13,695,827 members

Sign in

Search for articles, questions, tips

home articles quick answers discussions features community help

Articles » Platforms, Frameworks & Libraries » Windows Presentation Foundation » Applications

Composite Applications with Prism in WPF - Part 3

Snesh Prajapati, 5 Dec 2015

5.00 (12 votes) Rate this:

In this article we will learn about EventAggregator and IActiveAware interface and its uses in WPF application using Prism. This is continuation of second part of article series having total three parts.

Download Source Code for Part 3 - 1.5 MB

Introduction

In this article we will learn about EventAggregator and IActiveAware interface and its uses in WPF application using Prism. This is continuation of second part of 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 second part we have created common infrastructure and five modules. So far we are able to load modules on application start as well as on demand and able to save request to database (xml file). As we have discussed in last section of second part, there is an issue with "All Request" module. In this part we will fix that issue and fulfill one more requirement.

Note: If you have not gone through **first part** and **second part**, I recommend you to first visit those because it will give you better context and help you to understand this article better.

Outlines

Recap of the Issue

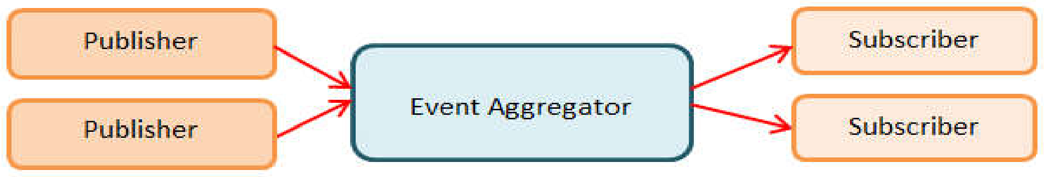
We left last demo application with an issue in "All Request" module discussed in last section of second part. Once "All Request" module is loaded for the first time, combobox is showing correct request Ids associated with entered Employee Id. But next time when Employee Id get change, combobox is **not** showing correct request Ids which are associated with latest Employee Id.

As per our requirement whenever Employee Id gets change, in "All Request" module combobox need to show all request Ids associated with recently entered Employee Id. To accomplish that we need to notify "All Request" module. This issue can be fixed with the help of EventAggregator.

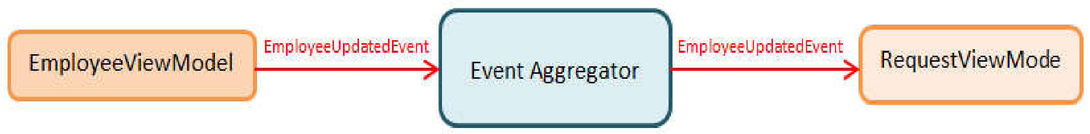
What is EventAggregator

EventAggregator provides a global service to publish or subscribe events across application. EventAggregator is a class which implements IEventAggregator interface. It comes under "Microsoft.Practices.Prism.Events" namespace in Microsoft.Practices.Prism.dll. EventAggregator works as a mediator between publisher and subscriber objects for events. Whosever object wants to publish any event, need to communicate to EventAggregator by specifying parameter of event if any. Whosever object subscribes that event is notified by EventAggregator with published parameter (payload of event).

Following screenshot shows general pictorial description of EventAggregator with multiple publishers and subscribers.



In Demo application, EmployeeViewModel is playing role of publisher and RequestViewModel is playing role of subscriber. EventAggregator works as a mediator between EmployeeViewModel and RequestViewModel module. Following screenshot shows implementation of EventAggregator in Demo application:



How to Use EventAggregator

To use EventAggregator, there are some standard steps need to be followed. We will follow those steps in this section to use EventAggregator in order to fix the issue.

1. First we need create a class inherited from "CompositePresentationEvent<TPayload>" that serve as event which we want to notify to different part of application. Already, in first part of the article series, we have added a class called EmployeeUpdatedEvent under Eventes folder in ITHelpDesk.Common.Infrastructure project. Typically such events are declared in a common project (here we have ITHelpDesk.Common.Infrastructure) because those events are supposed to be used in multiple projects in the application.

