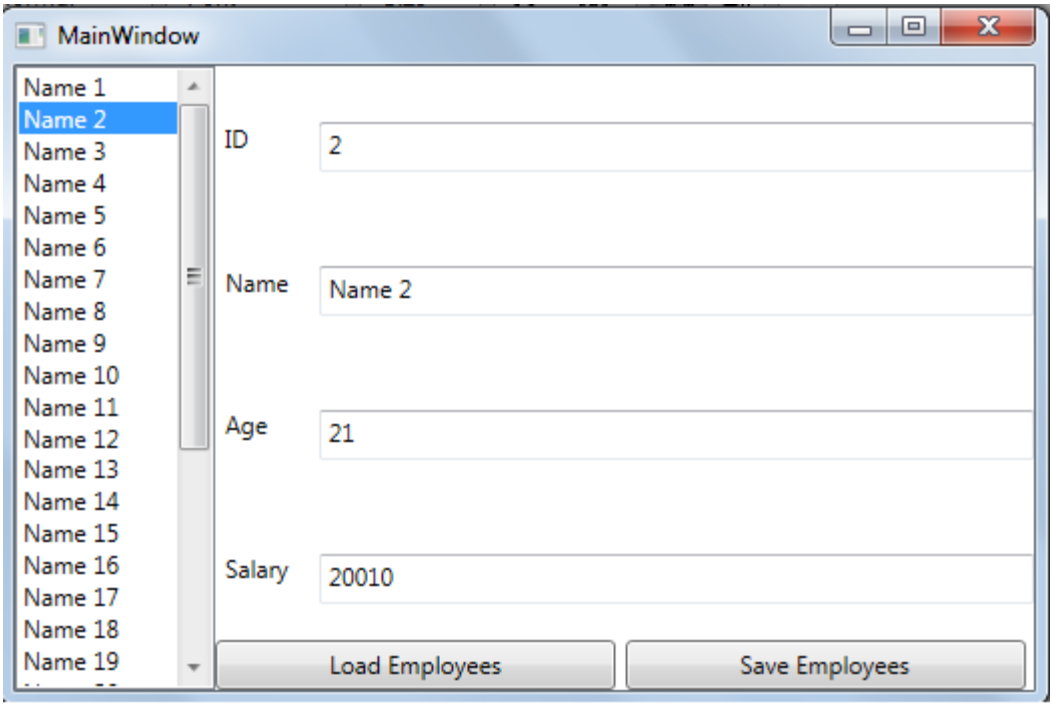
WPF with MVVM pattern and MVVM Toolkit

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

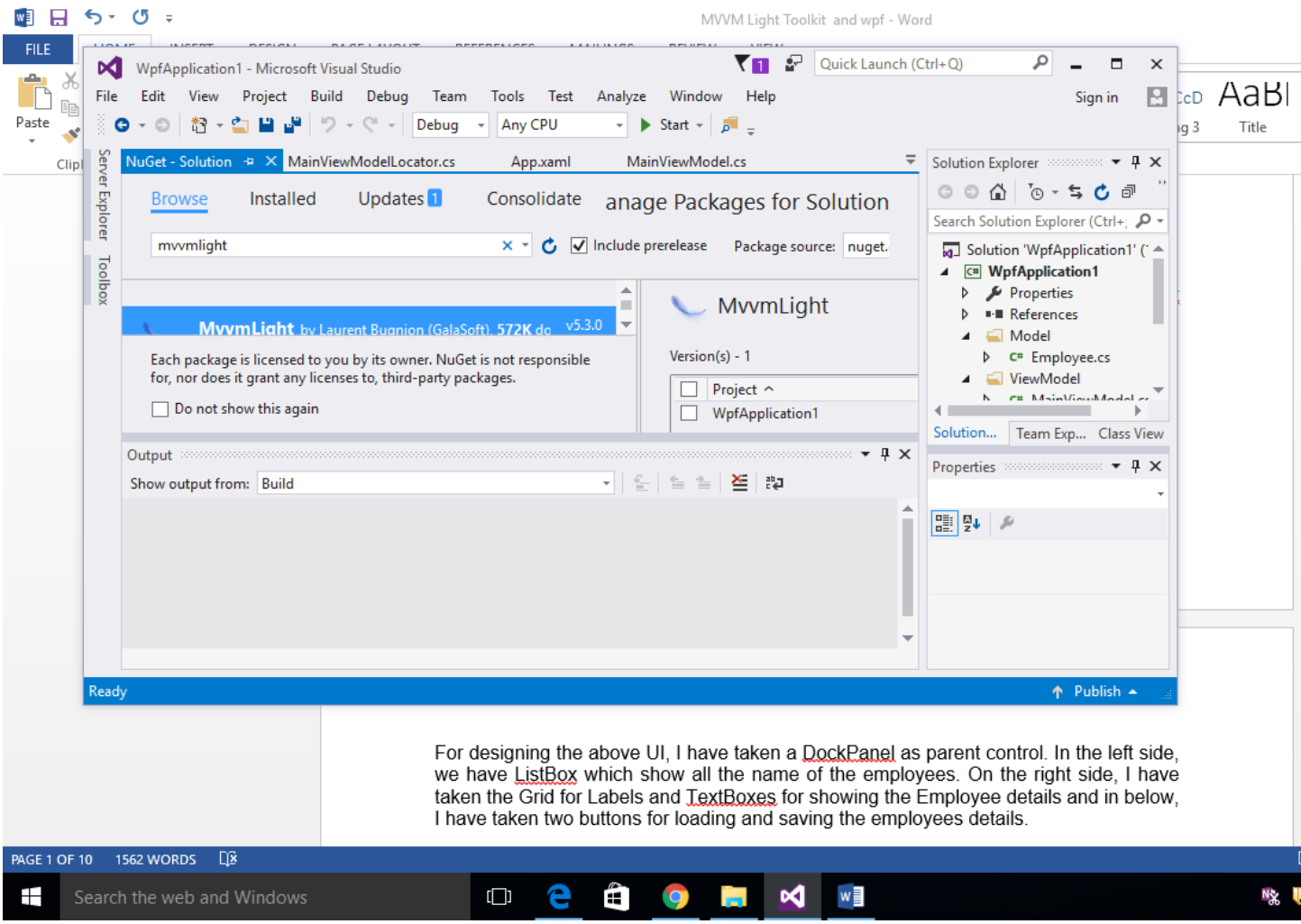


We will do below tasks one by one:

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-> NuGet 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 1

        <ListBox DockPanel.Dock="Left" ItemsSource="{Binding EmployeeList}"
DisplayMemberPath="Name" SelectedItem="{Binding SelectedEmployee}" Width="100" />//line 2
```

```
        <DockPanel DockPanel.Dock="Bottom">
            <Button DockPanel.Dock="Left" Content="Load Employees" Width="200"
Height="25" Command="{Binding LoadEmployeesCommand}" />//line 3
            <Button DockPanel.Dock="Right" Content="Save Employees" Width="200"
Height="25" Command="{Binding SaveEmployeesCommand}" />//line 4
        </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 1, 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 INofityPropertyChanged interface. This interface provides PropertyChanged event handler that notifiy 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.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using GalaSoft.MvvmLight;
using System.Collections.ObjectModel;
using GalaSoft.MvvmLight.Command;
using System.Windows.Input;

namespace WpfApplication1.Model
```

```
{
    public class Employee : ObservableObject
    {
        private int id;
        private string name;
        private int age;
        private decimal salary;

        public int ID
        {
            get
            {
                return id;
            }
            set
            {
                Set<int>(() => this.ID, ref id, value);
            }
        }

        public string Name
        {
            get
            {
                return name;
            }
            set
            {
                Set<string>(() => this.Name, ref name, value);
            }
        }

        public int Age
        {
            get
            {
                return age;
            }
            set
            {
                Set<int>(() => this.Age, ref age, value);
            }
        }

        public decimal Salary
        {
```

```
            get
            {
                return salary;
            }
            set
            {
                Set<decimal>(() => this.Salary, ref salary, value);
            }
        }

        public static ObservableCollection<Employee>
GetSampleEmployees()
        {
            ObservableCollection<Employee> employees = new
ObservableCollection<Employee>();
            for (int i = 0; i < 30; ++i)
            {
                employees.Add(new Employee
                {
                    ID = i + 1,
                    Name = "Name " + (i + 1).ToString(),
                    Age = 20 + i,
                    Salary = 20000 + (i * 10)
                });
            }
            return employees;
        }
    }
}
```

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.

