* 1. 

Hands-On Lab

ASP.NET MVC Custom Action Filter

* 1. Lab version: 1.1.0
  2. Last updated: 1/6/2011

Contents

[Overview 3](#_Toc282066973)

[Exercise 1: Logging Actions 4](#_Toc282066974)

[Task 1 – Creating a Custom Filter to Catch a Controller’s Request 5](#_Toc282066975)

[Task 2 – Injecting a Code Interceptor into the Store Controller Class 8](#_Toc282066976)

[Task 3 – Running the Application 9](#_Toc282066977)

[Summary 10](#_Toc282066978)

Overview

* 1. **Note:** This Hands-on Lab assumes you have basic knowledge of **ASP.NET MVC.** If you have not used **ASP.NET MVC** before, we recommend you to go over **ASP.NET MVC Fundamentals** Hand-on Lab.
  2. ASP.NET MVC provides Action Filters for performing logic either before an action method is called or after its run. Action Filters are custom attributes that provide a declarative means to add pre-action and post-action behavior to controller action methods.
  3. In this Hands-on Lab you will create a custom action filter attribute into MVC Music Store solution to catch controller’s requests and log site actvity into a database table. You will be able to add your logging filter by injection to any controller or action. Finally, you will see the log view that shows a visit list.

# Objectives

* 1. In this Hands-On Lab, you will learn how to:
  + Create a custom action filter attribute to extend filtering capabilities
  + Apply a custom filter attribute by injection to a specific level

# System Requirements

* 1. You must have the following items to complete this lab:
  + ASP.NET and ASP.NET MVC 3 Release Candidate
  + 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: Logging actions
  3. Estimated time to complete this lab: 20 minutes.
  4. **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: Logging Actions

Exercise 1: Logging Actions

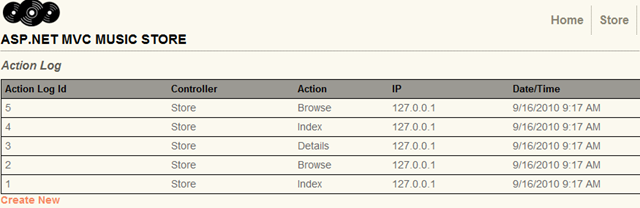
1. In this exercise, you will learn how to create a custom action log filter by using MVC3 Filter Providers. For that purpose you will apply a logging filter to the MusicStore site that will record all the activities in the selected controllers.
2. The filter will extend **ActionFilterAttributeClass** and override **OnActionExecuting** method to catch each request and then perform the logging actions. The context information about HTTP requests, executing methods, results and parameters will be provided by MVC **ActionExecutingContext** class**.**

# About MVC Music Store Application logging feature

This Music Store solution has a new data model table for site logging, **ActionLog**, with the following fields: Name of the controller that received a request, Called action, Client IP and Time stamp.

* 1. A new table, ActionLog was added.
     The fields are: ActionLogId, Controller, Action, IP, DateTime
  2. Figure 1
  3. Data model. ActionLog table.

The solution provides an MVC View for the Action log that can be found at **MvcMusicStores/Views/ActionLog**:

* 1. 
  2. Figure
  3. Action Log view

With this given structure, all the work will be focused on interrupting controller’s request and performing the logging by using custom filtering.

Task 1 – Creating a Custom Filter to Catch a Controller’s Request

* 1. In this task you will create a custom filter attribute class that will contain the logging logic. For that purpose you will extend MVC **ActionFilterAttribute** Class and implement the interface **IActionFilter**.
  2. **Note: ActionFilterAttribute** is the base class for all the attribute filters. It provides the following methods to execute a specific logic after and before controller action’s execution:
  3. **- OnActionExecuting(ActionExecutedContext** filterContext**)**