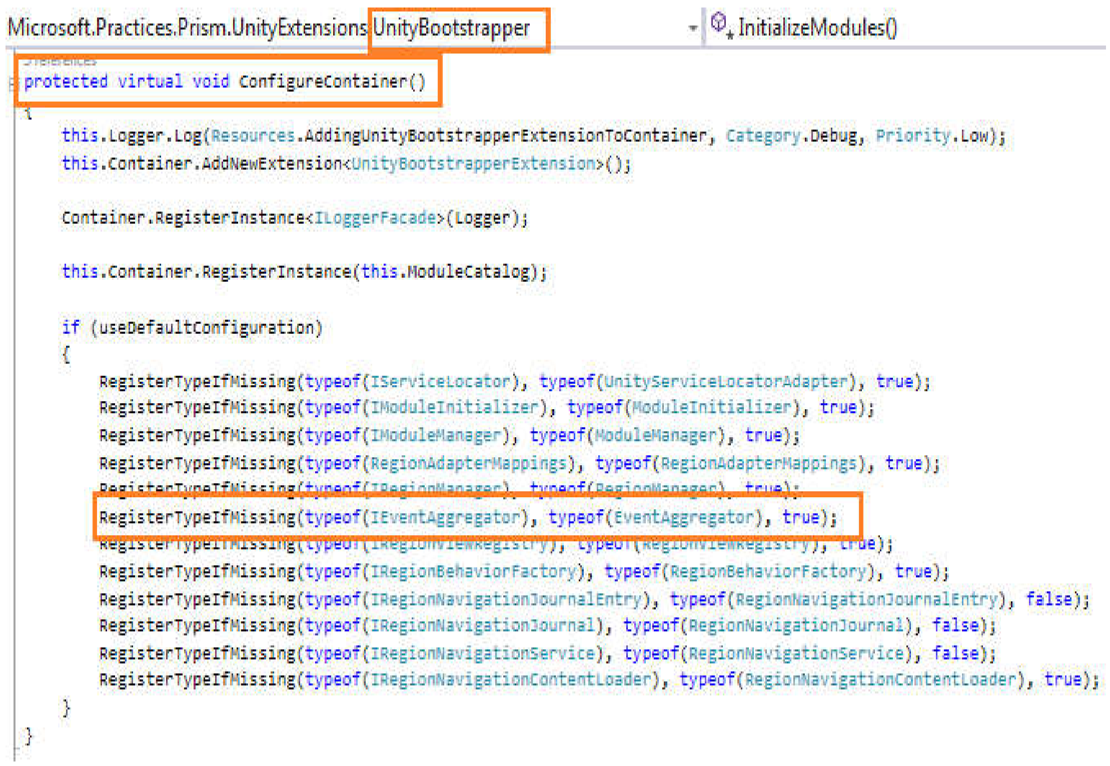
   When we use EventAggregator with Prism based application then we get generic class called "CompositePresentationEvent". Since it is generic class so we need to provide a specific type. This type of message will be passed to the subscribers. So we have created EmployeeUpdatedEvent class and inherited by CompositePresentationEvent class. With EmployeeUpdatedEvent publisher (EmpoyeeViewModel) want to publish latest Employee Id which is an integer. So the parent class of EmployeeUpdatedEvent is CompositePresentationEvent which takes message (Payload of event) type as int.

   Hide Copy Code

   ```
   public class EmployeeUpdatedEvent : CompositePresentationEvent<int> { }
   ```

   EmpoyeeViewModel need to notify RequestViewModel whenever Employee Id gets change. For that EmpoyeeViewModel will publish an event called EmployeeUpdatedEvent with latest Employee Id.

