Lab 3: Content Auditing

**Lab Time: 45 Minutes**

**Lab Directory: ECM401.Auditing**

**Lab Overview:**

In this lab, you will create a SharePoint feature to enable item-level auditing for any SharePoint list or document library. The built-in auditing support provided by Windows SharePoint Services must be enabled programmtically, so you will start by writing a feature receiver that enables auditing for all lists in the site collection. Then you will refine the feature to limit auditing to a specific list or document library.

# Exercise 1: Create the SharePoint Feature project

1. Start by creating a **SharePoint Feature** project named **ECM401.Auditing** using the **SharePoint Feature** project template. When completing the **Feature Details** dialog, be sure to select the **Site** scope for the feature as shown below.

Note: You are setting the feature scope to **"Site"** so that it can be activated for all sites in the site collection. If you open the generated **feature.xml** file, you will notice that the ReceiverAssembly and ReceiverClass attributes are automatically set by the project template to reference the project assembly and the generated FeatureReceiver class.

1. Open the **FeatureReceiver.cs** file for editing. The generated **FeatureReceiver** class allows you to implement methods of the abstract base class **SPFeatureReceiver** .
2. Next, you will handle the **FeatureActivated** event. Replace the **FeatureActivated** method with the following code snippet.

Code Snippet: 'Auditing - FeatureActivated'

public override void FeatureActivated(SPFeatureReceiverProperties properties)  
{  
 SPSite site = properties.Feature.Parent as SPSite;  
 if (site != null)  
 {  
 // Enable auditing for all items in the site collection.  
 site.Audit.AuditFlags = SPAuditMaskType.All;  
 site.Audit.Update();  
  
 // Modify the top-level website title to indicate that auditing is on.  
 // Save the title in the property bag for the site.  
 SPWeb web = site.RootWeb;  
 web.Properties[TitleKey] = web.Title;  
 web.Properties.Update();  
 web.Title = string.Format("{0} - Audited", web.Title);  
 web.Update();  
 }  
}

1. Similarly, replace the generated **FeatureDeactivating** method with the following code snippet.

Code Snippet: 'Auditing - FeatureDeactivating'

public override void FeatureDeactivating(SPFeatureReceiverProperties properties)  
{  
 SPSite site = properties.Feature.Parent as SPSite;  
 if (site != null)  
 {  
 // Disable auditing.  
 site.Audit.AuditFlags = SPAuditMaskType.None;  
  
 // Restore the original title.  
 SPWeb web = site.RootWeb;  
 web.Title = web.Properties[TitleKey];  
 web.Update();  
 }  
}

1. Save the file and build the project.
2. Test your work by navigating to **http://localhost** . Select **Site Settings** from the **Site Actions** dropdown menu and then click the **Site Collection Features** link.
3. On the **Site Collection Features** page, scroll down to the **ECM401.Auditing** feature and click the **Activate** button.
4. The root web title changes to a bracketed string indicating that the feature was successfully activated and that auditing is now enabled.
5. Verify this by navigating back to the **Site Settings** page and then click on **Configure Audit Settings** from the **Site Collection Administration** section. The **Configure Audit Settings** page should now have all items checked as shown below.
6. To view the audit log reports, you have to activate the MOSS reporting feature. Navigate to the **Site Collection Features** page and scroll down until you see the **Reporting** feature. Click the **Activate** button and then return to the **Site Settings** page. In the **Site Collection Administration** section, there is now a link for **Audit log reports** . Click this link to open the **View Auditing Reports** page. Click any of the links to view a report in Microsoft Excel.

# Exercise 2: Display the Audit Log in WSS

Since the Excel reports are available only in MOSS, another technique for viewing audit entries is to display them manually using a custom application page. In this exercise, you will extend the auditing feature to include a custom action that lets site administrators view the audit log. The custom action will be hidden from non-administrators.

1. From the Visual Studio **Solution Explorer** , open the **elements.xml** file for editing and enter the following code inside the <Elements> tag.

XML Snippet: 'Auditing - Custom Action'

<CustomAction  
 Id="SiteActions\_ViewAuditLog"  
 GroupId="SiteActions"  
 Location="Microsoft.SharePoint.StandardMenu"  
 Sequence="1001"  
 Title="View Audit Log"  
 Description="Display the audit log for this site collection."  
 RequireSiteAdministrator="TRUE"  
 ImageUrl="/\_layouts/images/LTTXTBOX.GIF">  
 <UrlAction Url="~sitecollection/\_layouts/ECM401.Auditing/AuditLog.aspx"/>  
 </CustomAction>

1. Next, you will create a custom application page to display the audit log entries.
2. Add a new folder to the project under the **12\TEMPLATE** folder called **LAYOUTS** . Within that folder, add another subfolder called **ECM401.Auditing** as shown below.
3. Right-click the folder and select **Add -> New Item...** from the context menu and create a new **TEXT** file with the name **AuditLog.aspx**

Note: Although you selected a text file type, Visual Studio will still open the file as a web page because of the **.ASPX** extension.

