INTEGRATE

Microsoft Dynamics® Security Synchronization Utility

Integrating with the ApplicationRoleProvider Framework

White Paper

Utilizing the ApplicationRoleProvider class to develop and configure standalone applications and create application providers compatible with the Microsoft Dynamics Security Synchronization Utility

Date: July 11, 2008

http://www.Microsoft.com/Dynamics

Table of Contents

Overview of the Microsoft Dynamics® Security Synchronization Utility

The Microsoft Dynamics Security Synchronization Utility is a standalone executable application that enables the synchronization of users across applications. The application supports various Microsoft® applications, including Windows® SharePoint® Services, Microsoft Dynamics GP Web Services, Microsoft Dynamics GP, and Business Portal for Microsoft Dynamics GP.

In addition, it is also a framework made up of a suite of APIs that can be used to generically and simple interact with the security subsystems of applications.

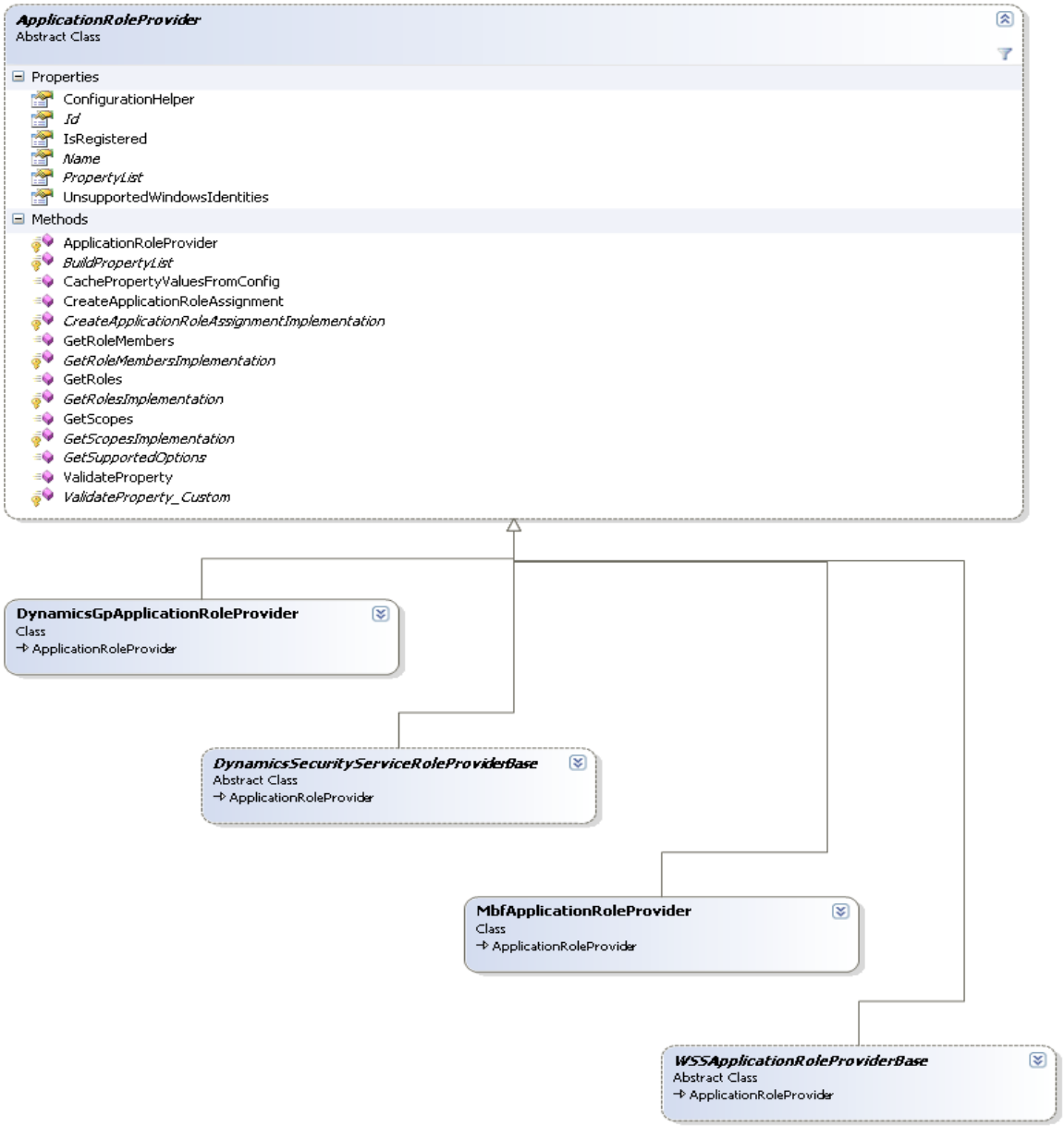
The utility reads from and writes to a configuration file that stores the property values specific to each application provider. These properties vary by application. The utility's user interface allows the user to specify values for these properties and writes them to the configuration file.

