Right" Content="Save Employees" Width="200"
            Height="25" Command="{Binding SaveEmployeesCommand}" />
        </DockPanel>

        <Grid>
            <Grid.Resources>
                <Style TargetType="{x:Type TextBox}">
                    <Setter Property="Height" Value="25" />
                    <Setter Property="Margin" Value="10,10,0,0" />
                </Style>
                <Style TargetType="{x:Type Label}">
                    <Setter Property="VerticalAlignment" Value="Center" />
                </Style>
            </Grid.Resources>
            <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>

            <Label Grid.Row="0" Grid.Column="0" Content="ID" />
            <TextBox Grid.Row="0" Grid.Column="1" Text="{Binding SelectedEmployee.ID}" />

            <Label Grid.Row="1" Grid.Column="0" Content="Name" />
            <TextBox Grid.Row="1" Grid.Column="1" Text="{Binding SelectedEmployee.Name}"
        />

            <Label Grid.Row="2" Grid.Column="0" Content="Age" />
            <TextBox Grid.Row="2" Grid.Column="1" Text="{Binding SelectedEmployee.Age}"
        />

            <Label Grid.Row="3" Grid.Column="0" Content="Salary" />
            <TextBox Grid.Row="3" Grid.Column="1" Text="{Binding
        SelectedEmployee.Salary}" />
        </Grid>
    </DockPanel>

</Window>

```

MVVM is mainly depend on Bindings for interaction between UI and the ViewModel classes.

In highlighted line 6, I have binded the DataContext property of DockPanel to the MainViewModel and set the Source property of binding to Locator. I will explain Locator in step 5. Below are the main bindings used in the above xaml.

Control	Property	Binding
ListBox	ItemsSource	EmployeeList
ListBox	SelectedItem	SelectedEmployee
Button(Load Employees)	Command	LoadEmployeesCommand
Button (Show Employees)	Command	SaveEmployeesCommand
TextBox (Employee ID)	Text	SelectedEmployee.ID
TextBox (Name)	Text	SelectedEmployee.Name
TextBox (Age)	Text	SelectedEmployee.Age
TextBox (Salary)	Text	SelectedEmployee.Salary

**MainWindows.xaml Bindings**

## Creating a Model class in MVVM Light Toolkit

Every model class in this MVVM Light Toolkit must inherit from ObservableObject. ObservableObject class is inherit from the INotifyPropertyChanged interface. This interface provides PropertyChanged event handler that notify clients that a property value has changed. ObservableObject use that event in RaisePropertyChanged method to notify other classes. In addition, ObservableObject class also provides a Set method to set the property and raise the PropertyChanged event automatically. Set method takes property name, reference to the private variable and the new value. Below I have shown an example of Employee class.

```
1    public class Employee : ObservableObject
2    {
3        private int id;
4        private string name;
5        private int age;
6        private decimal salary;
7
8        public int ID
9        {
10           get
11           {
12               return id;
13           }
14           set
```

```

12         {
13             Set<int>(() => this.ID, ref id, value);
14         }
15     }
16     public string Name
17     {
18         get
19         {
20             return name;
21         }
22         set
23         {
24             Set<string>(() => this.Name, ref name, value);
25         }
26     }
27     public int Age
28     {
29         get
30         {
31             return age;
32         }
33         set
34         {
35             Set<int>(() => this.Age, ref age, value);
36         }
37     }
38     public decimal Salary
39     {
40         get
41         {
42             return salary;
43         }
44         set
45         {
46             Set<decimal>(() => this.Salary, ref salary, value);
47         }
48     }
49     public static ObservableCollection<Employee> GetSampleEmployees()
50     {
51         ObservableCollection<Employee> employees = new ObservableCollection<Employee>();
52         for (int i = 0; i < 30; ++i)
53         {
54             employees.Add(new Employee
55             {
56                 ID = i + 1,
57                 Name = "Name " + (i + 1).ToString(),
58                 Age = 20 + i,
59                 Salary = 20000 + (i * 10)
60             });
61         }
62         return employees;
63     }

```

I have created four properties name ID, Name, Age, and Salary in above model class. All properties setter used the Set method of ObservableObject. Set method assign a new value to the property and then call the RaisePropertyChanged method. The use of RaisePropertyChanged method is must as we have to update the UI when any property changed.

The last method GetSampleEmployees() creates a temporary employee list in memory and returns an ObservableCollection of Employee class. I have used that list for binding to the UI so that we don't need to use the database to loading and saving the employee details.

## Create a ViewModel class using MVVM Light

Every viewmodel in MVVM Light Toolkit must inherit from ViewModelBase class. In the below MainViewModel class, we inherit our ViewModel from the ViewModelBase class.