1. Enter the following code for the page. Since Visual Studio code snippets don't work well for ASPX pages, you will have to copy and paste.

<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>  
<%@ Page Language="C#" MasterPageFile="~/\_layouts/application.master" Inherits="Microsoft.SharePoint.WebControls.LayoutsPageBase" %>  
<%@ Import Namespace="Microsoft.SharePoint" %>  
<%@ Import Namespace="System.Data" %>  
<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"  
Assembly="Microsoft.SharePoint, version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>  
  
<script runat="server">  
// \*\*\* code will go here \*\*\*  
</script>  
<asp:Content ID="contentMain" ContentPlaceHolderID="PlaceHolderMain" runat="server">  
<asp:Button ID="btnRefresh" runat="server" Text="Refresh" OnClick="btnRefresh\_Click" />  
<asp:Button ID="btnClear" runat="server" Text="Clear Log" OnClick="btnClear\_Click" />  
<hr />  
<SharePoint:SPGridView ID="grid" runat="server" AutoGenerateColumns="False" Width="100%">  
<AlternatingRowStyle CssClass="ms\_alternating" />  
</SharePoint:SPGridView>  
</asp:Content>  
  
<asp:Content ID="contentTitle" ContentPlaceHolderID="PlaceHolderPageTitle" runat="server">Audit Log</asp:Content>  
<asp:Content ID="contentTitleTitle" ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">Custom display of audit log entries.</asp:Content>

Note: To make your application page look like the other SharePoint landing pages, you will inherit from **LayoutsPageBase** and use the same **application.master** used by the other pages. Since the page is deployed to every Web front-end in the farm by an administrator, you can enter inline code.

1. This code sets up the page elements, which are an instance of the SPGridView control and two buttons - one to refresh the view and another to clear the log entries.Note: Clearing the log entries is a destructive action and might require more thought in a real application.
2. Now you are ready to add some code inside the script block at the top of the page. The first routine you add will lock down the page so that only administrators can view it.

protected override bool RequireSiteAdministrator {  
get { return true; }  
}

1. Next you will handle the page load event to create a **DataTable** that can be bound to the **SPGridView** control.

Code Snippet: 'Auditing - AuditLog OnLoad'

protected override void OnLoad(EventArgs e) {  
 // Access the site collection  
 SPSite siteCollection = SPContext.Current.Site;  
 SPWeb site = SPContext.Current.Web;  
  
 // Create an Audit Query object.  
 SPAuditQuery query = new SPAuditQuery(siteCollection);  
 SPAuditEntryCollection entries = siteCollection.Audit.GetEntries(query);  
  
 // Loop through the entries to create a data table  
 DataTable table = new DataTable();  
 table.Columns.Add("User", typeof(string));  
 table.Columns.Add("Source", typeof(string));  
 table.Columns.Add("Type", typeof(string));  
 table.Columns.Add("ID", typeof(string));  
 table.Columns.Add("Event", typeof(string));  
 table.Columns.Add("Date", typeof(DateTime));  
  
 DataRow row;  
  
 foreach (SPAuditEntry entry in entries)  
 {  
 row = table.Rows.Add();  
 row["User"] = site.SiteUsers.GetByID(entry.UserId).Name;  
 row["Source"] = entry.DocLocation;  
 row["Type"] = entry.ItemType.ToString();  
 row["ID"] = entry.ItemId.ToString();  
 row["Event"] = entry.Event;  
 row["Date"] = entry.Occurred.ToLocalTime();  
 }  
  
 // Bind the columns to matching fields in the table.  
 AddGridField("User","User");  
 AddGridField("Source","Source");  
 AddGridField("Type","Type");  
 AddGridField("ID","ID");  
 AddGridField("Event","Event");  
 AddGridField("Date","Date").ControlStyle.Width = new Unit(120);  
  
 // Bind the datasource to the grid.  
 grid.AutoGenerateColumns = false;  
 grid.DataSource = table.DefaultView;  
 grid.DataBind();  
 grid.AllowSorting=true;  
 grid.HeaderStyle.Font.Bold = true;  
  
