* 1. 

Hands-On Lab

ASP.NET MVC Global & Dynamic Action Filters

Lab version: 1.1.0

Last updated: 1/7/2011

Contents

[Overview 3](#_Toc282146885)

[Exercise 1: Creating a Global Action Filter 4](#_Toc282146886)

[Task 1 – Registering a Filter as Global 4](#_Toc282146887)

[Task 2 – Running the Application 5](#_Toc282146888)

[Exercise 2: Creating a Global Dynamic Filter 6](#_Toc282146889)

[Task 1 – Creating a Filter Provider 6](#_Toc282146890)

[Task 2 – Registering a Global Filter 10](#_Toc282146891)

[Task 3 – Running the Application 11](#_Toc282146892)

[Summary 12](#_Toc282146893)

Overview

* 1. **Note:** This Hands-on Lab assumes you have basic knowledge of **ASP.NET MVC and ASP.NET MVC Custom Action Filter**. If you have not used **ASP.NET MVC** before, we recommend you to go over **ASP.NET MVC Fundamentals** Hand-on Lab.

If you followed **ASP.NET MVC Custom Action Filter** Hands-on Lab before, you have applied an action filter as an attribute to a controller or action method. In this lab, you will learn how to apply a filter to all controllers in the application (global action filter) and how to apply them dynamically.

* 1. Similarly than in **ASP.NET MVC Custom Action Filter**, in this Hand-on Lab, you will be working with actions logging, writing events in a database table and showing the result in a dedicated controller and view.
  2. In the first exercise you will apply global filters using code residing in the Global.asax.cs source file. In the second exercise you will learn how dynamically define which actions will get logged and which not, using a filter provider.
  3. In this Hands-on Lab, you will learn how to:
  + Create a global filter to apply the logging of actions to all controllers in application.
  + Create a filter provider to apply logging to some controllers or actions, excluding others.

# System Requirements

* 1. You must have the following items to complete this lab:
  + ASP.NET and ASP.NET MVC 3
  + Visual Studio 2010 Express
  + SQL Server Database (Express edition or above)
    1. **Note:** You can install the previous system requirements by using the Web Platform Installer 3.0: <http://go.microsoft.com/fwlink/?LinkID=194638>.

# Setup

#### Installing Code Snippets

* 1. For convenience, much of the code you will be managing along this lab is available as Visual Studio code snippets. To install the code snippets run **.\Source\Assets\CodeSnippets.vsi** file.

# Exercises

* 1. This Hands-On Lab is comprised by the following exercises:
  2. Exercise 1: Creating a Global Action Filter
  3. Exercise 2: Creating a Global Dynamic Filter
  4. Estimated time to complete this lab: **30 minutes**.
  5. **Note:** Each exercise is accompanied by an **End** folder containing the resulting solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

