Lab 4: Information Management Policy

**Lab Time: 60 Minutes**

**Lab Directory: ECM401.InformationPolicy**

**Back Story:**

Litware management has decided to use Windows SharePoint Services 3.0 to manage sensitive company information. Because there are many different kinds of documents, the Litware IT department has decided to use information policy features to enable content administrators to apply different kinds of controls to various documents without having to create a separate feature for each document type.

**Lab Overview:**

In this lab, you will create a custom information policy feature that controls the distribution of printed material by limiting the printing of certain documents to a specific set of secure printers. Your policy feature will contain a list of trusted printers, and will work in conjunction with a **VSTO** add-in that traps the Print event to determine whether a trusted printer is being used. If not, then the print job is cancelled.

# Exercise 1: Create a SharePoint Feature Project

1. When working with policy features, it is useful to have a library of components that can be reused in several projects. In this exercise, you will employ a set of utility classes that greatly simplify the creation of custom policies, policy features and policy resources. Start by creating a new **SharePoint Feature** project named **ECM401.PrinterControlPolicyFeature** as shown below.
2. Because Information Policy Features are installed globally, your feature could be scoped to any level. For convenience and in anticipation of adding additional components, such as content types and fields, to your solution, set the feature scope to **Site** as shown in the following illustration.
3. In order to use the records management features, you will need to add a reference to the **Microsoft.Office.Policy** assembly. Right-click the References node and select Add Reference…. Click on the **Browse** tab and navigate to the **12\ISAPI** folder. Select the **Microsoft.Office.Policy.dll** file and click ok.
4. Next, add a reference to the **ECM401** component library, which is located in the **student\solution\ecm401** folder. You can either add a reference to the assembly, or you can load the project and add a dependency reference to it.

Note: Browse through the source code to familiarize yourself with how the base classes are written. This project contains a number of classes that are designed to simplify the development of ECM solutions.

1. The **ECM401** project includes some additional references you need to add in order for your project to build. First, add a reference to the **System.Configuration.Install** assembly from the **.NET** tab. You will also need the **System.Web** assembly.

# Exercise 2: Create the Printer Control Policy Feature class

1. Policy features allow administrators to select which parts of a policy shoule be applied to different items. In this exercise, you will create a custom policy feature that maintains a list of "trusted" printers that users are allowed to send output to. In fhte final exercise, you will create a **VSTO** add-in that uses this information to prevent the user from printing to an untrusted printer.
2. Add a class to the project called **PrinterPolicyFeature** and open the source file for editing. Add the following **using** statements at the top of the file.

using System;  
using System.ComponentModel;  
using System.Diagnostics;  
using System.Xml;  
using Microsoft.Office.RecordsManagement.InformationPolicy;  
using Microsoft.SharePoint;  
  
using ECM401.ContentTypes;  
using ECM401.InformationPolicy;

1. Delete the generated class declaration and replace it with the following code snippet.

Code Snippet: 'PrinterPolicyFeature - Class Declaration'

[Name("Document Print Controller")]  
[Description("Maintains a list of 'trusted' printers so that administrators "  
 + "can control where documents are printed.")]  
[Publisher("John F. Holliday")]  
public class PrinterPolicyFeature : SharePointPolicyFeature  
{  
 public const string TrustedPrintersFieldName = "TrustedPrinters";  
 public const string PrintControlPolicyNamespace = "urn:ecm401:policy.printcontrol";  
  
 public PrinterPolicyFeature()  
 {  
 }  
}

1. Your policy feature class inherits from the **ECM401.InformationPolicy.SharePointPolicyFeature** abstract base class. This class provides a default implementation of the **IPolicyFeature** interface, which must be implemented to enable SharePoint to communicate with the policy feature. Using an abstract base allows you to implement only the methods you require.
2. The base class also uses **attributes** to generate the CAML code needed to register your component in the SharePoint environment. The *TrustedPrintersFieldName* and *PrintControlPolicyNamespace* constants will be used to setup the content types and list items affected by the policy.
3. First, create a static method for writing trace messages during development. Add the following code to the class definition.