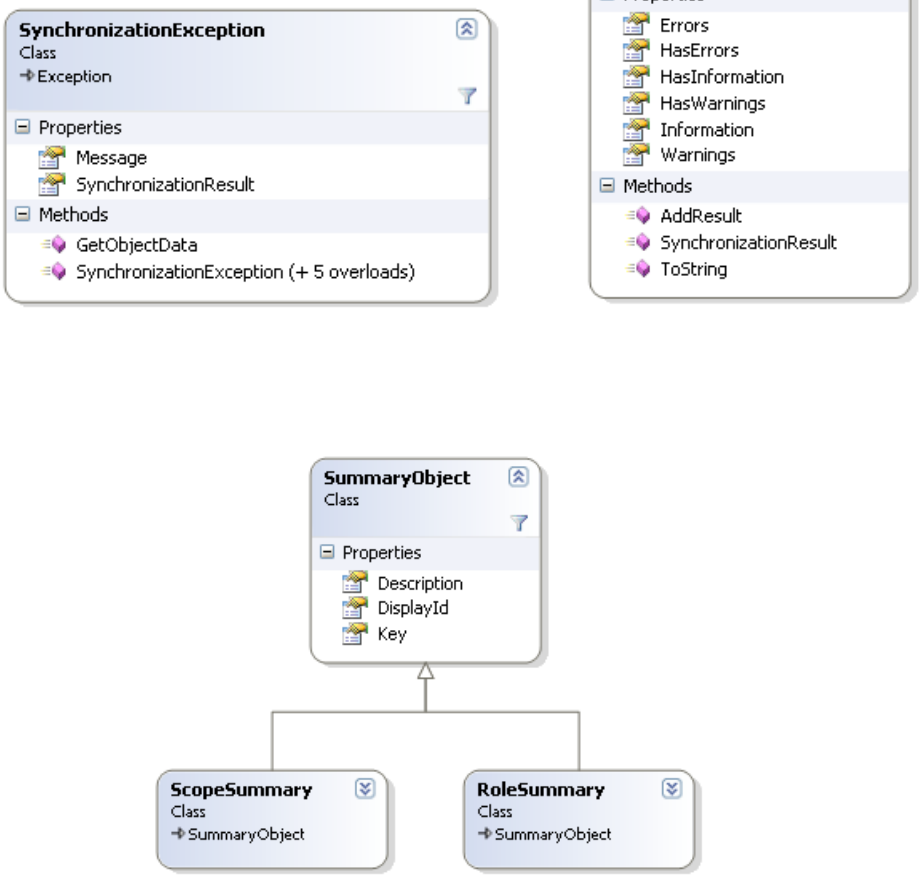
This document describes the ApplicationRoleProvider class and explains how to develop assemblies that will allow third-party applications to work with the synchronization utility. It also addresses customizing property validation and deployment of the assembly.

Code samples are included for all of the classes mentioned and a comprehensive example of creating a provider is included at the end.