```
using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;
using GalaSoft.MvvmLight.Messaging;
using System.Windows.Input;
using System.Collections.ObjectModel;
using WpfApplication1.Model;
namespace WpfApplication1.ViewModel
{
    /// <summary>
    /// This class contains properties that the main View can data
bind to.
    /// <para>
    /// Use the <strong>mvvminpc</strong> snippet to add bindable
properties to this ViewModel.
    /// </para>
    /// <para>
    /// You can also use Blend to data bind with the tool's support.
    /// </para>
    /// <para>
    /// See http://www.galasoft.ch/mvvm
    /// </para>
    /// </summary>
    public class MainViewModel : ViewModelBase
    {
        private ObservableCollection<Employee> employees;
        private Employee selectedEmployee;

        public MainViewModel()
        {
            LoadEmployeesCommand = new
RelayCommand(LoadEmployeesMethod);
            SaveEmployeesCommand = new
RelayCommand(SaveEmployeesMethod);
        }

        public ICommand LoadEmployeesCommand { get; private set; }
        public ICommand SaveEmployeesCommand { get; private set; }
```

```
        public ObservableCollection<Employee> EmployeeList
        {
            get
            {
                return employees;
            }
        }

        public Employee SelectedEmployee
        {
            get
            {
                return selectedEmployee;
            }
            set
            {
                selectedEmployee = value;
                RaisePropertyChanged("SelectedEmployee");
            }
        }

        public void SaveEmployeesMethod()
        {
            Messenger.Default.Send<NotificationMessage>(new
NotificationMessage("Employees Saved."));
        }

        private void LoadEmployeesMethod()
        {
            employees = Employee.GetSampleEmployees();
            this.RaisePropertyChanged(() => this.EmployeeList);
            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 2 and SelectedEmployee is binded to the SelectedItem property of ListBox in line 2.

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 3 and SaveEmployeeCommand is binded to button in line 4. 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.

```
using System;
using System.Windows;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using WpfApplication1.ViewModel;
using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Ioc;
using GalaSoft.MvvmLight.Messaging;
using Microsoft.Practices.ServiceLocation;


namespace WpfApplication1
{
    class MainViewModelLocator
    {
        public MainViewModelLocator()
        {
```

```
            ServiceLocator.SetLocatorProvider(() =>
SimpleIoc.Default); //line 1

            SimpleIoc.Default.Register<MainViewModel>();//line 2
            Messenger.Default.Register<NotificationMessage>(this,
NotifyUserMethod);//line 3
        }

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

        private void NotifyUserMethod(NotificationMessage message)
        {
            MessageBox.Show(message.Notification);
        }
    }
}
```

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 1, we set the IOC container for the application. MVVM Light Toolkit provides a default IOC container. IOC container is used for regitering and resolving instances.

In line 2, we register the MainViewModel to the IOC container.

In line 3, 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 througout the application, we create an instance of the class in the App.xaml using key "Locator".

```
<Application x:Class="WpfApplication1.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:WpfApplication1"
StartupUri="MainWindow.xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

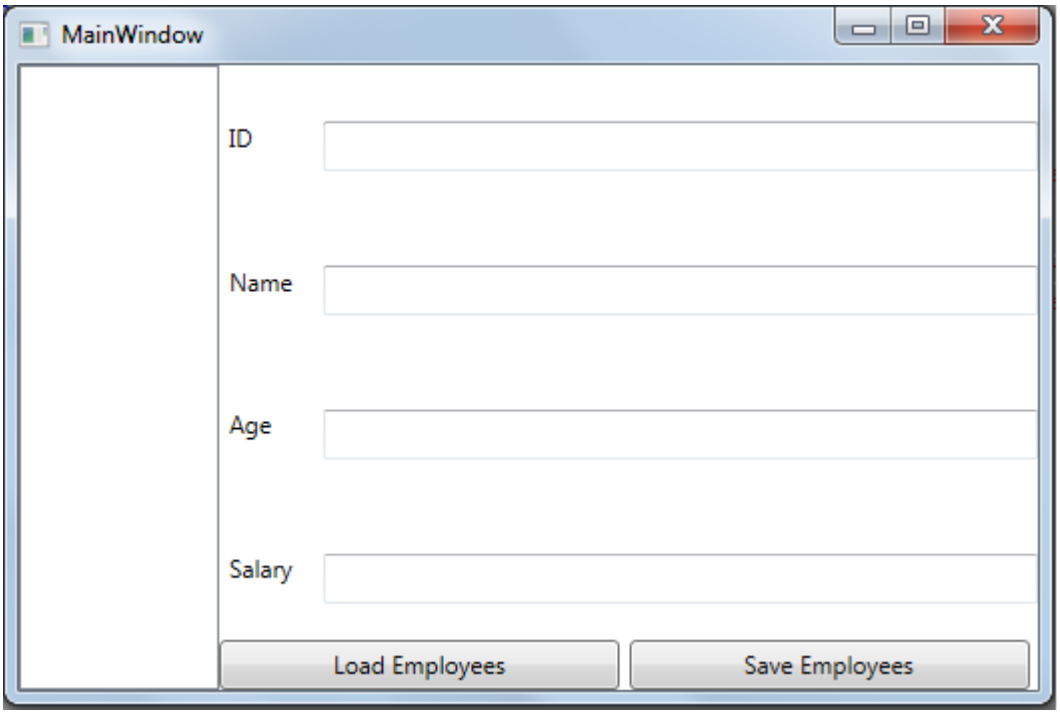
```
d1p1:Ignorable="d"
xmlns:d1p1="http://schemas.openxmlformats.org/markup-
compatibility/2006">
  <Application.Resources>
    <ResourceDictionary>
       <vm:MainViewModelLocator x:Key="Locator" d:IsDataSource="True"
xmlns:vm="clr-namespace:WpfApplication1" /> line 1
    </ResourceDictionary>
  </Application.Resources>