/// <summary>  
/// Helper method for logging trace messages.  
/// </summary>  
public static void Log(string message)  
{  
 Trace.WriteLine(message,"ECM401.PrinterPolicyFeature");  
}

1. Policy features work by hooking into the event receiver mechanism for the lists and content types to which they are attached. When the printer control policy feature is registered for a content type, it will setup event receivers for the **ItemAdded** and **ItemUpdated** events so that it can copy the list of trusted printers into a special column on the target list item. To make it easier to register and unregister event receivers, the **ECM401** utility library includes a class called **ItemEventReceiver** . In the next part of the exercise, you will inherit from this class to declare event receiver methods for these two event types.
2. Add a new *nested class* declaration inside the **PrinterPolicyFeature** class named **PrinterPolicyEventReceiver** that inherits from **ItemEventReceiver** . Your code should match the following code snippet.

Code Snippet: 'PrinterControlFeature - ItemEventReceiver'

/// <summary>  
/// This class implements the event receivers for the policy feature.  
/// </summary>  
public class PrinterPolicyEventReceiver : ItemEventReceiver  
{  
 /// <summary>  
 /// Handles the item added event to set the list of trusted printers.  
 /// </summary>  
 /// <param name="properties"></param>  
 public override void ItemAdded(SPItemEventProperties properties)  
 {  
 base.ItemAdded(properties);  
 Log("PrinterPolicyFeature.ItemAdded - " + properties.ListTitle);  
 AddPrintersToListItem(properties.ListItem);  
 }  
  
 /// <summary>  
 /// Handles the item updated event to set the list of trusted printers.  
 /// </summary>  
 /// <param name="properties"></param>  
 public override void ItemUpdated(SPItemEventProperties properties)  
 {  
 base.ItemUpdated(properties);  
 Log("PrinterPolicyFeature.ItemUpdated - " + properties.ListTitle);  
 AddPrintersToListItem(properties.ListItem);  
 }  
}

1. Both of these routines delegate the job of adding the list of printers to the list item whenever a new item is added or an existing item is updated. The list of printers is extracted from the content type *payload* (a custom XMLDocument added by the policy feature) and placed into the *TrustedPrinters* field of the list item. Add the following code snippet to the *PrinterPolicyEventReceiver* class definition.

Code Snippet: 'PrinterPolicyEventReceiver - AddPrintersToListItem'

/// <summary>  
/// Gets the list of trusted printers from the content type associated  
/// with the item and places it into the appropriate field.  
/// </summary>  
/// <param name="item"></param>  
private void AddPrintersToListItem(SPListItem item)  
{  
 if (item == null || item.ContentType == null)  
 return;  
 Log("AddPrintersToListItem '" + item.Title + "'");  
 string xml = item.ContentType.XmlDocuments[PrinterPolicyFeature.PrintControlPolicyNamespace];  
 if (string.IsNullOrEmpty(xml))  
 {  
 Log("-- content type payload is empty");  
 }  
 else  
 {  
 Log("-- loading content type payload");  
 XmlDocument xmlDoc = new XmlDocument();  
 xmlDoc.LoadXml(xml);  
 XmlNamespaceManager nsmgr = new XmlNamespaceManager(xmlDoc.NameTable);  
 nsmgr.AddNamespace("p", PrinterPolicyFeature.PrintControlPolicyNamespace);  
 XmlNode node = xmlDoc.SelectSingleNode("p:data/p:printers", nsmgr);  
 Log("-- '" + node.InnerText + "'");  
 SPField field = item.Fields[PrinterPolicyFeature.TrustedPrintersFieldName];  
  
 try  
 {  
 item[field.Id] = node.InnerText;  
 item.SystemUpdate();  
 Log("-- UPDATED!");  
 }  
 catch (SPException x)  
 {  
 Log(string.Format("-- FAILED to update item - {0}", x.Message));  
 }  
 }  
}

1. Now you are ready to override selected methods of the **SharePointPolicyFeature** base class. Add a region to the **PrinterPolicyFeature** class named *SharePointPolicyFeature Overrides* . Insert the following code snippet inside the region.