2. Publisher and Subscriber object must be using global EventAggregator provided by DI (Dependency Injection) Container. So here we will use EventAggregator provided by Unity Container and it is initialized in UnityBootstrapper class. This is out of the box and we need to look into the code of ConfigureContainer method of UnityBootstrapper class. Below is the code taken from open source Prism Library Code from Codeplex.



```
protected virtual void ConfigureContainer()
{
    this.Logger.Log(Resources.AddingUnityBootstrapperExtensionToContainer, Category.Debug, Priority.Low);
    this.Container.AddNewExtension<UnityBootstrapperExtension>();

    Container.RegisterInstance<ILoggerFacade>(Logger);

    this.Container.RegisterInstance(this.ModuleCatalog);

    if (useDefaultConfiguration)
    {
        RegisterTypeIfMissing(typeof(IServiceLocator), typeof(UnityServiceLocatorAdapter), true);
        RegisterTypeIfMissing(typeof(IModuleInitializer), typeof(ModuleInitializer), true);
        RegisterTypeIfMissing(typeof(IModuleManager), typeof(ModuleManager), true);
        RegisterTypeIfMissing(typeof(RegionAdapterMappings), typeof(RegionAdapterMappings), true);
        RegisterTypeIfMissing(typeof(IRegionManager), typeof(RegionManager), true);
        RegisterTypeIfMissing(typeof(IEventAggregator), typeof(EventAggregator), true);
        RegisterTypeIfMissing(typeof(IRegionViewRegistry), typeof(RegionViewRegistry), true);
        RegisterTypeIfMissing(typeof(IRegionBehaviorFactory), typeof(RegionBehaviorFactory), true);
        RegisterTypeIfMissing(typeof(IRegionNavigationJournalEntry), typeof(RegionNavigationJournalEntry), false);
        RegisterTypeIfMissing(typeof(IRegionNavigationJournal), typeof(RegionNavigationJournal), false);
        RegisterTypeIfMissing(typeof(IRegionNavigationService), typeof(RegionNavigationService), false);
        RegisterTypeIfMissing(typeof(IRegionNavigationContentLoader), typeof(RegionNavigationContentLoader), true);
    }
}
```

So creation of global EventAggregator is already done as our ITHelpDeskBootstrapper class inherits from UnityBootstrapper.

3. EmpoyeeViewModel will act as a publisher so we need to inject EventAggregator object. Now in constructor of EmpoyeeViewModel we are injecting IEventAggregator and initializing a private field eventAggregator in constructor itself as shown below:

Hide Copy Code

```
public EmployeeViewModel(IEmployeeService employeeService, IEventAggregator  eventAggregator)
{
      this.eventAggregator = eventAggregator;
      this.employeeService = employeeService;
}
```

4. Employee Id needs to be published with updated value. So in the set block on "EmployeeId" property, we have added one line of code for publishing EmployeeUpdatedEvent after "if" block.

Hide Copy Code

```
public int EmployeeId
{
       get { return employeeId; }
       set
          {
           if (value != 0)
             {
                 employeeId = value;
                 fetchEmployeeDetail(employeeId);
                 RaisePropertyChanged("EmployeeId");
             }
           eventAggregator.GetEvent<EmployeeUpdatedEvent>().Publish(value);
          }
}
```

Note: It is bit tricky to decide the right place for publishing and subscribing the event. To realize it, replace *eventAggregator.GetEvent<EmployeeUpdatedEvent>().Publish(value);* code at first line (just above *if (value != 0)* line) inside set block. Then we can see EventAggregator will not be helping us to fix the issue which we were facing.

5. Any number of objects can subscribe published event (EmployeeUpdatedEvent). Let us see how to subscribe "EmployeeUpdatedEvent" event in RequestViewModel. In constructor of RequestViewModel we are subscribing for EmployeeUpdatedEvent. While subscribing we are passing a method (employeeUpdateHandler) which will be called when event will occur. In employeeUpdateHandler method we are calling fetchAllRequest method. As we can see from fetchAllRequest method, GetAllRequest method of EmployeeService is being called. Here we are getting updated empId in employeeUpdateHandler but we are not using it as we know that in EmployeeService, updated Employee Id will be available already. Below is the related code:

Hide Copy Code

```
public RequestViewModel(IEmployeeService employeeService,IEventAggregator eventAggregator)
{
   this.employeeService = employeeService;
   this.eventAggregator = eventAggregator;
   eventAggregator.GetEvent<EmployeeUpdatedEvent>().Subscribe(employeeUpdateHandler);
   fetchAllRequest();
}

private void employeeUpdateHandler(int empId)
{
   fetchAllRequest();
}

private void fetchAllRequest()
{
    List<Request> listOfRequest = employeeService.GetAllRequest();
    AllRequests = new ObservableCollection<Request>(listOfRequest);
}
```

GetAllRequest method of Employee service is calling GetAllRequest method of repository by passing CurrentEmployeeId.

Hide Copy Code

```
public List<Request> GetAllRequest()
    {
        return dataSource.GetAllRequest(CurrentEmployeeId);
    }
```

Now run the application and combobox will always show request Ids associated with Employee Id.