</Application>
```

In line 1, we create an instance of locator class as key "Locator". We set this key in xaml in line 1 as Source property of DataContext of DockPanel.

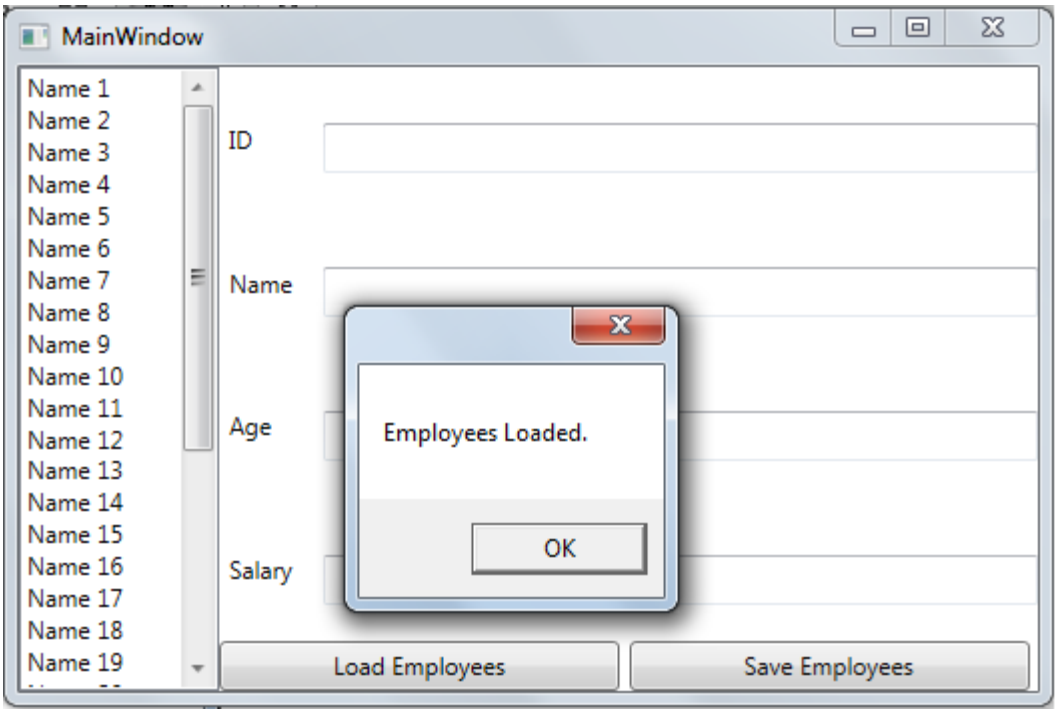
```
<DockPanel LastChildFill="True" DataContext="{Binding MainViewModel,
Source={StaticResource Locator}}">
```