Code Snippet: 'SharePointPolicyFeature - Registration Routines'

/// <summary>  
/// This method is called when the feature is registered for a content type.  
/// Adds a "TrustedPrinters" field to the content type.  
/// </summary>  
/// <param name="ct"></param>  
public override void Register(SPContentType ct)  
{  
 base.Register(ct);  
 Log("Registering print control for content type: " + ct.Name);  
  
 // Setup the item event receiver for the content type.  
 ItemEventReceiver.Create(ct, typeof(PrinterPolicyEventReceiver));  
  
 // Add the "TrustedPrinters" field to the content type.  
 SPFieldLink fieldRef = SharePointContentType.AddOrCreateFieldReference(ct,   
 PrinterPolicyFeature.TrustedPrintersFieldName, SPFieldType.Text, false,   
 true, true,true,true);  
}  
  
/// <summary>  
/// This method is called when the policy feature is removed.  
/// Uninstalls the event receiver from the content type.  
/// </summary>  
/// <param name="ct"></param>  
public override void UnRegister(SPContentType ct)  
{  
 base.UnRegister(ct);  
 Log("Unregistering PrintControl for content type: " + ct.Name);  
 ItemEventReceiver.Remove(ct, typeof(PrinterPolicyEventReceiver));  
}

1. These routines handle the registration and un-registration of the feature for a given content type. The code first creates an ItemEventReceiver for the content type using the nested class you just created, and then it adds a "TrustedPrinters" field to the content type.
2. Next, you will override the **OnCustomDataChange** method, which is called whenever the custom data associated with the policy item changes. This happens when the policy administrator enters a new value into the custom UI you will provide for specifying the list of trusted printer names. Add the following code snippet to the class definition.

Code Snippet: 'SharePointPolicyFeature - OnCustomDataChange'

/// <summary>  
/// This method is called when the custom data for a policy item changes.  
/// </summary>  
/// <param name="policyItem"></param>  
/// <param name="ct"></param>  
public override void OnCustomDataChange(PolicyItem policyItem, SPContentType ct)  
{  
 base.OnCustomDataChange(policyItem, ct);  
 Log("OnCustomDataChange for policy item: " + policyItem.Name  
 + " and content type " + ct.Name + " where policy item custom data = "  
 + policyItem.CustomData);  
  
 try  
 {  
 // Get the custom data from the policy item.  
 XmlDocument xmlDoc = new XmlDocument();  
 xmlDoc.LoadXml(policyItem.CustomData);  
 XmlNamespaceManager nsmgr = new XmlNamespaceManager(xmlDoc.NameTable);  
 nsmgr.AddNamespace("p", PrinterPolicyFeature.PrintControlPolicyNamespace);  
  
 // Add the custom data to the content type payload.  
 ct.XmlDocuments.Delete(PrinterPolicyFeature.PrintControlPolicyNamespace);  
 ct.XmlDocuments.Add(xmlDoc);  
 ct.Update();  
 }  
 catch (Exception x)  
 {  
 Log("OnCustomDataChange failed for item: " + policyItem.Name + " - " + x.ToString());  
 }  
}

1. SharePoint monitors the lifecycle of items affected by a policy and calls the **ProcessListItems** method when updates are needed for a given item. Next, you will override this method to update the **TrustedPrinters** field with the updated list from the policy item. Add the following code snippet to the class definition.

Code Snippet: 'SharePointPolicyFeature - ProcessListItem'

/// <summary>  
/// This method is called by the policy framework for list items that were  
/// created before the policy was applied to the list.  
/// </summary>  
/// <param name="site"></param>  
/// <param name="policyItem"></param>  
/// <param name="listItem"></param>  
/// <returns></returns>  
public override bool ProcessListItem(SPSite site, Microsoft.Office.RecordsManagement.InformationPolicy.PolicyItem policyItem, SPListItem listItem)  
{  
 Log("Processing list item: " + listItem.Title);  
 bool result = base.ProcessListItem(site, policyItem, listItem);  
  
 // Add the policy item custom data to the TrustedPrinters field.  
 try  
 {  
 // Get the policy data from the item payload.  
 XmlDocument xmlDoc = new XmlDocument();  
 xmlDoc.LoadXml(policyItem.CustomData);  
  
 XmlNamespaceManager nsmgr = new XmlNamespaceManager(xmlDoc.NameTable);  
 nsmgr.AddNamespace("p", PrinterPolicyFeature.PrintControlPolicyNamespace);  
 XmlNode node = xmlDoc.SelectSingleNode("p:data/p:printers", nsmgr);  
  
 // Store the list of trusted printers into the list item.  
 listItem[PrinterPolicyFeature.TrustedPrintersFieldName] = node.InnerText;  
 listItem.Update();  
 }  
 catch (Exception x)  
 {  
 Log("ProcessListItem failed for item: " + listItem.Title + " - " + x.ToString());  
 }  
 return result;  
}

1. In order to test the policy feature, it must be registered in the site collection when the feature is activated. Open the **FeatureReceiver.cs** file for editing. Replace the *FeatureActivated* method with the following code snippet.