Just before the action method is called

**- OnActionExecuted(ActionExecutingContext** filterContext**):**

* 1. After the action method is called and before the result is executed (before view render).
  2. **- OnResultExecuting(ResultExecutingContext** filterContext**):**
  3. Just before the result is executed (before view render).
  4. **- OnResultExecuted(ResultExecutedContext** filterContext**):**
  5. After the result is executed (after the view is rendered).
  6. By overriding any of these methods into a derived class, you can execute your own filtering code.
  7. Open the begin solution **MvcMusicStore.sln** at **Source\Ex01-Logging Actions\Begin**
  8. Create a new folder **Filters** at project root, which will include all the custom filters.
  9. Add a new C# class into the **Filters** folder and rename it to **ActionLogFilterAttribute.cs**
  10. Open **ActionLogFilterAttribute.cs** and add a reference to **System.Web.Mvc** and the **MvcMusicStore.Models** 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. **using MvcMusicStore.Models;**
  11. Inherit the **ActionLogFilterAttribute** class from **ActionFilterAttribute** and then make **ActionLogFilterAttribute** class implement **IActionFilter** interface.
      1. C#
      2. ...
      3. namespace MvcMusicStore.Filters
      4. {
      5. public class ActionLogFilterAttribute : **ActionFilterAttribute**, **IActionFilter**
      6. {
      7. ...
  12. Make **ActionLogFilterAttribute** class override the method **OnActionExecuting,** where you will write the logging code. After that, your class should look like the following:
      1. *(Code Snippet – ASP.NET MVC 3 Custom Action Filters – Ex1 Logging Actions – 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. using MvcMusicStore.Models;
      9. namespace MvcMusicStore.Filters
      10. {
      11. public class ActionLogFilterAttribute : ActionFilterAttribute, IActionFilter
      12. {
      13. **public override void OnActionExecuting(ActionExecutingContext filterContext)**
      14. **{**
      15. **MusicStoreEntities storeDB = new MusicStoreEntities();**
      16. **ActionLog log = new ActionLog()**
      17. **{**
      18. **Controller = filterContext.ActionDescriptor.ControllerDescriptor.ControllerName,**
      19. **Action = filterContext.ActionDescriptor.ActionName,**
      20. **IP = filterContext.HttpContext.Request.UserHostAddress,**
      21. **DateTime = filterContext.HttpContext.Timestamp**
      22. **};**
      23. **storeDB.AddToActionLogs(log);**
      24. **storeDB.SaveChanges();**
      25. **base.OnActionExecuting(filterContext);**
      26. **}**
      27. }
      28. }

**Note:**  **OnActionExecuting** methodis using **Entity Framework** to add a new ActionLog register. It creates and fills a new entity instance with the context information from **filterContext**.

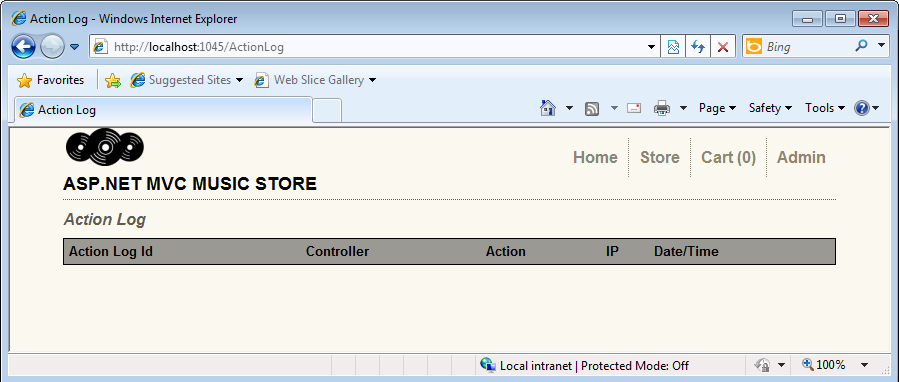
You could read more about **ControllerContext** class at [msdn](http://msdn.microsoft.com/en-us/library/system.web.mvc.controllercontext.aspx).