What is IActiveAware

IActiveAware is an interface which is used if we need to perform some action on Activation or Deactivation of a View. If we are following MVVM, we should implement this interface in corresponding ViewModel class.

This interface has two members: a property called as IsActive and an event IsActiveChanged:

Hide Copy Code

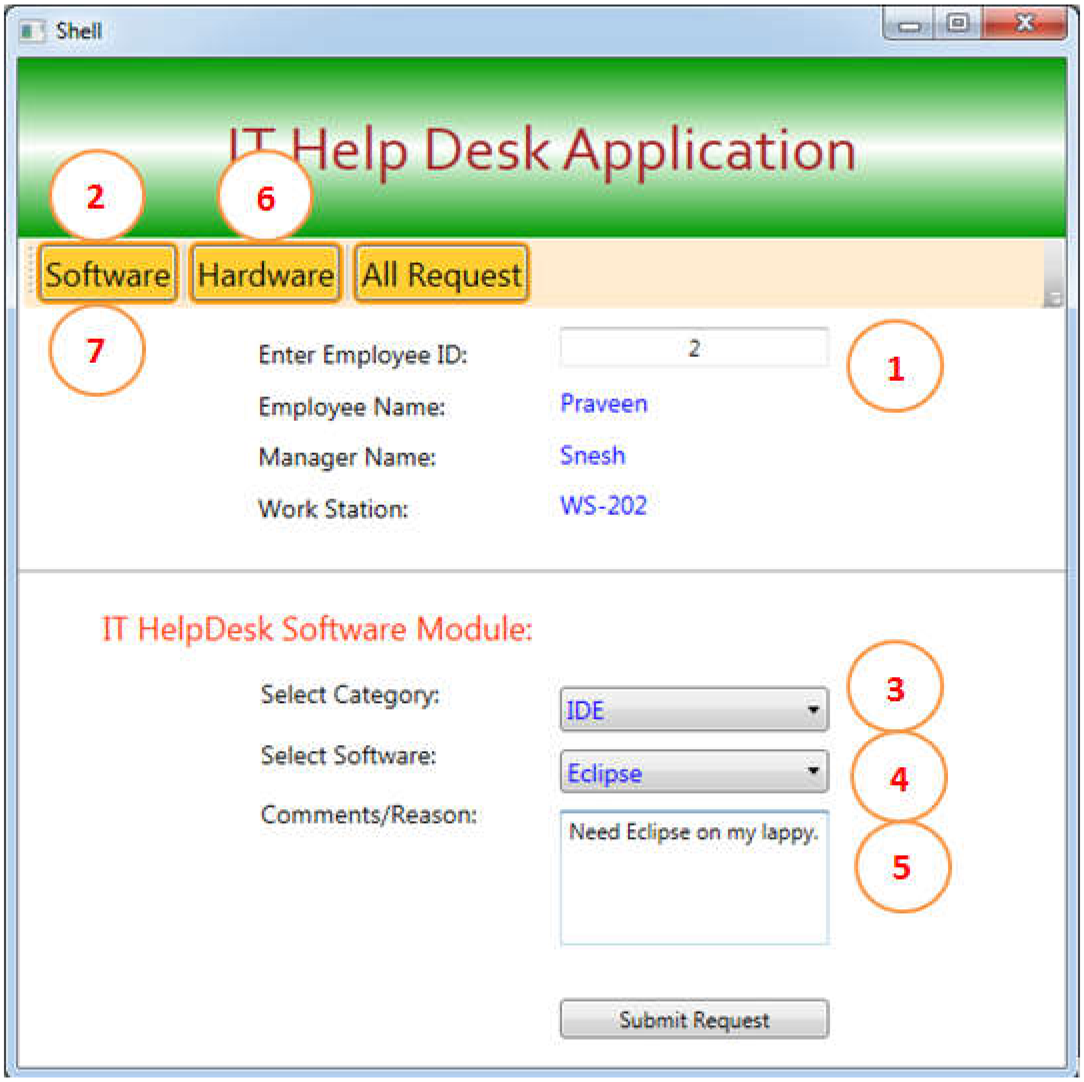
```
// Summary:
//     Interface that defines if the object instance is active and notifies when
//     the activity changes.
public interface IActiveAware
{
    // Summary:
    //     Gets or sets a value indicating whether the object is active.
    bool IsActive { get; set; }

    // Summary:
    //     Notifies that the value for Microsoft.Practices.Prism.IActiveAware.IsActive
    //     property has changed.
    event EventHandler IsActiveChanged;
}
```

When the View or corresponding ViewModel, gets active or deactivate, **RegionActiveAwareBehavior** class sets the new value of IsActive. We can use IsActiveChanged event to notify interested parties about Active status change.