```
1  public class MainViewModel : ViewModelBase
2  {
3      private ObservableCollection<Employee> employees;
4      private Employee selectedEmployee;
5
6      public MainViewModel()
7      {
8          LoadEmployeesCommand = new RelayCommand(LoadEmployeesMethod);
9          SaveEmployeesCommand = new RelayCommand(SaveEmployeesMethod);
10     }
11
12     public ICommand LoadEmployeesCommand { get; private set; }
13     public ICommand SaveEmployeesCommand { get; private set; }
14
15     public ObservableCollection<Employee> EmployeeList
16     {
17         get
18         {
19             return employees;
20         }
21     }
22
23     public Employee SelectedEmployee
24     {
25         get
26         {
27             return selectedEmployee;
28         }
29         set
30         {
31             selectedEmployee = value;
32             RaisePropertyChanged("SelectedEmployee");
33         }
34     }
35 }
```

```

31         public void SaveEmployeesMethod()
32         {
33             Messenger.Default.Send<NotificationMessage>(new NotificationMessage("Employees
34             Saved."));
35         }
36         private void LoadEmployeesMethod()
37         {
38             employees = Employee.GetSampleEmployees();
39             this.RaisePropertyChanged(() => this.EmployeeList);
40             Messenger.Default.Send<NotificationMessage>(new NotificationMessage("Employees
            Loaded."));
        }
    }

```

I have create two properties in this class name EmployeeList and SelectedEmployee. EmployeeList is binded to the ItemsSource property of ListBox in line 8 and SelectedEmployee is binded to the SelectedItem property of ListBox in line 8.

In addition, I have also created two commands name LoadEmployeesCommand and SaveEmployeeCommand. Both commands are instantiated using RelayCommand in the constructor of the class. RelayCommand is a command provided by the MVVM Light Toolkit which purpose is to relay its functionality to other objects by invoking delegates. LoadEmployeeCommand is binding to button in line 11 and SaveEmployeeCommand is binded to button in line 12. LoadEmployeeCommand execute the LoadEmployeesMethod and SaveEmployeesCommand execute the SaveEmployeesMethod when clicked.

In Load employees method, we load the employee list from the static method of Employee class. Next we notify the UI that EmployeeList property has been changed using the RaisePropertyChanged method.

In the last, we send the notification to the UI to show message box of text "Employee Loaded". For this we use the Messenger class of MVVM Light Toolkit. Messenger class allows object to exchange messages between different classes. The Default property of Messenger use the default instance of Messenger object provided by the toolkit. NotificationMessage is also provided by the toolkit and is used for passing a string message to recipient.

## ViewModelLocator Class for binding the ViewModel classes to View classes

We have created both our UI and ViewModel classes but for binding the UI to ViewModel classes we use another MVVM Light Toolkit feature called ViewModelLocator.