Code Snippet: 'Printer Control Policy Feature Activated'

/// <summary>  
/// Override to install the printer control policy feature.  
/// </summary>  
/// <param name="properties"></param>  
public override void FeatureActivated(SPFeatureReceiverProperties properties)  
{  
 using (SPSite site = properties.Feature.Parent as SPSite)  
 if (!SharePointPolicyFeature.Install(typeof(PrinterPolicyFeature)))  
 Trace.WriteLine("Printer Policy Feature installation failed.");  
}

1. This method gets the *SPSite* object from the feature receiver properties and uses the static *Install* method of the *SharePointPolicyFeature* base class to register the policy feature within the SharePoint environment.
2. Similarly, replace the *FeatureDeactivating* method with the code snippet shown below.

Code Snippet: 'Printer Control Policy Feature Deactivating'

/// <summary>  
/// Override to remove the printer control policy feature.  
/// </summary>  
/// <param name="properties"></param>  
public override void FeatureDeactivating(SPFeatureReceiverProperties properties)  
{  
 using (SPSite site = properties.Feature.Parent as SPSite)  
 if (!SharePointPolicyFeature.Uninstall(typeof(PrinterPolicyFeature)))  
 Trace.WriteLine("Printer Policy Feature removal failed.");  
}

1. Now you are ready to test your work so far. Build the project and navigate to any SharePoint site. From the *Site Settings* page, select *Site Collection Features* . You should see the your feature listed.
2. Navigate to any document library and then from the *List Settings* page, select the *Information management policy settings* link.
3. If the document library has content types enabled, then you will see a list of content types to choose from. Click one of the links to go to the policy settings page. If content types are not enabled for the list, you will go directly to that page.
4. From the policy settings page, select the *Define a policy...* link and click OK.
5. Now, you should see your policy feature displayed at the bottom of the list of available policy features to be enabled.
6. If you enable the feature, you will get a post-back, but nothing is displayed beneath the checkbox. In the next exercise, you will add a custom user interface to enable administrators to specify the list of trusted printers.

# Exercise 3: Add Custom Policy Feature Settings

1. In this exercise, you will create a custom control for entering a list of trusted printers. This will be a simple textbox with instructions to enter a semicolon-delimited list of printer names.

Note: In actual practice, you would more likely retrieve the list of trusted printers from an external database or from a SharePoint list and present them as a dropdown list or combo-box.

1. The first step is to create a control template that SharePoint can load when the administrator selects the checkbox next to the name of your policy feature.
2. Right-click the *TEMPLATE* folder in the *Solution Explorer* window and select **Add -gt; New Folder** from the context menu. Name the new folder **LAYOUTS** . Beneath the LAYOUTS folder, add a sub-folder named **ECM401.PrinterControlPolicyFeature** . Within that folder, add a new **Text File** item and give it the name **PrinterPolicySettings.ascx** .

Note: Although you selected *Text File* as the item type, Visual Studio uses the file extension to determine which editor to open. Here, you are creating an **ASCX** file to use as a template for your custom user interface.