Architecture

The overall architecture of the ApplicationRoleProvider class looks like this:




SynchronizationException
Class
Exception
Properties
Message
SynchronizationResult
Methods
GetObjectData
SynchronizationException (+ 5 overloads)
SynchronizationResult
Class
Properties
Errors
HasErrors
HasInformation
HasWarnings
Information
Warnings
Methods
AddResult
SynchronizationResult
ToString
SummaryObject
Class
Properties
Description
DisplayId
Key
ScopeSummary
Class
SummaryObject
RoleSummary
Class
SummaryObject
ApplicationRoleAssignment
Class
Properties
Application
Members
Role
Scope
Methods
ApplicationRoleAssignment
ApplicationRoleProviderConfigurationValues
Class
Properties
PropertyDisplayName
PropertyExample
PropertyIsRequired
PropertyName
PropertyType
PropertyValue
Methods
ApplicationRoleProviderConfigurationValues

Abstract Properties

PropertyList

Implementers must return a list of properties that can be configured for the provider. Properties that should be defined here include those for which a consumer of your code can define values. For example, if your provider uses a web service the Url for the web service should be included in PropertyList, as this value must be set for your provider to work. ApplicationRoleProvider uses the BuildPropertyList() method (see below) to populate this list.

Id

Implementers must return the Guid value defined in a *GuidAttribute*. This represents a unique identifier for the provider.

Name

The name of the provider used for display in the user experience.

Abstract Methods

GetSupportedOptions()

```
public override ApplicationRoleProviderOptions GetSupportedOptions()
    {
        if (supportedOptions == null)
        {
            supportedOptions = new ApplicationRoleProviderOptions();
            supportedOptions.DoesSupportDomainGroups = true;
            supportedOptions.DoesSupportEnterpriseWideScope = false;
            supportedOptions.DoesSupportScopes = false;
            supportedOptions.DoesRequireExistingApplicationUser = false;
            supportedOptions.DoesSupportScopeSpecificRoles = false;
        }
        return supportedOptions;
    }
```

Implementers must return a list of supported options for use by other providers. This method returns an ApplicationRoleProviderOptions object. This class contains the following properties:

DoesSupportDomainGroups

Defines whether the application allows Active Directory Domain groups to be assigned to application roles.

DoesSupportEnterpriseWideScope

Defines whether the application allows a role to be assigned to all available scopes simultaneously. In Microsoft Dynamics GP and Business Portal, this is referred to as the “All” companies scope. A scope is an area in which a role is contained, such as a company or other business unit.

DoesSupportScopes

Defines whether the application supports scopes. In Microsoft Dynamics GP and Business Portal scope refers to the specific company to which a role has access.

DoesRequireExistingApplicationUser

Defines whether the application requires a user to already exist in the application before synchronizing users from another application. If this is set to “false”, the application should be able to create a new user at the time of synchronization.

DoesSupportScopeSpecificRoles

Defines whether the application allows roles to be assigned to a specific scope.

GetScopesImplementation()

This method returns a collection of type `ScopeSummary` that represents the scopes that are supported by your application.

```
class MyClass
{
    private Guid id;
    private Scope myScope;

    public MyClass()
    {
        id = new Guid();

        id = 997EDB99-E1E0-4072-B844-18D7C57EB088;
        myScope = new Scope();

        myScope.Key = id;
    }
    protected override Collection<ScopeSummary> GetScopesImplementation()
    {
        Collection<ScopeSummary> scopes = new Collection<ScopeSummary>();

        //Create the scope info objects and add them to the collection
        ScopeSummary scopeSummary = new ScopeSummary();
        scopeSummary.Key = id.ToString();
        scopeSummary.DisplayId = "My scope’s display name";
        scopeSummary.Description = "This is the description for my scope";
        scopes.Add(scopeSummary);

        return scopes;
    }
}
```

The `ScopeSummary` type is contained in the Microsoft.Dynamics.Security.Synchronization assembly and has three main properties:

Key

The Key is the identifier by which your application knows a scope. Implementers must provide a mechanism by which the unique identifier for a scope within the application can be serialized as a string.

DisplayId

The DisplayId is the name by which the scope is presented in the user experience.

Description

The Description contains more information about the scope.