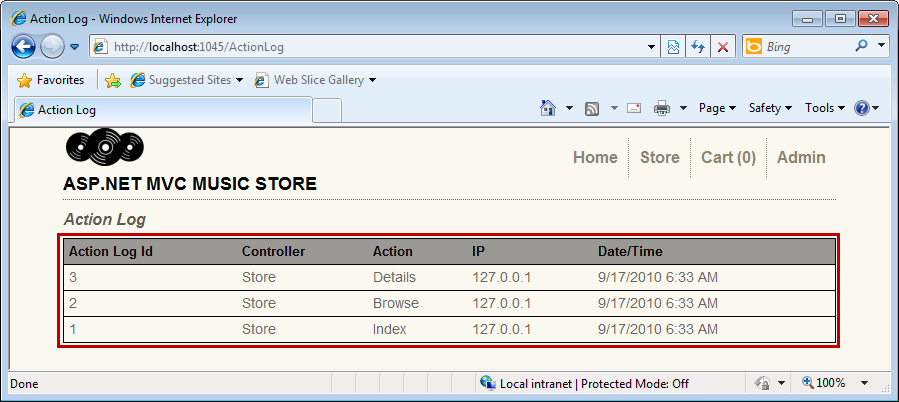
Task 2 – Injecting a Code Interceptor into the Store Controller Class

* 1. In this task you will add the custom filter by injecting it to all controller classes and controller actions that will be logged. For the purpose of this exercise, the Store Controller class will have a log.
  2. The method **OnActionExecuting** from **ActionLogFilterAttribute** custom filter runs when an injected element is called.
  3. It is also possible to intercept a specific controller method.
  4. Open the **StoreController** at **MvcMusicStore\Controllers** and add a reference to the **Filters** namespace:
     1. C#
     2. …
     3. using MvcMusicStore.ViewModels;
     4. using MvcMusicStore.Models;
     5. **using MvcMusicStore.Filters;**
     6. namespace MvcMusicStore.Controllers
     7. {
     8. …
  5. Inject the custom filter **ActionLogFilter** into **StoreController** class.
     1. C#
     2. …
     3. using MvcMusicStore.ViewModels;
     4. using MvcMusicStore.Models;
     5. using MvcMusicStore.Filters;
     6. namespace MvcMusicStore.Controllers
     7. {
     8. **[ActionLogFilter]**
     9. public class StoreController : Controller
     10. {
     11. MusicStoreEntities storeDB = new MusicStoreEntities();
     12. //
     13. // GET: /Store/
     14. public ActionResult Index()
     15. {
     16. // Create list of genres
     17. …
     18. **Note:** When a filter is injected into a controller class, all its actions are also injected. If you would like to apply the filter only for a set of actions, you would have to inject **[ActionLogFilter]** to each one of them:
     19. **[ActionLogFilter]**
     20. public ActionResult Index()
     21. {
     22. …
     23. }
     24. …
     25. **[ActionLogFilter]**
     26. public ActionResult Browse(string genre)
     27. {
     28. …
     29. }

Task 3 – Running the Application

* 1. In this task, you will test that the logging filter is working. You will start the application and visit the store, and then you will check logged activities.
  2. Press **F5** to run the application.
  3. Browse to **/ActionLog** to see log view initial state:



* + 1. Figure
    2. Log tracker status before page activity
  1. Browse to **/Store** and perform some actions there, like browsing an album detail.
  2. Browse to **/ActionLog** and if the log is empty press **F5** to refresh the page. Check that your visits were tracked:
     1. 
     2. Figure
     3. Action log with activity logged

# Next Step

Summary

Summary

* 1. By completing this Hands-On Lab you have learned how to extend an action filter to execute custom actions. You have also learned how to inject any filter to your page controllers. The following concepts were used:
  + How to create Custom Action filters with MVC ActionFilterAttribute class
  + How to inject filters into MVC controllers