1. Add the following code to the ASCX file.

XML Snippet: 'Printer Policy Settings Control Template'

<%@ Assembly Name="ECM401.PrinterControlPolicy, Version=1.0.0.0, Culture=neutral, PublicKeyToken=d3d13d75b9fd2b46" %>  
<%@ 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" %>  
<%@ Import Namespace="Microsoft.SharePoint" %>  
<%@ Control Language="C#" Inherits="ECM401.PrinterControlPolicy.PrinterPolicySettings" %>  
<p>  
 <table cellpadding="0" class="ms-authoringcontrols">  
 <tr>  
 <td>  
 &nbsp;</td>  
 <td>  
 <asp:Label ID="Label1" runat="server"   
 Text="Enter trusted printer names, separated by ';'"></asp:Label></td>  
 </tr>  
 <tr>  
 <td>  
 &nbsp;</td>  
 <td>  
 <asp:TextBox ID="TextBoxPrinters" runat="server" TextMode="MultiLine"   
 Rows="5" MaxLength="1024" Columns="40" class="ms-input"  
 ToolTip="Enter the list of trusted printers here." />  
 <asp:RequiredFieldValidator ID="RequiredValidatorPrinters"   
 ControlToValidate="TextBoxPrinters"  
 ErrorMessage="At least one printer name is required."   
 Text="Please enter a semicolon-delimited list of printer names."  
 EnableClientScript="false" runat="server" />  
 </td>  
 </tr>  
 </table>  
</p>

1. This template declares a Label, a TextBox and an associated FieldValidator control. It inherits from a code-behind class you will now implement in a separate file.
2. Add a new class to the project named **PrinterPolicySettings** and open the file for editing. Delete the generated class declaration and enter the following code snippet.

Code Snippet: 'PrinterPolicySettings - Class Declaration'

/// <summary>  
/// This class implements a custom Information Policy settings control  
/// that enables an administrator to enter the list of trusted printers.  
/// </summary>  
public class PrinterPolicySettings : CustomSettingsControl  
{  
 // holds the custom data associated with the control  
 string m\_customData = string.Empty;  
  
 // automatically bound to the TextBox control in the template  
 protected TextBox TextBoxPrinters;  
}

1. Add the following using statements at the top of the file.

using System.Web.UI.WebControls;  
using System.Xml;  
using Microsoft.Office.RecordsManagement.InformationPolicy;  
using Microsoft.SharePoint;

1. The *m\_customData* string will hold the custom data being transferred to and from the TextBox control. The protected *TextBoxPrinters* control is automatically bound to the template by ASP.NET.
2. Your *PrinterPolicySettings* class inherits from a base class that is provided by the SharePoint Information Policy framework for implementing policy feature user interfaces. The *CustomSettingsControl* class is an abstract class that declares several abstract methods. Because they are abstract methods, you must provide an implementation for all of them in your derived class.
3. Several methods will not be used in this solution, so you can enter stubs for the overrides as follows.

Code Snippet: 'PrinterPolicySettings - Unused Methods'

/// <summary>  
/// Not used - must be implemented because base class is abstract.  
/// </summary>  
public override void RaisePostDataChangedEvent(){}  
  
/// <summary>  
/// Not used - must be implemented because base class is abstract.  
/// </summary>  
public override SPList List { get; set; }  
  
/// <summary>  
/// Not used - must be implemented because base class is abstract.  
/// </summary>  
public override SPContentType ContentType { get; set; }

1. The custom data consists of a list of printer names that you will embed into an **XML** fragment so it can be added to the payload of a content type. Add the following method to the class definition.

Code Snippet: 'PrinterPOlicySettings - CustomData'

/// <summary>  
/// This accessor is called to get or set the custom data  
/// associated with the control.  
/// </summary>  
public override string CustomData  
{  
 get  
 {  
 XmlDocument xmlDoc = new XmlDocument();  
 string ns = PrinterPolicyFeature.PrintControlPolicyNamespace;  
 new XmlNamespaceManager(xmlDoc.NameTable).AddNamespace("p", ns);  
 XmlElement rootNode = xmlDoc.CreateElement("p", "data", ns);  
 xmlDoc.AppendChild(rootNode);  
 XmlElement printersNode = xmlDoc.CreateElement("p", "printers", ns);  
 rootNode.AppendChild(printersNode);  
 printersNode.InnerText = TextBoxPrinters.Text;  
 m\_customData = xmlDoc.InnerXml;  
 return m\_customData;  
 }  
 set  
 {  
 m\_customData = value;  
 }  
}

1. The *LoadPostData* method is called to set the custom data string after the text in the control is modified.

Code Snippet: 'PrinterPolicySettings - LoadPostData'

/// <summary>  
 /// This method updates the custom data associated with the control.  
 /// </summary>  
 /// <param name="postDataKey"></param>  
 /// <param name="values"></param>  
 /// <returns></returns>  
 public override bool LoadPostData(string postDataKey, System.Collections.Specialized.NameValueCollection values)  
 {  
 string oldData = this.CustomData;  
 string newData = values[postDataKey];  
 if (oldData != newData)  
 {  
 this.CustomData = newData;  
 return true;  
 }  
 return false;  
 }

1. Finally, you need to initialize the TextBox with the initial data when the page is loaded.

Code Snippet: 'PrinterPolicySettings - OnLoad'

/// <summary>  
/// Loads the custom data string into the textbox whenever the control is loaded.  
/// </summary>  
/// <param name="e"></param>  
protected override void OnLoad(EventArgs e)  
{  
 base.OnLoad(e);  
 if (!(base.IsPostBack || string.IsNullOrEmpty(m\_customData)))  
 {  
 try  
 {  
 XmlDocument xmlDoc = new XmlDocument();  
 xmlDoc.LoadXml(m\_customData);  
 XmlNamespaceManager nsmgr = new XmlNamespaceManager(xmlDoc.NameTable);  
 nsmgr.AddNamespace("p", PrinterPolicyFeature.PrintControlPolicyNamespace);  
 XmlNode node = xmlDoc.SelectSingleNode("p:data/p:printers", nsmgr);  
 TextBoxPrinters.Text = node.InnerText;  
 }  
 catch (Exception x)  
 {  
 System.Diagnostics.Trace.WriteLine("Failed to load printer settings - " + x.Message, GetType().Name);  
 }  
 }  
}

1. Save and close the file.
2. Now that you have created the control template and implemented its code-behind class, you need to tell SharePoint where it is located on the server. You also need to provide some instructions for the administrator describing the purpose of the feature and how it should be used.
3. Reopen the *PrinterPolicyFeature.cs* file for editing.

Note: Because the base *SharePointPolicyFeature* class inherits from the *Installer* class, Visual Studio treats the class file as a *Component* , causing the IDE to attempt to load a designer for it. To go directly to the code editor, you must right-click the file and choose *View Code* .

1. Add the following lines to the *PrinterPolicyFeature* constructor.

Code Snippet: 'PrinterPolicyFeature - Constructor'

this.ConfigPage = "ECM401.PrinterControlPolicyFeature/PrinterPolicySettings.ascx";  
 this.ConfigPageInstructions = @"Add the names of trusted printers "  
 + "here, separated by semicolons. This will prevent users from printing "  
 + "documents to which this policy is applied on unsecure printers. "  
 + "NOTE: In order for this to work, you must also deploy the "  
 + "PrintController add-on for Microsoft Office to all client machines.";

1. Save your work and rebuild the project.
2. Navigate back to the *Policy Settings* page for the policy you created earlier. Now, your custom instructions appear beneath the *Document Print Controller* caption. When you click the checkbox next to the *Enable Document Print Controller* policy feature, you should see a labeled TextBox control where you can enter the list of trusted printers.
3. To test your work, enter some names into the TextBox control and save the policy. Whenever you add or update an item to which the policy is attached, you should see the list of printers in the *Trusted Printers* column of the item.

Note: It is not necessary to copy the list of trusted printers into a column of each list item. This was done in the lab only to illustrate the steps that are typically required when writing policy features. For the purposes of this scenario - managing trusted printers - you can access the policy definition along with its custom data directly from the content type payload. In the next exercise, you will use this approach to prevent users from printing to untrusted printers.

# Exercise 4: Create a Print Monitor Add-in for Word 2007

1. To prevent users from violating the printer control policy, you need some additional code that will run on client machines. This code must be installed on each client and configured as an application-level add-in for the office client applications. In this exercise, you will use **Visual Studio Tools for Office** to create a print monitor add-in for Word 2007 that will read the trusted printer list from the document and compare them to the currently active printer. If the currently active printer name is not in the list, then any attempt to print the document is aborted with a message.
2. When an information policy is attached to a content type, a copy of the policy is embedded within any document that is associated with the type. This means that the entire policy definition travels with the document and is accessible from your add-in code.
3. Start by creating a new **VSTO** project called **ECM401.PrintMonitorAddin** . From the **New Project** dialog, select **Word 2007 Add-in** from the **Office/2007** section.
4. Open the **ThisAddIn.cs** file for editing. Add the folllwing namespaces to the *using* statements at the top of the file.

using System.Xml;  
using System.Windows.Forms;

1. The first bit of code to add will handle the *Startup* event to install a handler for the print event. This handler will be called whenever the user attempts to print the active document. Add the following code inside the **ThisAddIn\_Startup** event handler that was auto-generated by the *New Project* wizard.

// install a handler for the print event  
this.Application.DocumentBeforePrint  
 += new Word.ApplicationEvent4\_DocumentBeforePrintEventHandler(  
 Application\_DocumentBeforePrint);

1. Replace the generated event handler stub with the following code snippet.

Code Snippet: 'PrintMonitorAddin - DocumentBeforePrint'

/// <summary>  
/// This event handler gets the name of the printer the user is  
/// attempting to use. It then obtains the list of "trusted" printers  
/// as defined by the PrintControl policy associated with the document.  
/// If the current printer is not trusted, then a message is displayed  
/// and the operation is cancelled.  
/// </summary>  
/// <param name="Doc"></param>  
/// <param name="Cancel"></param>  
void Application\_DocumentBeforePrint(Word.Document Doc, ref bool Cancel)  
{  
 if (!PrinterIsTrusted(Application.ActivePrinter))  
 {  
 MessageBox.Show(String.Format(  
 "Sorry. Big Brother does not want you to print this document " +   
 "on printer '{0}'! Please select a different printer.",   
 Application.ActivePrinter),  
 "Printer Is Not Trusted");  
 Cancel = true;  
 }  
}

1. The next routine will return true if the specified printer is a trusted printer as defined by the policy.

Code Snippet: 'PrintMonitorAddin - PrinterIsTrusted'

/// <summary>  
/// Determines whether a specified printer is trusted.  
/// </summary>  
/// <param name="printerName"></param>  
/// <returns>true if the specified printer is in the list of trusted printers</returns>  
bool PrinterIsTrusted(string printerName)  
{  
 // Retrieve the list of trusted printers from the attached policy.  
 List<string> trustedPrinters = GetTrustedPrintersList();  
 // Check whether the specified printer is in the list.  
 if (trustedPrinters.Count == 0)  
 return true;  
 return trustedPrinters.Contains(printerName);  
}

1. Finally, you need a routine to extract the list of trusted printers from the information policy attached to the document. The information policy is embedded as a *CustomXMLPart* object in the active document. The collection of custom xml parts is exposed as a property of the *Microsoft.Office.Interop.Word.Document* object. Add the following code snippet to the class definition.

Code Snippet: 'PrintMonitorAddin - GetTrustedPrintersList'

/// <summary>  
/// Retrieves the list of trusted printers from the PrintControl policy  
/// associated with the document.  
/// </summary>  
/// <remarks>  
/// This method searches through the list of custom XML parts in  
/// the document until it finds one matching the PrintControl policy URN.  
/// </remarks>  
/// <returns>the list of trusted printers from the attached policy</returns>  
List<string> GetTrustedPrintersList()  
{  
 List<string> result = new List<string>();  
 const string policyNamespace = "urn:ecm401:policy.printcontrol";  
 Office.CustomXMLParts parts = Application.ActiveDocument.CustomXMLParts;  
 foreach (Office.CustomXMLPart part in parts)  
 {  
 if (part.XML.Contains(policyNamespace))  
 {  
 // extract the list of trusted printers.  
 XmlDocument xmlDoc = new XmlDocument();  
 xmlDoc.LoadXml(part.XML);  
 XmlNamespaceManager nsmgr = new XmlNamespaceManager(xmlDoc.NameTable);  
 nsmgr.AddNamespace("p", policyNamespace);  
 XmlNode node = xmlDoc.SelectSingleNode("p:data/p:printers", nsmgr);  
 if (node != null)  
 {  
 string[] printers = node.InnerText.Split(";".ToCharArray());  
 foreach (string printer in printers)  
 result.Add(printer);  
 break;  
 }  
 }  
 }  
 return result;  
}

1. To test your work, simply run the project in the debugger and open a document from the document library you used to test the information policy. Try to print the document. You should see a dialog like the one shown below.

**This concludes the lab exercises.**