GetRolesImplementation()

This method returns a collection of type `RoleSummary` that represents the roles that are supported by your application.

```
class MyClass
{
    private Guid id;
    private Role myRole;

    public MyClass()
    {
        id = new Guid();

        id = 997EDB99-E1E0-4072-B844-18D7C57EB088;
        myRole = new Role();

        myRole.Key = id;
    }
}
```

The `RoleSummary` type is contained in the Microsoft.Dynamics.Security.Synchronization assembly:

```
protected override Collection<RoleSummary> GetRolesImplementation(string scopeKey)
{
    Collection<RoleSummary> roles = new Collection<RoleSummary>();

    //Create the role info objects and add them to the collection
    RoleSummary roleSummary = new RoleSummary();
    roleSummary.Key = id.ToString();
    roleSummary.DisplayId = "My role's display name";
    roleSummary.Description = "This is the description for my role";
    roles.Add(roleSummary);

    return roles;
}
```

The collection contains these properties:

Key

The Key is the identifier by which your application knows a role. Implementers must provide a mechanism by which the unique identifier for a scope within the application can be serialized as a string.

DisplayId

The DisplayId is the name by which your scope is presented in the user experience.

Description

The Description contains more information about the role.

GetRoleMembersImplementation()

This method returns a collection that represents the members that are assigned to a specified role within a specified scope. This method should utilize some exception handling for Soap or other errors that may occur. Providers shipped with the synchronization utility that support scopes and roles utilize the SynchronizationException and SynchronizationResult objects for this purpose.

```
protected override Collection<string> GetRoleMembersImplementation(string roleKey, string scopeKey)
{
    Collection<string> members = new Collection<string>();
    SynchronizationResult sResult = new SynchronizationResult();

    try
    {
        string myScopeKey = scopeKey;

        string[] roleMembers = { "Member1", "Member2", "Member3" };
        foreach (string roleMember in roleMembers)
        {
            members.Add(roleMember);
        }

        return members;
    }
    catch (Exception e)
    {
        if (e.Message.Contains("InvalidRole"))
        {
            sResult.Errors.Add(new SynchronizationError("Invalid role: " + roleKey.ToString()));

            throw new SynchronizationException(sResult);
        }
        else
        {
            // Unknown error, so re-throw
            throw;
        }
    }
}
```

CreateApplicationRoleAssignmentImplementation()

This method handles adding specific users to roles within the application. It returns a SynchronizationResult object and takes an ApplicationRoleAssignment object as a parameter (see the section titled “Additional classes available to developers” for more information on these objects).

Typically, implementation will involve invoking a web service to create the role assignment.

```
protected override SynchronizationResult
CreateApplicationRoleAssignmentImplementation(ApplicationRoleAssignment applicationRoleAssignment)
{
    SynchronizationResult sResult = new SynchronizationResult();
```

```
    string myScopeKey = applicationRoleAssignment.Scope;

    RoleAssignment myRoleAssignment = new RoleAssignment();

    myRoleAssignment.RoleId = new Guid(applicationRoleAssignment.Role);
    myRoleAssignment.ContainerKey = myScopeKey;

    foreach (string member in applicationRoleAssignment.Members)
    {
        myRoleAssignment.Member = member;
        try
        {
            // Invoke a web service to create the role assignment in my application
            this.Invoke("CreateApplicationRoleAssignment", new object[] {roleAssignment});
        }
        catch (Exception e)
        {
            if (e.Message.Contains("InvalidMember"))
            {
                // Invalid member, so add it to the list and keep going.
                sResult.Errors.Add(new SynchronizationError("Invalid member: " +
                myRoleAssignment.Member));
            }
            else
            {
                // Unknown error, so re-throw
                throw;
            }
        }
    }

    return sResult;
}
```

BuildPropertyList()

Implementers are required to provide a list of of properties that can be configured. Declare a variable of type `List<ApplicationRoleProviderConfigurationValues>`. For each property, declare a `ApplicationRoleProviderConfigurationValues`.

```
protected override List<ApplicationRoleProviderConfigurationValues> BuildPropertyList()
    {
        List<ApplicationRoleProviderConfigurationValues> propertyList = new
        List<ApplicationRoleProviderConfigurationValues>();

        ApplicationRoleProviderConfigurationValues urlValues;

        urlValues = new ApplicationRoleProviderConfigurationValues();

        //this is the actual property name, not a display name
        urlValues.PropertyName = "Url";
```

```
        urlValues.PropertyType = typeof(string);
        urlValues.PropertyValue = //return value from property getter;
        urlValues.PropertyIsRequired = true;
        urlValues.PropertyExample = "http://www.Url.com";

        propertyList.Add(urlValues);

        return propertyList;
    }
```

For this type, the following properties should be set:

PropertyName

The name of the property as it appears in the property declaration.