}  
  
// Adds a bound field to the grid.  
private SPBoundField AddGridField(string headerText, string fieldName)  
{  
 SPBoundField field = new SPBoundField();  
 field.HeaderText = headerText;  
 field.DataField = fieldName;  
 grid.Columns.Add(field);  
 return field;  
}

1. Finally, add the button click event handlers.

Code Snippet: 'Auditing - AuditLog ButtonClick'

// Handle the refresh button click.  
protected void btnRefresh\_Click(object sender, EventArgs e)  
{  
 Response.Redirect(Request.RawUrl);  
}  
  
// Handle the clear button click.  
protected void btnClear\_Click(object sender, EventArgs e)  
{  
 SPSite siteCollection = SPContext.Current.Site;  
 siteCollection.Audit.DeleteEntries(DateTime.Now.ToLocalTime().AddDays(1));  
 siteCollection.Audit.Update();  
 Response.Redirect(Request.RawUrl);  
}

1. To test your work, build the project and then navigate to the **Site Settings** page and select the **Site Collection Features** link.
2. Deactivate and then re-activate the **ECM401.Auditing** Feature.
3. From the **Site Actions** menu, click the new **View Audit Log** link. You page should resemble the one shown below.

# Exercise 3: Enable Item-Level Auditing

In this exercise, you will enable site users to display auditing event entries for individual list items. This will require a custom menu item added to the drop-down menu associated with each item. When the user selects the menu command, another custom application page will be displayed showing audit events for that item.

1. Start by adding another custom action to the **elements.xml** file.

XML Snippet: 'Auditing - ECB CustomAction'

<CustomAction  
 Id="ECB\_ViewAuditLog"  
 RegistrationType="List"  
 RegistrationId="101"  
 ImageUrl="/\_layouts/images/GORTL.GIF"  
 Location="EditControlBlock"  
 Sequence="401"  
 Title="View Audit History">  
  
<UrlAction Url="~site/\_layouts/ECM401.Auditing/itemAudit.aspx?ItemId={ItemId}&amp;ListId={ListId}"/>  
  
</CustomAction>

Note: The **UrlAction** element includes special tokens that supply the current list and list item identifiers to the page. Also note that the url is specified relative to the current **site** and not the site collection.

1. Now you will create the **ItemAudit.aspx** page. Right-click on the **12\TEMPLATE\LAYOUTS\ECM401.Auditing** folder and select **Add -> New Item...** from the context menu.
2. Create a new text file named **ItemAudit.aspx** and enter the following code.

<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>  
<%@ Page Language="C#" MasterPageFile="~/\_layouts/application.master" Inherits="Microsoft.SharePoint.WebControls.LayoutsPageBase" %>  
<%@ Import Namespace="Microsoft.SharePoint" %>  
<%@ Import Namespace="System.Data" %>  
<%@ Import Namespace="System.IO" %>  
<%@ Import Namespace="System.Security" %>  
<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"  
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>  
<%@ Register TagPrefix="Utilities" Namespace="Microsoft.SharePoint.Utilities" Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>  
  
<script runat="server">  
// \*\*\* CODE GOES HERE \*\*\*  
</script>  
  
<asp:Content ID="Content6" ContentPlaceHolderID="PlaceHolderMain" runat="server">  
<SharePoint:SPGridView ID="grid" runat="server" AutoGenerateColumns="False"  
Width="100%">  
<AlternatingRowStyle CssClass="ms-alternating" />  
</SharePoint:SPGridView>  
</asp:Content>  
<asp:Content ID="Content1" ContentPlaceHolderID="PlaceHolderPageTitle" runat="server">  
<asp:Label ID="ListTitle" runat="server" />  
</asp:Content>  
<asp:Content ID="Content2" ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea"  
runat="server">  
<asp:Label ID="ItemName" runat="server" />  
</asp:Content>

1. This code sets up the page elements, which include an **SPGridView** to display the audit entries and two labels - one to show the list title and another to show the selected item name.
2. Now you will add inline code to handle the page load event. Insert the following code inside the <script> tag at the highlighted location.