```

1     public class ViewModelLocator
2     {
3         public ViewModelLocator()
4         {
5             ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);
6         }
7     }

```

```

5
6         SimpleIoc.Default.Register<MainViewModel>();
7         Messenger.Default.Register<NotificationMessage>(this, NotifyUserMethod);
8     }
9
10    public MainViewModel MainViewModel
11    {
12        get
13        {
14            return ServiceLocator.Current.GetInstance<MainViewModel>();
15        }
16    }
17
18    private void NotifyUserMethod(NotificationMessage message)
19    {
20        MessageBox.Show(message.Notification);
21    }
22

```

ViewModelLocator class is the mediator between your UI and ViewModels that binds the ViewModels to the UI. If you want to use your ViewModel as binding source for the UI, you must expose that ViewModel as property in the ViewModelLocator class.

In the line 5, we set the IOC container for the application. MVVM Light Toolkit provides a default IOC container. IOC container is used for registering and resolving instances.

In line 7, we register the MainViewModel to the IOC container.

In line 8, we register the NotifyUserMethod with the Messenger class. So that, when we send the text message with the Messenger class using NotificationMessage it automatically execute the NotifyUserMethod.

For using the ViewModelLocator class throughout the application, we create an instance of the class in the App.xaml using key "Locator".

```

1
2    <Application x:Class="WpfApplication1.App" xmlns="http://schemas.microsoft.com/winfx/2006/xaml"
3        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4        StartupUri="MainWindow.xaml" xmlns:d="http://schemas.microsoft.com/expression/2010/declarative"
5        xmlns:vm="clr-namespace:WpfApplication1.ViewModel">
6        <Application.Resources>
7            <vm:ViewModelLocator x:Key="Locator" />
8        </Application.Resources>
9    </Application>