PropertyDisplayName

The property name as it should appear in the synchronization utility user experience.

PropertyType

This is typically set to typeof(string).

PropertyValue

This is the string value of the property.

PropertyIsRequired

A Boolean that defines whether the property is required to have a value before the provider can write to the configuration file.

PropertyExample

A string displayed in the synchronization utility user experience to demonstrate the correct formatting of the value to be written to the configuration file.

> **Note:** We recommend that all providers implement a property that lists unsupported Windows identities. These are identities reserved for Windows. Any application that tries to synchronize users with another application should not contain a user with an ID that matches a reserved Windows ID, or the synchronization will fail. An example of such an identity is SHAREPOINT\System.

ValidateProperty_Custom()

Performs vaildations on property values set through the user experience. See the section titled Creating a custom validation for more information on this method.

Available Base Classes

If you are building an application based on either Windows SharePoint Services or Microsoft Dynamics GP Web Services, consider deriving from the following classes, both of which are contained in the Microsoft.Dynamics.Security.Synchronization assembly and derive from ApplicationRoleProvider.

WSSApplicationRoleProviderBase

If you are building an application based on Windows SharePoint Services, you can derive from WSSApplicationRoleProvider, which implements many of the methods necessary for interfacing with SharePoint.

DynamicsSecurityServiceRoleProviderBase

If you are building an application based on Microsoft Dynamics GP Web Services, you can derive from DynamicsSecurityServiceRoleProviderBase, which implements many of the methods necessary for interfacing with Microsoft Dynamics GP Web Services.

Additional classes available to developers

This section outlines some other classes contained in the Microsoft.Dynamics.Security.Synchronization assembly that are available to be used by your provider objects.

ApplicationRoleAssignment

This class provides a way to connect specific roles and the members of those roles to a scope within an application. Note that this class can be utilized for applications that do not support scopes. A scope is an area in which a role is contained, such as a company or other business unit.

```
namespace Microsoft.Dynamics.Security.Synchronization
{
    public class ApplicationRoleAssignment
    {
        // Constructor
        public ApplicationRoleAssignment()
        {
            members = new Collection<string>();
        }
        // Properties
        // Application to which the assignment is associated.
        public Guid Application
        {
            get {…}
            set {…}
        }
        // In Microsoft Dynamics GP and Business Portal scope refers to the specific company
        // to which a role has access.
        public string Scope
        {
            get {…}
            set {…}
        }
        // The role within the application which may be assigned.
        public string Role
        {
            get {…}
            set {…}
        }
        // A collection containing all the users assigned to a role.
        public Collection<string> Members
        {
            get {…}
        }
    }
}
```

RoleSummary

This class contains properties useful to a role, including a key, a display Id and description. It derives from Microsoft.Dynamics.Security.Synchronization.SummaryObject:

```
public class SummaryObject
{
    // Properties
    // Unique identifier for a role
    public string Key
    {
        get { return key; }
        set { key = value; }
    }
    // Identifier that is displayed in the user experience
    public string DisplayId
    {
        get { return displayId; }
        set { displayId = value; }
    }
    // Description of the role
    public string Description
    {
        get { return description; }
    }
}
```

ScopeSummary

This class contains properties useful to a scope, including a key, a display Id and description. It derives from Microsoft.Dynamics.Security.Synchronization.SummaryObject. See the section “Role Summary” above for the properties available in this class.