Code Snippet: 'Auditing - ItemAudit PageLoad'

protected void Page\_Load(object sender, EventArgs e) {  
  
 SPSite siteColl = SPContext.Current.Site;  
 SPWeb site = SPContext.Current.Web;  
  
 string ListId = Request.QueryString["ListId"];  
 SPList theList = site.Lists[new Guid(ListId)];  
  
 string ItemId = Request.QueryString["ItemId"];  
 SPListItem theItem = theList.Items.GetItemById(Convert.ToInt32(ItemId));  
  
 // Append a custom audit event to record that this page has been viewed.  
 theItem.Audit.WriteAuditEvent(SPAuditEventType.Custom, "CustomViewAuditEvent", "");  
  
 /\*\*\* ADDITIONAL CODE GOES HERE \*\*\*/  
  
}  
  
private SPBoundField AddGridField(string headerText, string fieldName)  
{  
 SPBoundField field = new SPBoundField();  
 field.HeaderText = headerText;  
 field.DataField = fieldName;  
 grid.Columns.Add(field);  
 return field;  
}  
  
string GetUserNameById(int UserId, SPWeb site)   
{  
 try   
 {  
 return site.SiteUsers.GetByID(UserId).Name;  
 }  
 catch   
 {  
 return UserId.ToString();  
 }  
}  
  
// Parse the version string for readability.  
protected string ParseVersionNumber(string versionString) {  
 try {  
 int startMajor = versionString.IndexOf("<Major>") + 7;  
 int endMajor = versionString.IndexOf("</Major>");  
 int lengthMajor = endMajor - startMajor;  
 int startMinor = versionString.IndexOf("<Minor>") + 7;  
 int endMinor = versionString.IndexOf("</Minor>");  
 int lengthMinor = endMinor - startMinor;  
  
 string majorNumber = versionString.Substring(startMajor, lengthMajor);  
 string minorNumber = versionString.Substring(startMinor, lengthMinor);  
  
 if (majorNumber == "0" && minorNumber == "-1")  
 return "N/A";  
  
 return majorNumber + "." + minorNumber;  
 }  
 catch {  
 return "N/A";  
 }  
}

1. To retrieve audit log entries in the context of a non-administrator, elevated security privileges are required. To achieve this use the SPSecurity class by adding the following code to the PageLoad method. This code replaces the comment **"/\*\*\* ADDITIONAL CODE GOES HERE \*\*\*/"**

Code Snippet: 'Auditing - ItemAudit SPSecurity'

SPSecurity.RunWithElevatedPrivileges(delegate() {  
 using (SPSite ElevatedSiteCollection = new SPSite(siteColl.ID)) {  
 using (SPWeb ElevatedSite = ElevatedSiteCollection.OpenWeb(site.ID)) {  
  
 SPList list = ElevatedSite.Lists[new Guid(ListId)];  
 ListTitle.Text = "List: " + list.Title;  
  
 SPListItem item = list.Items.GetItemById(Convert.ToInt32(ItemId));  
 ItemName.Text = "Item: " + item.Name;  
  
 SPAuditQuery query;  
 SPAuditEntryCollection auditEntries;  
  
 query = new SPAuditQuery(ElevatedSiteCollection);  
 query.RestrictToListItem(item);  
 auditEntries = ElevatedSite.Audit.GetEntries(query);  
  
 DataTable table = new DataTable();  
 table.Columns.Add("User", typeof(string));  
 table.Columns.Add("Event", typeof(string));  
 table.Columns.Add("Date", typeof(DateTime));  
 table.Columns.Add("Version", typeof(string));  
  
 DataRow newRow;  
  
 foreach (SPAuditEntry entry in auditEntries) {  
 newRow = table.Rows.Add();  
 newRow["User"] = GetUserNameById(entry.UserId, site);  
  
 // check for our custom event  
 if (entry.SourceName == "CustomViewAuditEvent") {  
 newRow["Event"] = "View Audit Log";  
 }  
 else {  
 newRow["Event"] = entry.Event;  
 }  
 newRow["Date"] = entry.Occurred.ToLocalTime();  
 newRow["Version"] = ParseVersionNumber(entry.EventData);  
 }  
  
 AddGridField("User","User");  
 AddGridField("Event","Event");  
 AddGridField("Date","Date").ControlStyle.Width = new Unit(140);  
 AddGridField("Version","Version");  
  
 grid.AutoGenerateColumns = false;  
 grid.DataSource = table.DefaultView;  
 grid.DataBind();  
  
 grid.AllowSorting = true;  
 grid.HeaderStyle.Font.Bold = true;  
 }  
 }  
 });

1. Rebuild the project, re-activate the feature and test your work. Now, from the **Edit Control Block** of any document library, you should see an extra command to display the audit log entries for the selected item. Select the command to open a page showing the audit log entries for the item.

**This concludes the lab exercises.**