# Next Step

Exercise 1: Creating a Global Action Filter

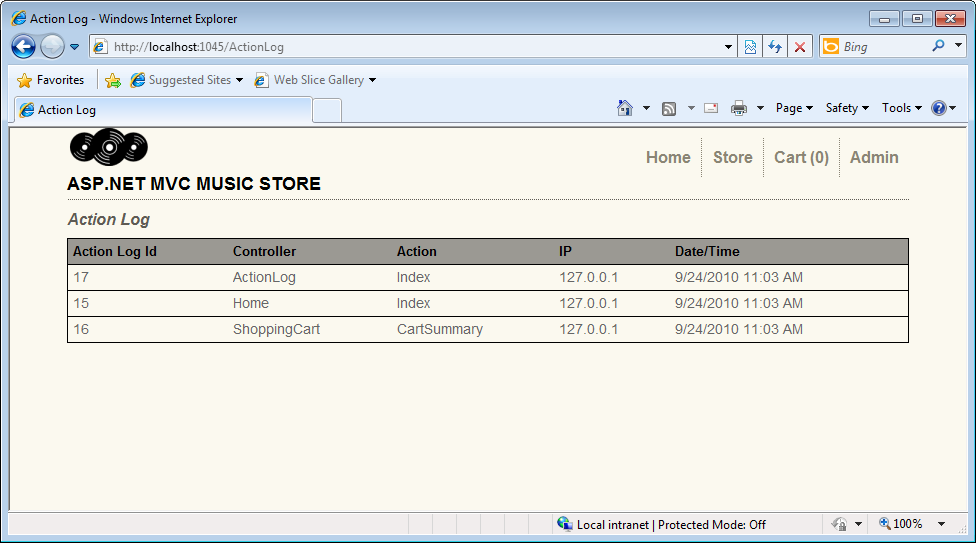
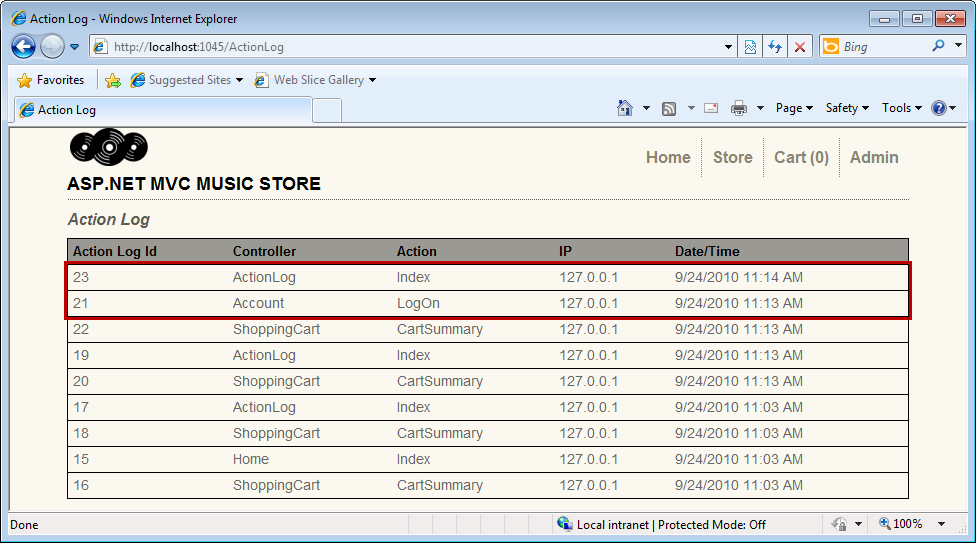
Exercise 1: Creating a Global Action Filter

* 1. In this exercise, you will learn how to add a global filter that will perform certain logic before displaying each view. In order to do that, you will register as global the custom filter for application logging created in ASP.NET MVC Custom Action Filter Hands-on Lab.

Task 1 – Registering a Filter as Global

* 1. In this task, you will register a global filter to manage your site logging at a global level. In order to do that, you will add **ActionLogFilterAttribute** filter into MVC **GlobalFilterCollection** at **Global.asax.cs**.
  2. Open the begin solution **MvcMusicStore.sln** at Source\Ex01-Global Action Filter\Begin.
  3. Open **Global.asax.cs** at project root. In the method **RegisterGlobalFilters** , add a new security filter to the **filters** collection:
     1. C#
     2. public static void RegisterGlobalFilters(GlobalFilterCollection filters)
     3. {
     4. filters.Add(new HandleErrorAttribute());
     5. **filters.Add(new ActionLogFilterAttribute());**
     6. }
     7. **Note:** the previous implementation of **RegisterGlobalFilters** (with the **HandleErrorAttribute** filter only) is the one included in an MVC 3 application by default. In this task, you are only adding an additional filter, **ActionLogFilterAttribute.**

Task 2 – Running the Application

* 1. Press **F5** to run the application.
  2. Browse to **/ActionLog** View. You will see that the initial activities have been tracked.
     1. 
     2. Figure 1
     3. ActionLogView: Initial site activity was tracked
  3. Click on **Admin** to load the **LogOn** View.
  4. Browse **/ActionLog** view again to see the administration site tracked. If you see the same table as before, please refresh the page.
     1. 
     2. Figure 2
     3. ActionLogView: Recent activities logged

# Next Step

Exercise 2: Creating a Global Dynamic Filter

Exercise 2: Creating a Global Dynamic Filter

* 1. In this exercise, you will learn how to create a global dynamic filter that will get executed or not depending on the context. This means that you could set any filter as global, but use it only for the controllers or actions that satisfy a certain condition. For that purpose you will implement **GetFilters** method in your custom filter provider to add your own logic behind the filter retrieval.