```

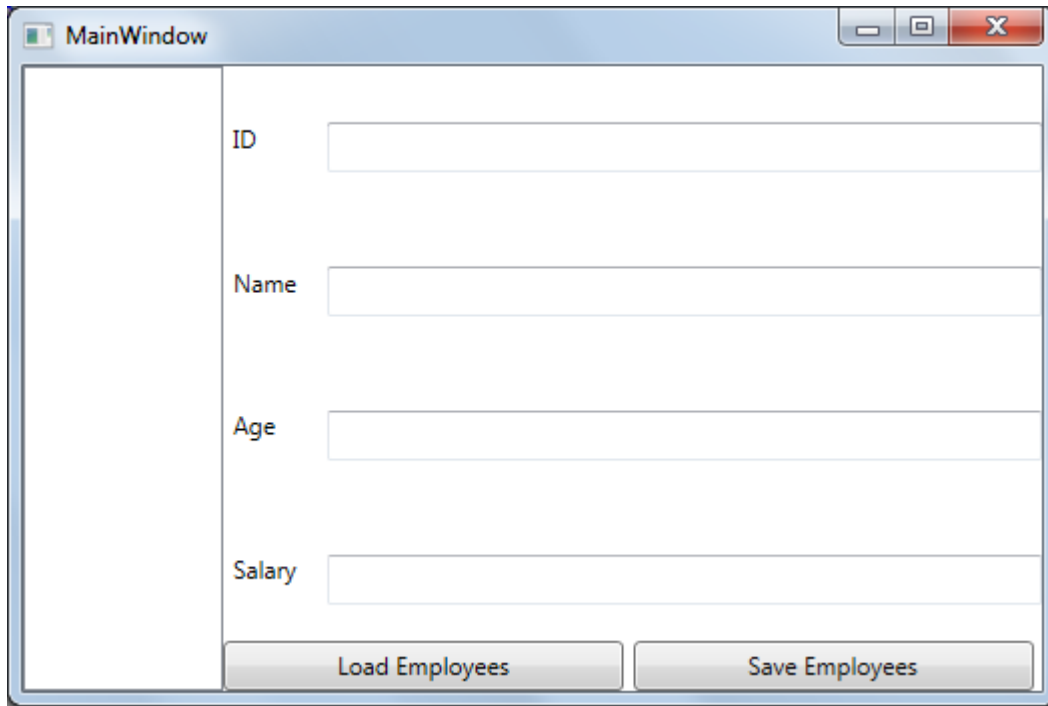
In line 6, we create an instance of locator class as key "Locator". We set this key in xaml in line 6 as Source property of DataContext of DockPanel.



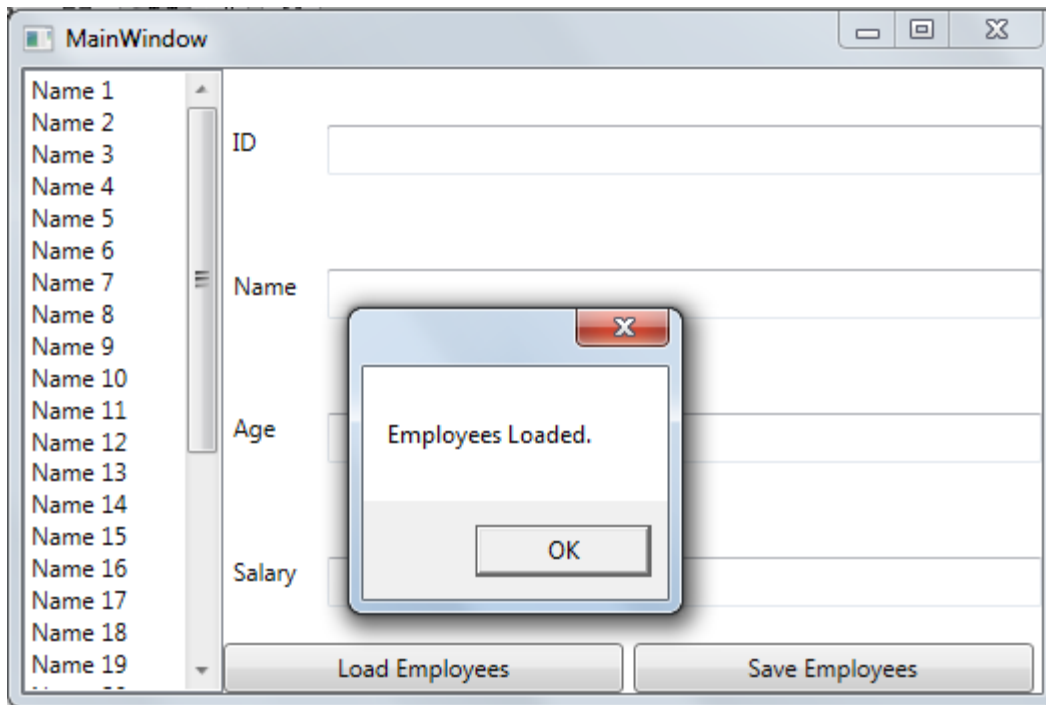
1 <DockPanel LastChildFill="True" DataContext="{Binding MainViewModel, Source={StaticResource

## Build the project and test application

Lets build the project and run.

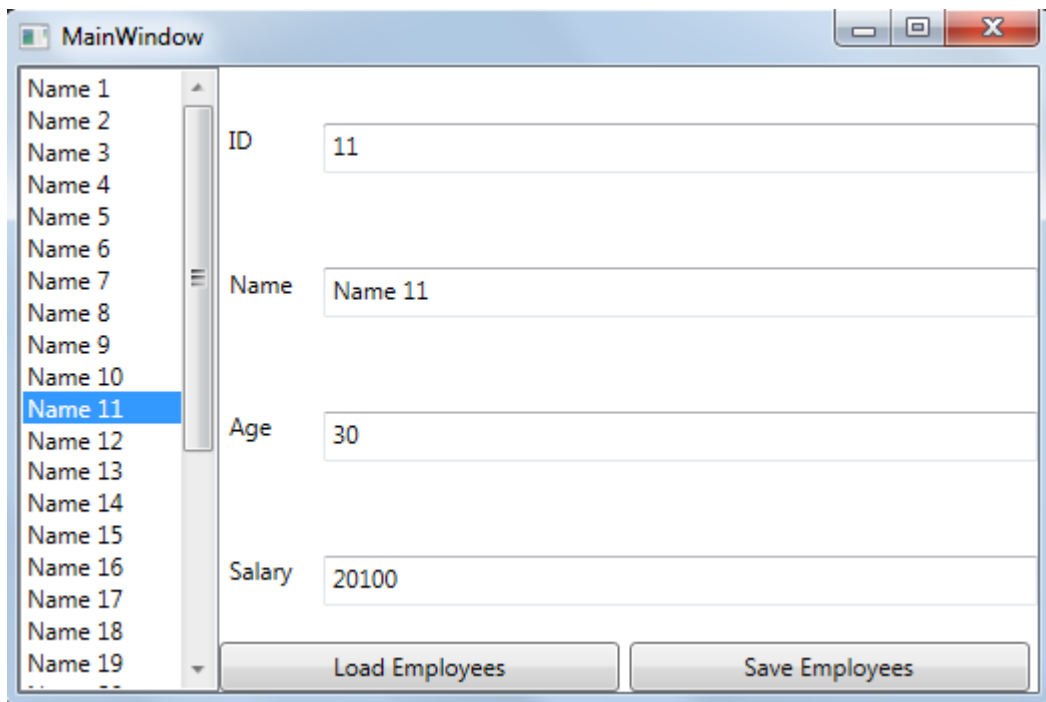


Click on the "Load Employees" button.

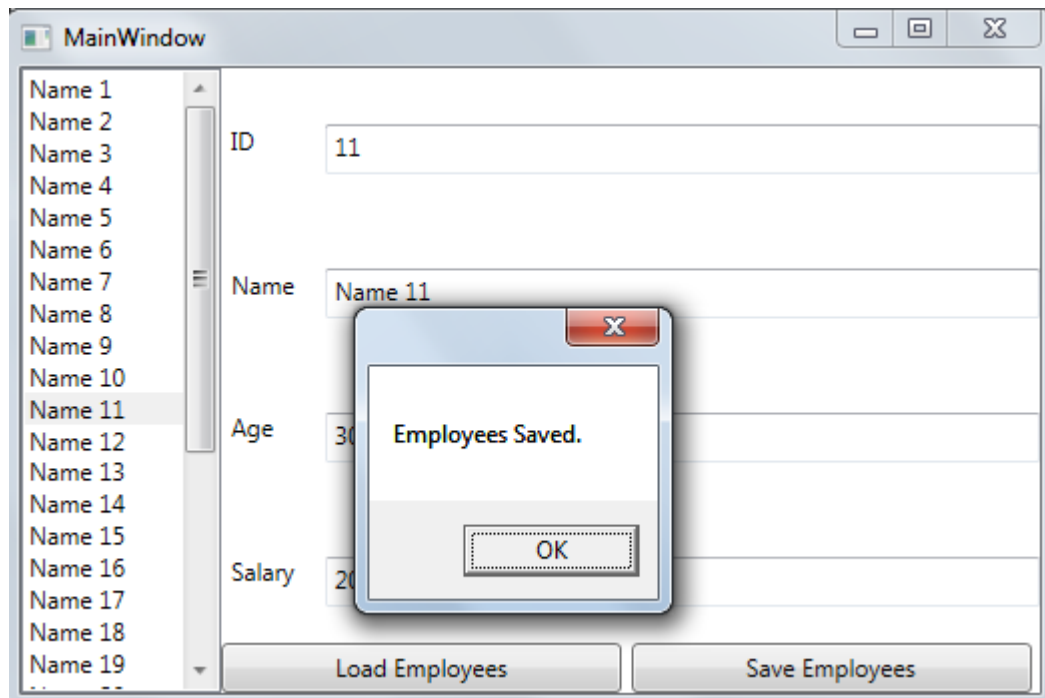


This will show the message box as "Employees Loaded" message and load the employees on the left side ListBox.

Click on any name in the left side ListBox and it automatically fill the right side textboxes with the details of the selected employee.



and last click on the Save Employees button, it will show a message box of "Employee Saved" text.



Thats it. We have created a very simple application in MVVM using MVVM Light Toolkit.