New Requirement: We will understand IActiveAware by implementing a new scenario. Let's say we get a requirement that when user revisits any module, its controls must be refreshed. Those controls should not be holding old values. Those must be empty while revisiting again.

As of now if we make some selection in Software Module, and then Navigate to other module and revisit Software Module again we will see the old selected values in controls. Below screenshot is having steps to highlight this behavior in the demo we have developed so far:



Enter Employee ID: 2
Employee Name: Praveen
Manager Name: Snesh
Work Station: WS-202

IT HelpDesk Software Module:

Select Category: IDE
Select Software: Eclipse
Comments/Reason: Need Eclipse on my lappy.

In next section we will implement IActiveAware to meet this new requirement.

How to Use IActiveAware

First we will implement IActiveAware interface in SoftwareViewModel class. In order to reset the controls, we need to set bound properties with default values. Let us inherit SoftwareViewModel with IActiveAware interface and provide the implementation as shown in below code. Here **refreshSoftwareModule** method is being called from set block of IsActive property. In this method we are setting bound properties with default values.

Hide Shrink ▲ Copy Code

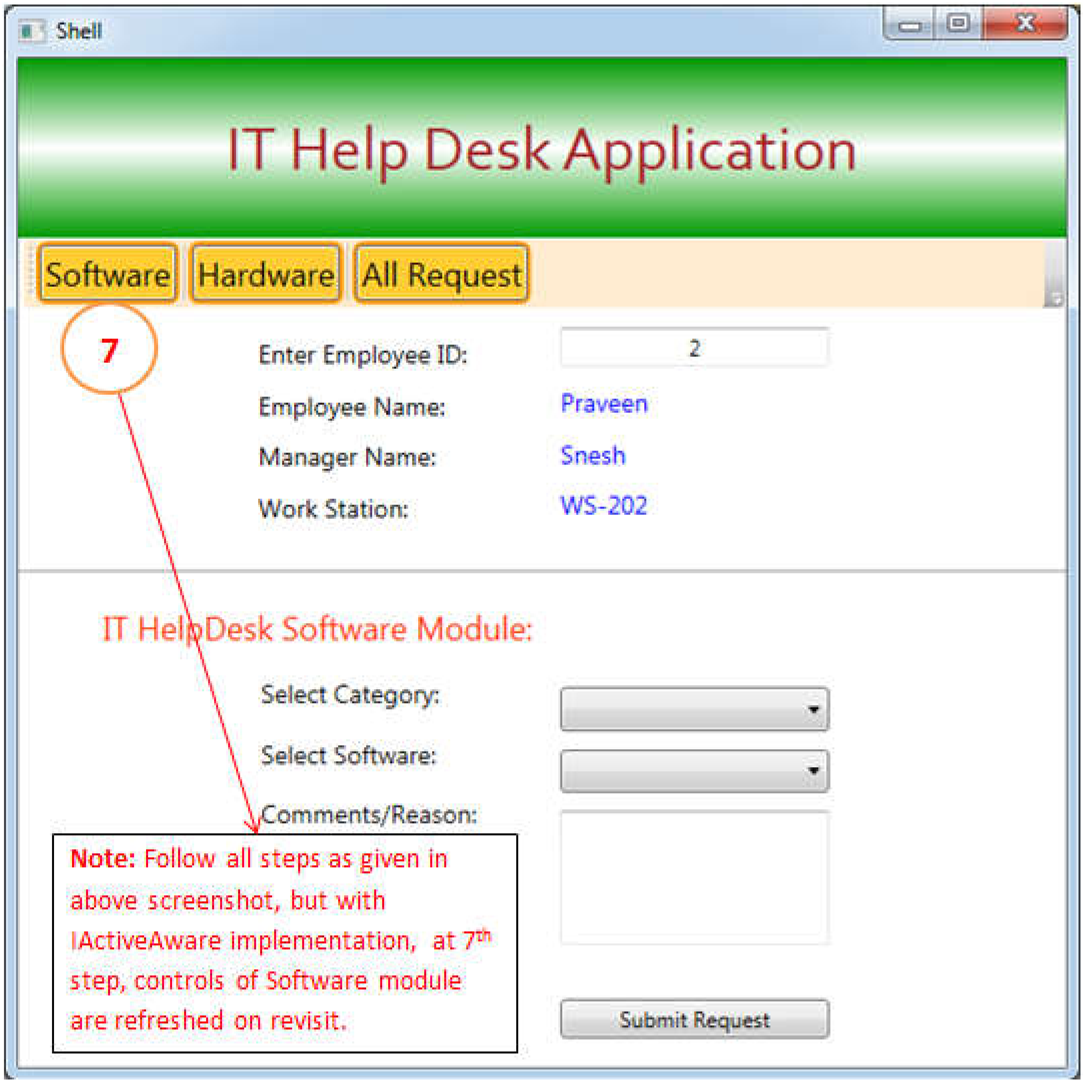
```
 region IActiveAware implementation
bool active;
public bool IsActive
{
    get
    {
        return active;
    }
    set
    {
        if (active != value)
        {
            active = value;
            if (active == true)
                refreshSoftwareModule();
        }
    }
}

private void refreshSoftwareModule()
{
    SelectedSoftwareCategory = null;
    SelectedSoftware ame = null;
    Comment = null;
}

public event EventHandler IsActiveChanged;

 endregion
]
```