Task 1 – Creating a Filter Provider

* 1. In this task, you learn how custom Filter Providers manage the global behavior of any MVC Filter. You will create a custom Filter Provider that stores the controller actions to log and applies the filter when an action is in the list.
  2. Open **Begin** solution from **\Ex02 – Global Dynamic Filter\Begin**.
  3. Create a new C# class in **Filters** folder and rename it **ActionLogFilterProvider.cs**.
  4. Add the reference to the **System.Web.Mvc** namespace.
     1. C#
     2. using System;
     3. using System.Collections.Generic;
     4. using System.Linq;
     5. using System.Web;
     6. **using System.Web.Mvc;**
     7. namespace MvcMusicStore.Filters
     8. {
     9. public class ActionLogFilterProvider
     10. {
     11. }
     12. }
  5. Make **ActionLogFilterProvider** implement the interface **IFilterProvider**.
     1. C#

public class ActionLogFilterProvider **: IFilterProvider**

* 1. Define the internal class **ControllerAction** inmediatly after **ActionLogFilterProvider** class definition that will have a controller and action names as the members.
     1. (Code Snippet – ASP.NET Global and Dynamic Action Filters – Ex2 ControllerAction – CSharp)
     2. C#
     3. using System;
     4. using System.Collections.Generic;
     5. using System.Linq;
     6. using System.Web;
     7. using System.Web.Mvc;
     8. namespace MvcMusicStore.Filters
     9. {
     10. public class ActionLogFilterProvider
     11. {
     12. }
     13. **internal class ControllerAction**
     14. **{**
     15. **internal string ControllerName { get; set; }**
     16. **internal string ActionName { get; set; }**
     17. **}**
     18. }
  2. Then define a list of **ControllerAction** elements as new **ActionLogFilter** property:
     1. (Code Snippet – ASP.NET Global and Dynamic Action Filters – Ex2 Actions List – CSharp)
     2. C#
     3. using System;
     4. using System.Collections.Generic;
     5. using System.Linq;
     6. using System.Web;
     7. using System.Web.Mvc;
     8. namespace MvcMusicStore.Filters
     9. {
     10. public class ActionLogFilterProvider : IFilterProvider
     11. {
     12. **private IList<ControllerAction> actions = new List<ControllerAction>();**
     13. }
     14. internal class ControllerAction
     15. {
     16. internal string ControllerName { get; set; }
     17. internal string ActionName { get; set; }
     18. }
     19. }
  3. Create an **Add** method to **ActionLogFilterProvider** class to store in the list all the controllers and actions that will be logged:
     1. (Code Snippet – ASP.NET Global and Dynamic Action Filters – Ex2 Add Method – CSharp)
     2. C#
     3. …
     4. private IList<ControllerAction> actions = new List<ControllerAction>();
     5. **public void Add(string controllername, string actionname)**
     6. **{**
     7. **actions.Add(new ControllerAction() { ControllerName = controllername, ActionName = actionname });**
     8. **}**
     9. …
  4. Implement in **ActionLogFilterProvider** class the method **GetFilters** that will read the list of controller actions and evaluate which ones apply to the filter.
     1. (Code Snippet – ASP.NET Global and Dynamic Action Filters – Ex2 GetFilters – CSharp)
     2. C#
     3. …
     4. **public IEnumerable<Filter> GetFilters(ControllerContext controllerContext, ActionDescriptor actionDescriptor)**
     5. **{**
     6. **foreach (ControllerAction action in actions)**
     7. **if ((action.ControllerName == actionDescriptor.ControllerDescriptor.ControllerName || action.ControllerName == "\*")**
     8. **&& (action.ActionName == actionDescriptor.ActionName || action.ActionName == "\*")) {**
     9. **yield return new Filter(new ActionLogFilterAttribute(), FilterScope.First, null);**
     10. **break;**
     11. **}**
     12. **yield break;**
     13. **}**