Build the project and test application

Lets build the project and run.

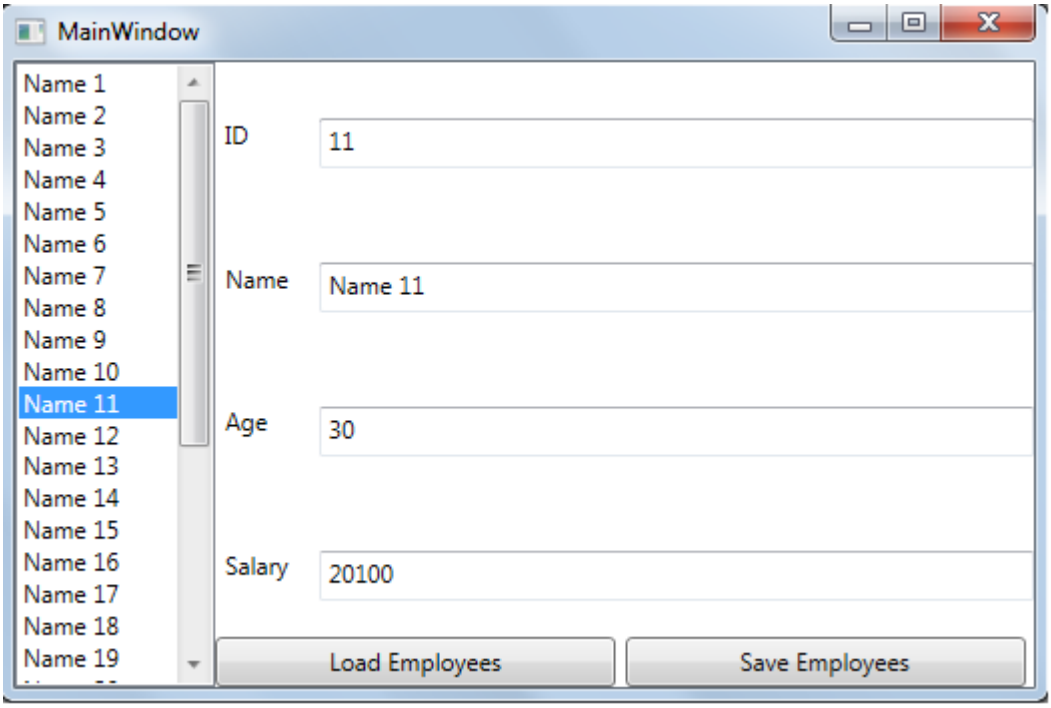


Click on the "Load Employees" button.

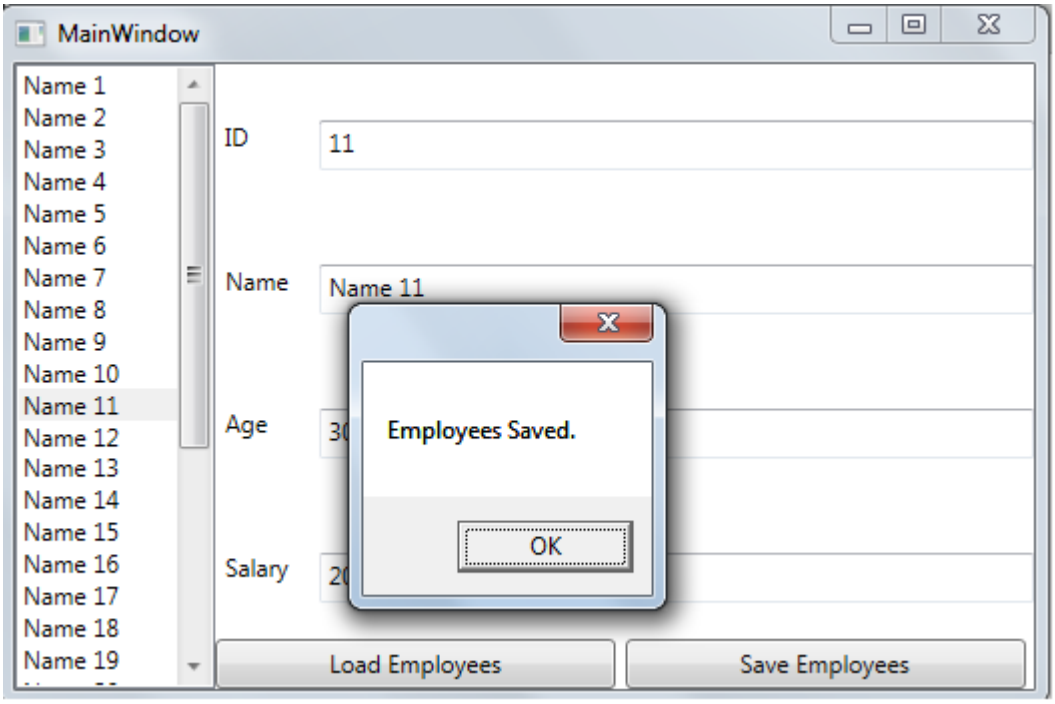


This will show the message box as "Emloyees Loaded" message and load the emplyees on the left side ListBox.

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.