Composite Applications with Prism in WPF - Part 2

Snesh Prajapati, 27 Nov 2015

This is second part of the article series with a demo WPF application using Prism and MVVM. In this part we will enhance the demo by adding common infrastructures and five modules.

Download Source Code for Part 2 - 1.5 MB

Introduction

This is second part of the article series having total three parts. In first part we have created Shell and defined the regions to load modules within regions of Shell. In this part we will enhance the demo created in first part by adding common infrastructures and five modules as described in the architecture section of first part. We will also see how to load modules either *on application start* or later *on demand*. So let us download code from first part, open ITHelpDeskDemoApp solution and will start working on it in further sections of this article.

Note: If you have not gone through first part, I recommend you to first visit there because it will give you better context and help you while going through this article.

Outlines

Part 1

- Overview of Final Demo
- Architecture of Final Demo
- What is Prism
- What is Shell
- Creating Shell and Defining Regions

Part 2

- Creating Common Infrastructure
- Creating Modules
    1. Navigation Module
    2. Employee Module

3. Software Module
4. Hardware Module
5. Request Module

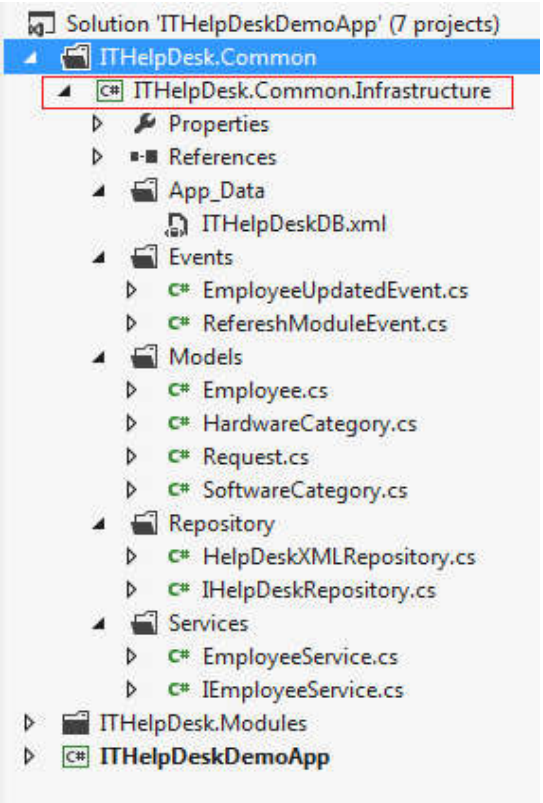
- Loading Modules
- Issues in Demo Application

Part 3

- What is Event Aggregator
- Use of Event Aggregator
- What is IActiveAware Interface
- Use of IActiveAware

Creating Common Infrastructure

First add a folder named as "ITHelpDesk.Common" to solution (ITHelpDeskDemoApp). Now add a Class Library project named as ITHelpDesk.Common.Infratructure to this folder. Purpose of this project is to have common code which will be used in all other projects of the application. This project will provide Services, Events, Models etc. needed for application to work. In this project we are using xml file as a database. Make the project structure same as shown below by adding folders and files to ITHelpDesk.Common.Infratucture project.



In above screenshot of "ITHelpDesk.Common.Infrastructure" project, we have added App_Data, Events, Models, Repository and Services folders and files to them. Following is the explanation of each files and folders.

App_Data: When we develop any application, it may require to interact with database to perform CRUD operations. In this demo app, ITHelpDeskDB.xml file is used as database. We can use any database at the place of ITHelpDeskDB.xml. Here we are using XML file as it is easy for readers and downloaded code can be executed without doing any database setup.

Event: It is having two EmployeeUpdatedEvent and RefereshModuleEvent class files, we will discuss about it in next part of this article.

Models: It is having all required entity classes which are required in application.