SynchronizationResult

This class provides a framework with which to collect errors, warnings, and informational messages associated with a provider. Since more than one error/warning/information event can happen while processing a collection of users, scopes, or roles, this object allows them to be collected at runtime and handled by the user experience. You can add `SynchronizationException` objects to one of three collections (of types `SynchronizationError`, `SynchronizationWarning` and `SynchronizationInformation`).

```
[Serializable]
public class SynchronizationResult
{
    // Default constructor.
    public SynchronizationResult() {…}

    // Properties
    public SynchronizationItemCollection<SynchronizationError> Errors
    {
        get {…}
```

```
    }

    public SynchronizationItemCollection<SynchronizationWarning> Warnings
    {
        get {…}
    }

    public SynchronizationItemCollection<SynchronizationInformation> Information
    {
        get {…}
    }

    // Returns true if a synchronization produced errors.
    public bool HasErrors
    {
        get {…}
    }

    // Returns true if a synchronization produced warnings.
    public bool HasWarnings
    {
        get {…}
    }

    // Returns true is a synchronization produced informational messages.
    public bool HasInformation
    {
        get {…}
    }

    // Methods

    // Adds the items in a seperate SynchronizationResult to this SynchronizationResult,
    // including errors, warnings and information.
    public void AddResult(SynchronizationResult result) {…}

    // Actively determines the validation exception message based on contained errors and
    //warnings.
    public override string  ToString(){…}
}
```

SynchronizationException

This class provides exception handling functionality for providers and can be added to a `SynchronizationResult` object for collection. It derives from System.Exception.

An example of exception handling can be found in the code sample in the section titled “A sample provider”.

```
[Serializable]
public class SynchronizationException : Exception
{
```

```
    // Default constructor
    public SynchronizationException() {…}

    // Default constructor for setting SynchronizationResult.
    public SynchronizationException(SynchronizationResult result) {…}

    // Constructor for setting the Message property.
    public SynchronizationException(string message) {…}

    // Constructor for setting the Message and SynchronizationResult properties.
    public SynchronizationException(SynchronizationResult synchronizationResult, string message) {…}

    // Constructor used with serialization.
    // <param name="info">The SerializationInfo into which the transparent proxy is
    serialized.</param>
    // <param name="context">The source and destination of the serialization.</param>
    protected SynchronizationException(SerializationInfo info, StreamingContext context):
    base(info, context) {…}

    // Constructor used to specify a message and inner exception.
    public SynchronizationException(string message, Exception innerException): base(message,
    innerException) {…}
    #endregion

    // Properties

    // The SynchronizationResult containing the SynchronizationItems that resulted in this
    // exception.
    public SynchronizationResult SynchronizationResult
    {
        get {…}
        set {…}
    }

    // The overrridden message for the exception.
    public override string Message
    {
        get {…}
    }

    // Methods

    // Adds the transparent proxy of the object represented by the current instance of
    // RealProxy to the specified SerializationInfo.
    // <param name="info">The SerializationInfo into which the transparent proxy is
    // serialized.</param>
    // <param name="context">The source and destination of the serialization.</param>
    public override void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        if (info == null) {…}
}
```

Creating a custom validation

The ApplicationRoleProvider class performs basic validations on properties in which the PropertyIsRequired property is set to true, as defined in the BuildPropertyList() method. This validation takes the form of checking the string.IsNullorVoid status of the property value. If a required field's value is null or empty, the synchronization utility user interface will inform the user of the error and discontinue writing the property values to the configuration file. If the property value passes this check, the ValidateProperty_Custom() method is called.

By implementing the ValidateProperty_Custom() method, a provider can validate user input from the synchronization utility before writing property values to the configuration file. While implementing specific validations for each property is not required, it is strongly suggested. Exceptions thrown from validation will be written to the Application Event Log once they are handled by the user experience.

An example of creating a custom validation can be found in the code sample in the section titled A sample provider.

Deploying the provider

The synchronization utility uses reflection to determine which assemblies contain classes that inherit from ApplicationRoleProvider. Only assemblies present in the same location in which the utility is installed will be checked. For this reason, providers must be deployed alongside the synchronization utility.

By default, the synchronization utility is installed in C:\Program Files\Microsoft Dynamics\SecuritySyncUtility.

A sample provider

The following is an example of a provider that derives from the ApplicationRoleProvider class. For the sake of simplicity, key attributes (scope keys, role keys) are presented as strings; it is recommended that a more robust type be used for these properties, such as guids or serialized keys.

Note: You also can copy the sample code from the file attached below. To view attachments, you must be viewing this document in Adobe Reader 6.0 or later, or in the full version of Acrobat. Right-click the paperclip icon to the left of the file name, and choose to open or save the file.