Now run the application and follow the steps from 1 to 7 as per above screenshot. At 7th step we will see controls of Software Module are refreshed as per the given new requirement as shown below:



Note: Follow all steps as given in above screenshot, but with IActiveAware implementation, at 7th step, controls of Software module are refreshed on revisit.

In similar way we can implement IActiveAware interface for Hardware Module also.

Conclusion

In this part of the article series, we learned about how to use EventAggregator and IActiveAware in the Prism application. This is last part of the article series, hope you will find this series helpful. Your comments/suggestions and queries are most welcome. Thanks.

License

This article, along with any associated source code and files, is licensed under The Code Project Open License (CPOL)

Share

TWITTER FACEBOOK

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

How to a database implementation on this Prism Project?	**Member 13772199**	**22-Aug-18 0:42**
Great Article. This is the best article on WPF and Prism.	**krishnaabburi**	**29-Jun-18 10:06**
Very good article madam!	**Raghavendran Srinivasan**	**11-Dec-17 15:24**
NIce article	**lopezan**	**6-Dec-17 0:36**
Really a great article series	**Member 13142924**	**29-Jun-17 1:02**
Good Starting Point for Prism 4.0, useful for new versions too...	**earthStrapped**	**21-Jun-17 14:47**
Re: Good Starting Point for Prism 4.0, useful for new versions too...	Snesh Prajapati	21-Jun-17 16:13
Really a helpful article	**kidilams**	**24-Apr-17 19:22**
Re: Really a helpful article	Snesh Prajapati	25-Apr-17 4:36
Best article series on Prism	**@nkit Bajpai**	**18-Apr-17 4:42**
Re: Best article series on Prism	Snesh Prajapati	18-Apr-17 6:09
Re: Best article series on Prism	@nkit Bajpai	2-May-17 3:05
Re: Best article series on Prism	Snesh Prajapati	2-May-17 15:28
Re: Best article series on Prism	@nkit Bajpai	3-May-17 3:06
My vote of 5	**Santhk**	**5-Dec-15 7:43**
Re: My vote of 5	Snesh Prajapati	5-Dec-15 15:39
Re: My vote of 5	Santhk	5-Dec-15 19:01
Re: My vote of 5	Snesh Prajapati	5-Dec-15 20:09

Last Visit: 31-Dec-99 18:00 Last Update: 14-Sep-18 5:52 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.

Permalink | Advertise | Privacy | Cookies | Terms of Use | Mobile
Web06-2016 | 2.8.180910.1 | Last Updated 5 Dec 2015

Layout: fixed | fluid

Article Copyright 2015 by Snesh Prajapati
Everything else Copyright © CodeProject, 1999-2018