Repository: Here we are using repository pattern to decouple persistent layer and rest of the application. It is having interface called IHelpDeskRepository and class called HelpDeskXMLRepository. Outlines of code are shown below:

```
public class HelpDeskXMLRepository : IHelpDeskRepository
{

    private XDocument itHeplData;

    private string filePath = "../../../ITHelpDesk.Common.Infrastructure/App_Data/ITHelpDeskDB.xml";
    XElement ex;
    0 references
    public HelpDeskXMLRepository()...
    2 references
    public Employee GetEmployeeById(int empId)...
    2 references
    public string GetManagerNameById(int empId)...
    2 references
    public List<SoftwareCategory> GetSoftwareCategories()...
    2 references
    public List<SoftwareItems> GetSoftwareItemsByCategory(string softwareCategoryName)...
    2 references
    public List<HardwareCategory> GetHardwareCategories()...
    2 references
    public List<HardwareItems> GetHardwareItemsByCategory(string hardwareCategoryName)...
    2 references
    public bool SaveSoftwareRequest(int empId, string selectedCategory, string selectedSoftware, string comment)...
    2 references
    public bool SaveHardwareRequest(int empId, string selectedCategory, string selectedSoftware, string comment)...
    2 references
    public List<Request> GetAllRequest(int empId)...
}
```

Services: This folder is simply having two files, one is interface called IEmployeeService and second is class called EmployeeService. EmployeeService is implementing IEmployeeService interface as shown below. It is will provide employees related services to the app. Outlines of code are shown below:

```
public class EmployeeService : IEmployeeService
{
    private IHelpDeskRepository dataSource;
    9 references
    public int CurrentEmployeeId { get; set; }
    0 references
    public EmployeeService(IHelpDeskRepository repository)...
    2 references
    public Employee GetCurrentEmployeeDetail(int currentEmployeeId)...
    2 references
    public string GetManagerNameOfCurrentEmp()...
    2 references
    public List<SoftwareCategory> GetSoftwareCategories() ...
    2 references
    public List<SoftwareItems> GetSoftwareItemsByCategory(string softwareCategoryName) ...
    2 references
    public List<HardwareCategory> GetHardwareCategories()...
    2 references
    public List<HardwareItems> GetHardwareItemsByCategory(string hardwareCategoryName) ...
    2 references
    public bool SaveSoftwareRequest(string selectedCategory, string selectedSoftware, string comment)...
    2 references
    public bool SaveHardwareRequest(string selectedCategory, string selectedSoftware, string comment)...
    2 references
    public List<Request> GetAllRequest()...
}
```

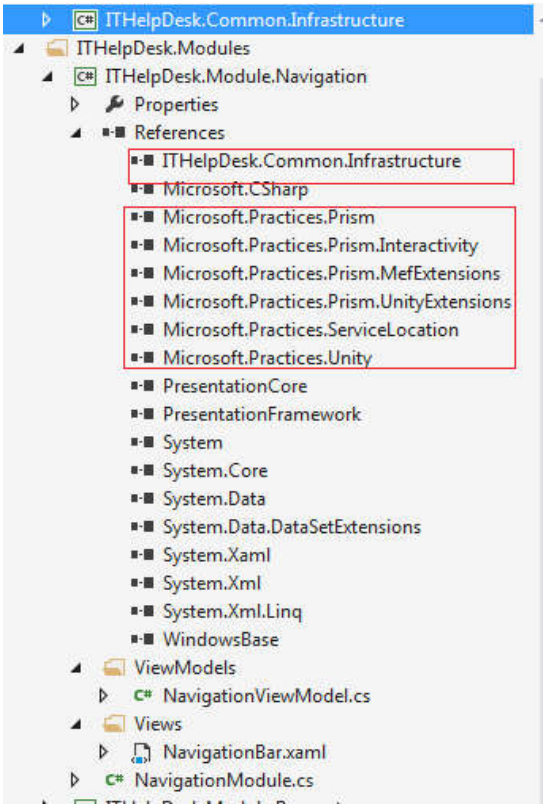
In constructor of EmployeeService we are passing/injecting dependencies of EmployeeService class. EmploeeService class is dependent on IHelpDeskRepository. For IHelpDeskRepository, we are injecting instance of HelpDeskXMLRepository. We will configure this association in ITHelpDeskBootstrapper class of startup project ITHelpDeskDemoApp.

Creating Modules

All modules need to be placed in a folder called *ITHelpDesk.Modules* at Solution level so that all module project will be together. So let's add a folder with the same name.

Navigation Module

Add a project to ITHelpDesk.Modules folder called as ITHelpDesk.Module.Navigation and make project structure same as shown below by adding reference of Prism 4 and required folder and files.



In Views folder add a UserControl named as NavigationBar.xaml. This file is having code to create navigation bar with three button controls. Those buttons are bound with thre different DelegateCommands in NavigationViewModel. On each button click related DelegateCommand will be fired to call appropriate method. That method will load appropriate module into specified region within Shell.

NavigationViewModel: Most of the code in NavigationViewModel class file is straight forward so we will only discuss about the code which is responsible to activate and deactivate module.