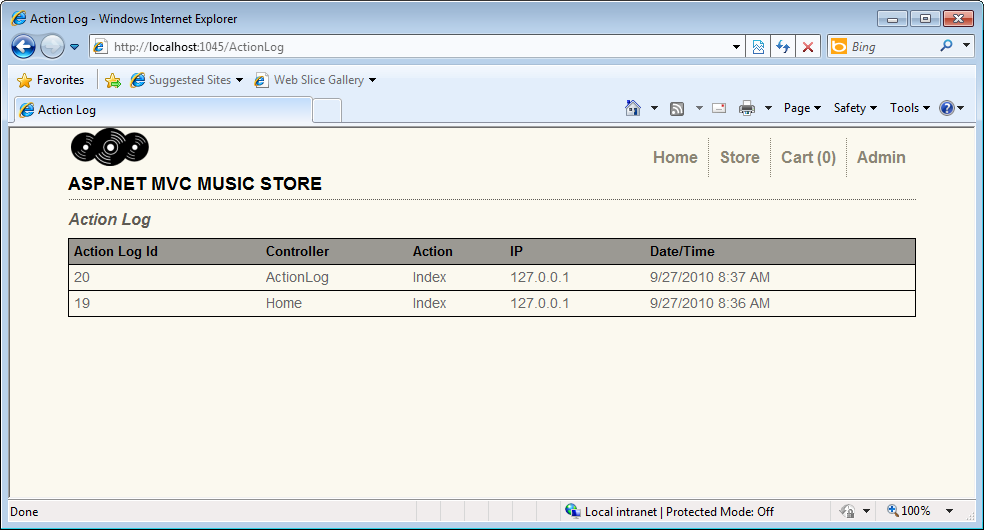
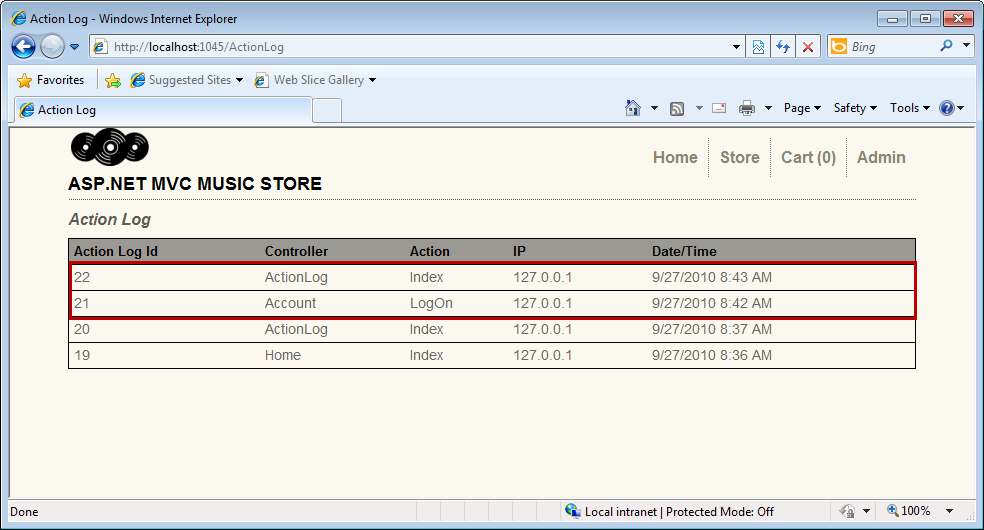
**Note:** This implementation of **GetFilters** method checks if the controller name and the action name are in the list and returns a new instance of the custom action filter class **ActionLogFilterAttribute**. You could also use the **Controller Context** parameter to add more complex logic to this method.