- SampleCode.txt

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Text;
using Microsoft.Dynamics.Security.Synchronization;
using System.Collections.ObjectModel;
using System.Runtime.InteropServices;

namespace SampleProvider
{
    [Guid("6BE6EE8B-941C-4211-92D9-B17F1C2F5D31")]
    public class MyApplicationRoleProvider : ApplicationRoleProvider
    {
        private Guid id;
```

```
private Dictionary<string, ApplicationRoleProviderConfigurationValues> propertyList;
private ApplicationRoleProviderOptions supportedOptions;

Attribute IMyApplicationRoleProviderAttribute =
Attribute.GetCustomAttribute(typeof(MyApplicationRoleProvider), typeof(GuidAttribute));

public MyApplicationRoleProvider() : base()
{
    id = new Guid(((GuidAttribute)IMyApplicationRoleProviderAttribute).Value);
    propertyList = base.CachePropertyValuesFromConfig();
}

public override Guid Id
{
    get { return id; }
}

public override string Name
{
    get { return "MyApplicationRoleProvider"; }
}

public override Dictionary<string, ApplicationRoleProviderConfigurationValues>
PropertyList
{
    get { return propertyList; }
}

public string Url
{
    get { return "http://server_name:port/SampleProvider/SampleProviderService.asmx"; }
}

public override ApplicationRoleProviderOptions GetSupportedOptions()
{
    if (supportedOptions == null)
    {
        supportedOptions = new ApplicationRoleProviderOptions();
        supportedOptions.DoesSupportDomainGroups = false;
        supportedOptions.DoesSupportEnterpriseWideScope = true;
        supportedOptions.DoesSupportScopes = true;
        supportedOptions.DoesRequireExistingApplicationUser = false;
        supportedOptions.DoesSupportScopeSpecificRoles = false;
    }

    return supportedOptions;
}

protected override List<ApplicationRoleProviderConfigurationValues> BuildPropertyList()
{
    List<ApplicationRoleProviderConfigurationValues> apcValuesList = new
    List<ApplicationRoleProviderConfigurationValues>();
```

```
    ApplicationRoleProviderConfigurationValues urlValues;
    ApplicationRoleProviderConfigurationValues unsupportedIdsValues;

    urlValues = new ApplicationRoleProviderConfigurationValues();
    urlValues.PropertyName = "Url";
    urlValues.PropertyType = typeof(string);
    urlValues.PropertyValue = "";
    urlValues.PropertyIsRequired = true;
    urlValues.PropertyExample =
    "http://server_name:port/SampleProvider/SampleProviderService.asmx";
    urlValues.PropertyDisplayName = "Sample Provider Web Service URL ";

    unsupportedIdsValues = new ApplicationRoleProviderConfigurationValues();
    unsupportedIdsValues.PropertyName = "UnsupportedWindowsIdentities";
    unsupportedIdsValues.PropertyType = typeof(Collection<string>);

    StringBuilder unsupportedIdentities = new StringBuilder();

    foreach (string item in UnsupportedWindowsIdentities)
    {
        if (item != string.Empty)
        {
            if (unsupportedIdentities.ToString().Length > 0)
            {
                unsupportedIdentities.Append(",");
            }

            unsupportedIdentities.Append(item);
        }
    }

    unsupportedIdsValues.PropertyValue = unsupportedIdentities.ToString();
    unsupportedIdsValues.PropertyIsRequired = false;
    unsupportedIdsValues.PropertyExample = "SHAREPOINT\\System,etc. (comma-delimited
    list)";
    unsupportedIdsValues.PropertyDisplayName = "Unsupported Windows Identities";


    apcValuesList.Add(urlValues);
    apcValuesList.Add(unsupportedIdsValues);

    return apcValuesList;
}

protected override SynchronizationResult
CreateApplicationRoleAssignmentImplementation(ApplicationRoleAssignment
applicationRoleAssignment)
{
    SynchronizationResult synchronizationResult = new SynchronizationResult();

    string myScopeKey = applicationRoleAssignment.Scope;