Following code is executed on "Software" button click in navigation bar. On that DelegateCommand we have bound loadSoftwareModule method as delegate which will be executed.

Hide Copy Code

```
private void loadSoftwareModule()
{
      // LoadModule method is responsible to load and initialize the module
      // It loads only if module is not initialize already.
      ModuleManager.LoadModule("SoftwareModule");
      var requestInfoRegion = RegionManager.Regions["RequestInfoRegion"];
      var newView = requestInfoRegion.GetView("SoftwareDetail");
      // As RequestInfoRegion uses ContentControlRegionAdapter so at a time only one view will be activated.
      requestInfoRegion.Activate(newView);
}
```

To behave a class as module that class must be implementing **IModule** interface. That is why at the root level of this project we have created NavigationModule file which is implementing IModule interface. IModule interface has a single method named as Initialize. In this method we need to write logic to be executed while initializing the module.

In the constructor of NavigationModule module we are injecting two dependency called as IUnityContainer and IRegionManager. Here IUnityContainer is responsible to resolve the dependencies and IRegionManager is used to register view to region (NavigationBar to NavigationRegion) by using extension method called RegisterViewWithRegion of RegionManagerExtensions class.

Hide Copy Code

```
using Microsoft.Practices.Prism.Modularity;
using Microsoft.Practices.Prism.Regions;
using Microsoft.Practices.Unity;
namespace ITHelpDesk.Module.Navigation
{
```

```
        public class NavigationModule : IModule
        {
            private readonly IRegionManager regionManager;
            private readonly IUnityContainer container;
            public NavigationModule(IUnityContainer container, IRegionManager    regionManager)
            {
                this.container = container;
                this.regionManager = regionManager;
            }
            public void Initialize()
            {
                this.regionManager.RegisterViewWithRegion("NavigationRegion",
                                                        () => this.container.Resolve<navigationbar>());

            }
        }
    }
</navigationbar>
```

In Initialize method, we are registering NavigationBar view to NavigationRegion region of Shell with the help of regionManager.

Employee Module

Project structure for this module is same as Navigation module. It is having code to create a layout for employee module and display employee's information when user enter employee id. This module is loaded within "EmployeeInfoRegion" of Shell. Employee Module is loaded when application loads.

Software Module

This module is loaded within "RequestInfoRegion" of Shell. Software module is loaded on demand when user clicks on Software button. In the view page (SoftwareDetail.xaml) of this module we have written code to create layout. There are two combobox, one textbox and one button is used in SoftwareDetail view. ItemSource property of **first combobox** is bound with "SoftwareCategories" property in SoftwareViewModel. "SoftwareCategories" property returns ObservableCollection of SoftwareCategory.

SoftwareViewModel ViewModel fetches data by calling fetchSoftwareCategory method from its constructor. fetchSoftwareCategory method brings list of software from the ITHelpDeskDB.xml using EmployeeService. EmployeeService calls appropriate method from Repository which pulls data from ITHelpDeskDB.xml file. Code for fetchSoftwareCategory method is shown below:

Hide Copy Code

```
private void fetchSoftwareCategory()
{
    List<SoftwareCategory> allsoftwareCategories = employeeService.GetSoftwareCategories();
    SoftwareCategories = new ObservableCollection<SoftwareCategory>(allsoftwareCategories);
}
```

ItemSource property of **second combobox** is bound with "SoftwareList" property in SoftwareViewModel. "SoftwareList" property returns ObservableCollection of SoftwareList by calling fetchSoftwareList method from the set method of "SelectedSoftwareCategory" property of SoftwareViewModel. Code for fetchSoftwareList method is shown below:

Hide Copy Code

```
private void fetchSoftwareList(string selectedCategory)
{
    List<SoftwareItems> allSoftwareList = employeeService.GetSoftwareItemsByCategory(selectedCategory);
    SoftwareList = new ObservableCollection<SoftwareItems>(allSoftwareList);
}
```

Textbox is used for writing comments. Command property of "Submit Request" is boud with "SubmitRequestCommand" in SoftwareViewModel. SubmitRequestCommand is a DelegateCommand which calles saveRequest method to save request. Following method is responsible to save request to ITHelpDeskDB.xml .

Hide Copy Code

```
private void saveRequest()
{
    string selectedCategory = SelectedSoftwareCategory;
    string selectedSoftware = SelectedSoftwareName;
    string comment = Comment;
    bool result = employeeService.SaveSoftwareRequest(selectedCategory, selectedSoftware, comment);
    if (result == true)
    {
        Message = "Successful:  our request is accepted.";
        IsPopupOpen = true;
    }
    else
    {
        Message = "Unsuccessful: Please try again.";
        IsPopupOpen = true;
    }
}
```

Hardware Module

Hardware module is having similar functionality and code as we have implemented in SoftwareModule. So please follow similar steps to create this module.

All Request Module

Project structure is same as above modules as we have implemented. User need to enter Employee Id in "Enter Employee Id" field. All request Ids associated with current Employee Id will be populated to "Select Request Id" combobox. As user selects a request Id from the combobox, all information related to selected request Id will be shown.

As per our requirement, we need to show all request Ids associated with particular Employee Id at any time. As user enters Employee Id in textbox of EmployeeModule call goes to EmployeeService to get employee's information. At that time we are storing the recently entered Employee Id in public property of EmployeeService called CurrentEmployeeId. And from the constructor of RequestViewModel we are calling fetchAllRequest method which makes call to EmployeeService and get filtered data based on CurrentEmployeeId.

So far we have implemented all modules, now we need to write code to load modules on application start and on demand. Let us implement this functionality in next section.

Loading Modules

There are two approaches for loading modules. With this demo application we are using both approaches as described below.

On Application Start: Navigation and Employee module is loading when application starts.

On Demand: Software, Hardware and All Request modules are loading on demand.

To achieve that in ITHelpDeskBootstrapper class, we are overriding ConfigureModuleCatalog method. ConfigureModuleCatalog is virtual method of Bootstrapper class. In this method we are defining which module need to be loaded on application start and on demand.

Hide Copy Code

```
protected override void ConfigureModuleCatalog()
{
    ModuleCatalog moduleCatalog = (ModuleCatalog)this.ModuleCatalog;
    moduleCatalog.AddModule(typeof( mployeeModule));
    moduleCatalog.AddModule(typeof(NavigationModule));
    moduleCatalog.AddModule(typeof(SoftwareModule), InitializationMode.OnDemand);
    moduleCatalog.AddModule(typeof(HardwareModule), InitializationMode.OnDemand);
    moduleCatalog.AddModule(typeof(RequestModule), InitializationMode.OnDemand);
}
```

In ConfigureContainer method, we are registering types (HelpDeskXMLRepository and EmployeeService). So unity will be aware how to resolve corresponding interfaces.

Hide Copy Code

```
protected override void ConfigureContainer()
{
    Container.RegisterType<IHelpDeskRepository, HelpDesk MLRepository>();
    Container.RegisterType<I mployeeService,  mployeeService>(new ContainerControlledLifetimeManager());
    base.ConfigureContainer();
}
```

Issues in Demo Application

As of now application works fine in general flow, but there are cases when it shows wrong data. Such one cases are described below:

If look closely, when user enter Employee Id in Employee module, let say, 2, and Navigate to All Request module, in combobox, all requests Id's associated with current Employee Id 2 are available. Now when user change to Employee Id 3, request Ids in combobox does not change. Combobox is still showing old data.

The reason and solution (Communication between Modules) will be discussed in next article of this series. There we will also look into such more bugs with specific cases producing wrong data and will resolve those.

Conclusion

In this part of the article series, we learned about how to add Modules and Infrastructure to the application. In next part of this article series, we will continue with same demo and enhance it further. We will be focusing on making it robust for specific use cases where current demo app does not work as exepected. Your comments/suggestions and queries are most welcome. Thanks.

License

Share

About the Author

Snesh Prajapati
Software Developer
India

I am a Software Developer working on Microsoft technologies. My interest is exploring and sharing the awesomeness of emerging technologies.

You may also be interested in...

Flip jQuery Plugin for Previewing Web Content

AdES: An Implementation of CAdES for Windows in C++

Comments and Discussions

You must Sign In to use this message board.

Search o ents

Spacing ela ed Layout or al Per page pdate

First Prev Next

Please add link to Part 3	**@nkit Bajpai**	**15-Apr-17 0:30**
Re: Please add link to Part 3	Snesh Prajapati	18-Apr-17 4:40
Re: Please add link to Part 3	@nkit Bajpai	19-Apr-17 3:23
Suggestion	**Afzaal Ahmad Zeeshan**	**28-Nov-15 0:19**

Last Visit: 31-Dec-99 18:00 Last Update: 14-Sep-18 5:51 Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.