* + 1. The **"\*"** will be used as a wildcard for not filtering by a controller name or an action name. You can find **ControllerContext** class reference in this article at [msdn](http://msdn.microsoft.com/en-us/library/dd492673.aspx).
  1. Your filter provider class should finally look like this:
     1. C#
     2. using System;
     3. using System.Collections.Generic;
     4. using System.Linq;
     5. using System.Web;
     6. using System.Web.Mvc;
     7. namespace MvcMusicStore.Filters
     8. {
     9. public class ActionLogFilterProvider : IFilterProvider
     10. {
     11. private IList<ControllerAction> actions = new List<ControllerAction>();
     12. public void Add(string controllername, string actionname)
     13. {
     14. actions.Add(new ControllerAction() { ControllerName = controllername, ActionName = actionname });
     15. }
     16. public IEnumerable<Filter> GetFilters(ControllerContext controllerContext, ActionDescriptor actionDescriptor)
     17. {
     18. foreach (ControllerAction action in actions)
     19. if ((action.ControllerName == actionDescriptor.ControllerDescriptor.ControllerName || action.ControllerName == "\*")
     20. && (action.ActionName == actionDescriptor.ActionName || action.ActionName == "\*"))
     21. {
     22. yield return new Filter(new ActionLogFilterAttribute(), FilterScope.First, null);
     23. break;
     24. }
     25. yield break;
     26. }
     27. }
     28. internal class ControllerAction
     29. {
     30. internal string ControllerName { get; set; }
     31. internal string ActionName { get; set; }
     32. }
     33. }
     34. In the following step you will add this provider to the global filter provider collection.

Task 2 – Registering a Global Filter

* 1. In this task, you will register a custom filter provider as global in the **Global.asax.cs**.
  2. Add an instance of your filter provider into **MVC FilterProvider** collection, in the method **RegisterGlobalFilters** at **Global.asax.cs** class:
     1. (Code Snippet – ASP.NET Global and Dynamic Action Filters – Ex2 LogProviders – CSharp)
     2. C#
     3. public static void RegisterGlobalFilters(GlobalFilterCollection filters)
     4. {
     5. filters.Add(new HandleErrorAttribute());
     6. **ActionLogFilterProvider provider = new ActionLogFilterProvider();**
     7. **provider.Add("Account", "LogOn");**
     8. **provider.Add("Store", "\*");**
     9. **provider.Add("\*", "Index");**
     10. **FilterProviders.Providers.Add(provider);**
     11. }
     12. **Note: ActionLogFilterProvider** Add method receives the controller name and the action name parameters of the elements that will perform logging. In this solution the elements are:
     13. **1. “Account” ,“LogOn”:** By sending the controller and the view you add logging to a specific functionality
     14. **2. “Store”, “\*”:** Using the wildcard “\*” at action name, All the actions from “Store” are logged
     15. **3. “\*”, “index”:** Using the wildcard “\*” at controller name, All actions “index” at any controller are logged

Task 3 – Running the Application

* 1. In this task, you will check that the controllers added to the filter provider list are actually performing the log.
  2. Press **F5** to run the application.
  3. Browse to **/ActionLog** to see the initial status of the log:
     1. 
     2. Figure 3
     3. ActionLogView Initial state
  4. Browse **/StoreManager** to see the **LogOn** view, which is selected for the log filter.
  5. Browse **/Account/Register** view, which is not selected for the log filter.
  6. Browse **/ActionLog** and press **F5** to see the logged information:
     1. 
     2. Figure 4
     3. ActionLogView: logged activity only for the selected views

# Next Step

1. Summary

Summary

* 1. By completing this Hands-On Lab you have learned how to use Global Action Filters:
  + Applying MVC Filter model
  + Reusing ActionLogAttribute filter
  + Using RegisterGlobalFilters
  + Using IFilterProvider
  + Implementing and using ActionLogFilterProvider