    RoleAssignment myRoleAssignment = new RoleAssignment();
```

```
        myRoleAssignment.RoleId = new Guid(applicationRoleAssignment.Role);
        myRoleAssignment.ContainerKey = myScopeKey;

        foreach (string member in applicationRoleAssignment.Members)
        {
            myRoleAssignment.Member = member;
            try
            {
                //implement code here to create roles within your application
            }
            catch (Exception e)
            {
                if (e.Message.Contains("InvalidMember"))
                {
                    // Invalid member, so add it to the list and keep going.
                    synchronizationResult.Errors.Add(new SynchronizationError("Invalid member:
                    " + myRoleAssignment.Member));
                }
                else if (e.Message.Contains("InvalidRole"))
                {
                    synchronizationResult.Errors.Add(new SynchronizationError("Invalid role:"
                    + myRoleAssignment.RoleId));
                    throw new SynchronizationException(synchronizationResult);
                }
                else if (e.Message.Contains("InvalidContainerKey"))
                {
                    synchronizationResult.Errors.Add(new SynchronizationError("Invalid
                    container key: " + myScopeKey));
                    throw new SynchronizationException(synchronizationResult);
                }
                else
                {
                    // Unknown error, so re-throw
                    throw;
                }
            }
        }

        return synchronizationResult;
    }

    protected override Collection<string> GetRoleMembersImplementation(string roleKey, string
    scopeKey)
    {
        Collection<string> members = new Collection<string>();
        SynchronizationResult synchronizationResult = new SynchronizationResult();

        try
        {
            string myScopeKey = scopeKey;

            // Role members of Project Mgr role of the Sample Provider
```

```
            string[] roleMembers = { "domain\\user1", "domain\\user2", "domain\\user3",
            "domain\\user4" };
            foreach (string roleMember in roleMembers)
            {
                members.Add(roleMember);
            }

            return members;
        }
        catch (Exception e)
        {
            if (e.Message.Contains("InvalidRole"))
            {
                synchronizationResult.Errors.Add(new SynchronizationError("Invalid role: " +
                roleKey.ToString()));
                throw new SynchronizationException(synchronizationResult);
            }
            else if (e.Message.Contains("InvalidContainerKey"))
            {
                synchronizationResult.Errors.Add(new SynchronizationError("Invalid container
                key: " + scopeKey.ToString()));
                throw new SynchronizationException(synchronizationResult);
            }
            else
            {
                // Unknown error, so re-throw
                throw;
            }
        }
    }

    protected override Collection<RoleSummary> GetRolesImplementation(string scopeKey)
    {
        Collection<RoleSummary> roles = new Collection<RoleSummary>();

        //Create the role info objects and add them to the collection
        RoleSummary roleSummary = new RoleSummary();
        roleSummary.Key = "MyRoleKey1";
        roleSummary.DisplayId = "Project Mgr Role";
        roleSummary.Description = "Project Mgr Role of Sample Provider";
        roles.Add(roleSummary);

        return roles;
    }

    protected override Collection<ScopeSummary> GetScopesImplementation()
    {
        Collection<ScopeSummary> scopes = new Collection<ScopeSummary>();

        //Create the scope info objects and add them to the collection
        ScopeSummary scopeSummary = new ScopeSummary();
        scopeSummary.Key = "MyScopeKey1";
        scopeSummary.DisplayId = "MyScope1_DisplayName";
```

```
            scopeSummary.Description = "This is the description for MyScope1";
            scopes.Add(scopeSummary);

            return scopes;
        }

        protected override void ValidateProperty_Custom(string propertyName, string
        propertyDisplayName, string propertyValue)
        {
            // Test this ValidateProperty_Custom method...
            // Check that the URL entered by user is the valid asmx page...

            if
                (propertyName.Contains("Url") && !propertyValue.Contains(Url))
                throw new FormatException("You must provide the path to the asmx page");
        }
    }

    class RoleAssignment
    {
        private Guid roleId;
        private string containerKey;
        private string member;

        public Guid RoleId
        {
            get
            {
                return this.roleId;
            }
            set
            {
                this.roleId = value;
            }
        }
        public string ContainerKey
        {
            get
            {
                return this.containerKey;
            }
            set
            {
                this.containerKey = value;
            }
        }
        public string Member
        {
            get
            {
                return this.member;
            }
```

```
            set
            {
                this.member = value;
            }
        }
    }
}
```

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics