Helper Code: ServerConnection Class

[Applies to: Microsoft Dynamics CRM 2011]

The primary purpose of the **ServerConnection** class is to show how to connect to the Microsoft Dynamics CRM 2011 web services to invoke web methods. In addition, applications must typically perform other tasks such as obtaining server and organization information, obtaining user login credentials, creating a service proxy, and refreshing the WCF connection security token. The **ServerConnection** class provides this needed functionality.

Caution

The **ServerConnection** class is used by most samples that ship with the Microsoft Dynamics CRM SDK. The class is updated periodically with new functionality. Do not simply reuse the helper code for authentication in your applications. It is code that the Microsoft Dynamics CRM SDK uses to provide the optimum experience when you run our Console application samples included in the SDK. It contains all the key elements of authentication and demonstrates their use, but it may not represent the best solution for your application. It is sample code that you can use as a basis for designing an authentication management system that fits the requirements of your application. Refer to the topic Simplified Connection to Microsoft Dynamics CRM for an alternate method for authenticating with the web services.

This sample code can be found in the following location in the SDK download:

`SDK\SampleCode\CS\HelperCode\CrmServiceHelpers.cs`

`SDK\SampleCode\VB\HelperCode\CrmServiceHelpers.vb`

Use the class source code in the CrmServiceHelpers files as a basis for your own classes or use the CrmConnection class for just the basic functionality of setting up a service connection.

Requirements

For more information about the requirements for running the sample code provided in this SDK, see Use the Sample and Helper Code.

Demonstrates

The **ServerConnection** class demonstrates how to establish a connection to the Microsoft Dynamics CRM 2011 web services for the purpose of invoking web methods. Before a connection can be created, information about the server, organization, and user must first be obtained. This information is gathered in either of two ways: by interactively prompting the user in the console window, or by loading saved server configuration information from the local disk. The server connection information is persisted to a file named Credentials.xml located at `C:\Users\<username>\AppData\Roaming\CrmServer`.

For each server configuration saved in the Credentials.xml file, the user's logon password is stored as a generic credential in the Windows Credential Manager. The credential naming format is Microsoft_CRMSDK:*<server>*:*<organization>*:*<userID>*.

Useful code for authentication can be found in the **GetProxy** and **GetOrganizationProxy** methods. Also, the code that creates and reads a user's password in the Windows Credential Manager may be of interest.

Example

C# VB

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.DirectoryServices.AccountManagement;
using System.IO;
using System.Runtime.InteropServices;
using System.Security;
using System.ServiceModel;
using System.ServiceModel.Description;
using System.Text;
```

```
using System.Xml;
using System.Xml.Linq;

// These namespaces are found in the Microsoft.Xrm.Sdk.dll assembly
// located in the SDK\bin folder of the SDK download.
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Client;
using Microsoft.Xrm.Sdk.Discovery;
using Microsoft.Crm.Services.Utility;


namespace Microsoft.Crm.Sdk.Samples
{
    /// <summary>
    /// Provides server connection information.
    /// </summary>
    public class ServerConnection
    {
        #region Inner classes
        /// <summary>
        /// Stores Microsoft Dynamics CRM server configuration information.
        /// </summary>
        public class Configuration
        {
            public String ServerAddress;
            public String OrganizationName;
            public Uri DiscoveryUri;
            public Uri OrganizationUri;
            public Uri HomeRealmUri = null;
            public ClientCredentials DeviceCredentials = null;
            public ClientCredentials Credentials = null;
            public AuthenticationProviderType EndpointType;
            public String UserPrincipalName;
            #region internal members of the class
            internal IServiceManagement<IOrganizationService> OrganizationServiceManage
            internal SecurityTokenResponse OrganizationTokenResponse;
            internal Int16 AuthFailureCount = 0;
            #endregion

            public override bool Equals(object obj)
            {
                //Check for null and compare run-time types.
                if (obj == null || GetType() != obj.GetType()) return false;

                Configuration c = (Configuration)obj;

                if (!this.ServerAddress.Equals(c.ServerAddress, StringComparison.Invari
                    return false;
                if (!this.OrganizationName.Equals(c.OrganizationName, StringComparison.
                    return false;
                if (this.EndpointType != c.EndpointType)
                    return false;
                if (null != this.Credentials && null != c.Credentials)
                {
                    if (this.EndpointType == AuthenticationProviderType.ActiveDirectory
                    {

                        if (!this.Credentials.Windows.ClientCredential.Domain.Equals(
                            c.Credentials.Windows.ClientCredential.Domain, StringCompar
                            return false;
                        if (!this.Credentials.Windows.ClientCredential.UserName.Equals(
                            c.Credentials.Windows.ClientCredential.UserName, StringComp
                            return false;

                    }
                    else if (this.EndpointType == AuthenticationProviderType.LiveId)
                    {
                        if (!this.Credentials.UserName.UserName.Equals(c.Credentials.Us
                            StringComparison.InvariantCultureIgnoreCase))
                            return false;
                        if (!this.DeviceCredentials.UserName.UserName.Equals(
                            c.DeviceCredentials.UserName.UserName, StringComparison.Inv
                            return false;
                        if (!this.DeviceCredentials.UserName.Password.Equals(
                            c.DeviceCredentials.UserName.Password, StringComparison.Inv
                            return false;
                    }
                    else
                    {
```

```
                if (!this.Credentials.UserName.UserName.Equals(c.Credentials.Us
                    StringComparison.InvariantCultureIgnoreCase))
                    return false;

            }
        }
        return true;
    }

    public override int GetHashCode()
    {
        int returnHashCode = this.ServerAddress.GetHashCode()
            ^ this.OrganizationName.GetHashCode()
            ^ this.EndpointType.GetHashCode();
        if (null != this.Credentials)
        {
            if (this.EndpointType == AuthenticationProviderType.ActiveDirectory
                returnHashCode = returnHashCode
                    ^ this.Credentials.Windows.ClientCredential.UserName.GetHas
                    ^ this.Credentials.Windows.ClientCredential.Domain.GetHashC
                else if (this.EndpointType == AuthenticationProviderType.LiveId)
                returnHashCode = returnHashCode
                    ^ this.Credentials.UserName.UserName.GetHashCode()
                    ^ this.DeviceCredentials.UserName.UserName.GetHashCode()
                    ^ this.DeviceCredentials.UserName.Password.GetHashCode();
            else
                returnHashCode = returnHashCode
                    ^ this.Credentials.UserName.UserName.GetHashCode();
        }
        return returnHashCode;
    }

}
#endregion Inner classes

#region Public properties

public List<Configuration> configurations = null;

#endregion Public properties

#region Private properties

private Configuration config = new Configuration();

#endregion Private properties

#region Static methods
/// <summary>
/// Obtains the organization service proxy.
/// This would give a better performance than directly calling GetProxy() gener
/// as it uses cached OrganizationServiceManagement in case it is present.
/// </summary>
/// <param name="serverConfiguration">An instance of ServerConnection.Configura
/// <returns>An instance of organization service proxy</returns>
public static OrganizationServiceProxy GetOrganizationProxy(
    ServerConnection.Configuration serverConfiguration)
{
    // If organization service management exists, then use it.
    // Otherwise generate organization service proxy from scratch.
    if (null != serverConfiguration.OrganizationServiceManagement)
    {
        // Obtain the organization service proxy for the Federated, Microsoft a
        if (serverConfiguration.EndpointType != AuthenticationProviderType.Acti
        {
            // get the organization service proxy.
            return GetProxy<IOrganizationService, OrganizationServiceProxy>(ser

        }
        // Obtain organization service proxy for ActiveDirectory environment
        // using existing organization service management.
        else
        {
            return new ManagedTokenOrganizationServiceProxy(
                serverConfiguration.OrganizationServiceManagement,
                serverConfiguration.Credentials);
        }
    }

    // Obtain the organization service proxy for all type of environments.
```

```
            return GetProxy<IOrganizationService, OrganizationServiceProxy>(serverConfi

        }
        #endregion Static methods

        #region Public methods
        /// <summary>
        /// Obtains the server connection information including the target organization
        /// Uri and user logon credentials from the user.
        /// </summary>
        public virtual Configuration GetServerConfiguration()
        {
            Boolean ssl;
            Boolean addConfig;
            int configNumber;
            // Read the configuration from the disk, if it exists, at C:\Users\<usernam
            Boolean isConfigExist = ReadConfigurations();

            // Check if server configuration settings are already available on the disk
            if (isConfigExist)
            {
                // List of server configurations that are available from earlier saved
                Console.Write("\n(0) Add New Server Configuration (Maximum number up to
                for (int n = 0; n < configurations.Count; n++)
                {
                    String user;

                    switch (configurations[n].EndpointType)
                    {
                        case AuthenticationProviderType.ActiveDirectory:
                            if (configurations[n].Credentials != null)
                                user = configurations[n].Credentials.Windows.ClientCred
                                    + configurations[n].Credentials.Windows.ClientCrede
                            else
                                user = "default";
                            break;
                        default:
                            if (configurations[n].Credentials != null)
                                user = configurations[n].Credentials.UserName.UserName;
                            else
                                user = "default";
                            break;
                    }

                    Console.Write("\n({0}) Server: {1},  Org: {2},  User: {3}\t",
                        n + 1, configurations[n].ServerAddress, configurations[n].Organ
                }

                Console.WriteLine();

                Console.Write("\nSpecify the saved server configuration number (1-{0})
                String input = Console.ReadLine();
                Console.WriteLine();
                if (input == String.Empty) input = configurations.Count.ToString();
                if (!Int32.TryParse(input, out configNumber)) configNumber = -1;

                if (configNumber == 0)
                {
                    addConfig = true;
                }
                else if (configNumber > 0 && configNumber <= configurations.Count)
                {
                    // Return the organization Uri.
                    config = configurations[configNumber - 1];
                    // Reorder the configuration list and save it to file to save the r
                    if (configNumber != configurations.Count)
                    {
                        Configuration temp = configurations[configurations.Count - 1];
                        configurations[configurations.Count - 1] = configurations[confi
                        configurations[configNumber - 1] = temp;
                    }
                    addConfig = false;
                }
                else
                    throw new InvalidOperationException("The specified server configura
            }
            else
                addConfig = true;

            if (addConfig)
```

```
        {
            // Get the server address. If no value is entered, default to Microsoft
            // CRM Online in the North American data center.
            config.ServerAddress = GetServerAddress(out ssl);

            if (String.IsNullOrWhiteSpace(config.ServerAddress))
                config.ServerAddress = "crm.dynamics.com";


            // One of the Microsoft Dynamics CRM Online data centers.
            if (config.ServerAddress.EndsWith(".dynamics.com", StringComparison.Inv
            {
                // Check if the organization is provisioned in Microsoft Office 365
                if (GetOrgType(config.ServerAddress))
                {
                config.DiscoveryUri =
                    new Uri(String.Format("https://disco.{0}/XRMServices/2011/Disco
                }
                else
                {
                config.DiscoveryUri =
                    new Uri(String.Format("https://dev.{0}/XRMServices/2011/Discove

                // Get or set the device credentials. This is required for Microsof
                config.DeviceCredentials = GetDeviceCredentials();
                }
            }
            // Check if the server uses Secure Socket Layer (https).
            else if (ssl)
                config.DiscoveryUri =
                    new Uri(String.Format("https://{0}/XRMServices/2011/Discovery.s
            else
                config.DiscoveryUri =
                    new Uri(String.Format("http://{0}/XRMServices/2011/Discovery.sv

            // Get the target organization.
            config.OrganizationUri = GetOrganizationAddress();
            configurations.Add(config);
            int length = configurations.Count;
            int i = length - 2;
            // Check if a new configuration already exists.
            // If found, reorder list to show latest in use.
            while (i > 0)
            {

                if (configurations[configurations.Count - 1].Equals(configurations[
                {
                    configurations.RemoveAt(i);
                }
                i--;
            }
            // Set max configurations to 9 otherwise overwrite existing one.
            if (configurations.Count > 9)
            {
                configurations.RemoveAt(0);
            }
        }
        else
        {
            // Get the existing user's logon credentials.
            config.Credentials = GetUserLogonCredentials(config);
        }
        SaveConfigurations();
        return config;
    }

    /// <summary>
    /// Discovers the organizations that the calling user belongs to.
    /// </summary>
    /// <param name="service">A Discovery service proxy instance.</param>
    /// <returns>Array containing detailed information on each organization that
    /// the user belongs to.</returns>
    public OrganizationDetailCollection DiscoverOrganizations(IDiscoveryService ser
    {
        if (service == null) throw new ArgumentNullException("service");
        RetrieveOrganizationsRequest orgRequest = new RetrieveOrganizationsRequest(
        RetrieveOrganizationsResponse orgResponse =
            (RetrieveOrganizationsResponse)service.Execute(orgRequest);

        return orgResponse.Details;
```

```
}

/// <summary>
/// Finds a specific organization detail in the array of organization details
/// returned from the Discovery service.
/// </summary>
/// <param name="orgFriendlyName">The friendly name of the organization to find
/// <param name="orgDetails">Array of organization detail object returned from
/// <returns>Organization details or null if the organization was not found.</r
/// <seealso cref="DiscoveryOrganizations"/>
public OrganizationDetail FindOrganization(string orgFriendlyName,
    OrganizationDetail[] orgDetails)
{
    if (String.IsNullOrWhiteSpace(orgFriendlyName))
        throw new ArgumentNullException("orgFriendlyName");
    if (orgDetails == null)
        throw new ArgumentNullException("orgDetails");
    OrganizationDetail orgDetail = null;

    foreach (OrganizationDetail detail in orgDetails)
    {
        if (String.Compare(detail.FriendlyName, orgFriendlyName,
            StringComparison.InvariantCultureIgnoreCase) == 0)
        {
            orgDetail = detail;
            break;
        }
    }
    return orgDetail;
}

/// <summary>
/// Reads a server configuration file.
/// Read the configuration from disk, if it exists, at C:\Users\YourUserName\Ap
/// </summary>
/// <returns>Is configuration settings already available on disk.</returns>
public Boolean ReadConfigurations()
{
    Boolean isConfigExist = false;

    if (configurations == null)
        configurations = new List<Configuration>();

    if (File.Exists(CrmServiceHelperConstants.ServerCredentialsFile))
    {
        XElement configurationsFromFile =
            XElement.Load(CrmServiceHelperConstants.ServerCredentialsFile);
        foreach (XElement config in configurationsFromFile.Nodes())
        {
            Configuration newConfig = new Configuration();
            var serverAddress = config.Element("ServerAddress");
            if (serverAddress != null)
                if (!String.IsNullOrEmpty(serverAddress.Value))
                    newConfig.ServerAddress = serverAddress.Value;
            var organizationName = config.Element("OrganizationName");
            if (organizationName != null)
                if (!String.IsNullOrEmpty(organizationName.Value))
                    newConfig.OrganizationName = organizationName.Value;
            var discoveryUri = config.Element("DiscoveryUri");
            if (discoveryUri != null)
                if (!String.IsNullOrEmpty(discoveryUri.Value))
                    newConfig.DiscoveryUri = new Uri(discoveryUri.Value);
            var organizationUri = config.Element("OrganizationUri");
            if (organizationUri != null)
                if (!String.IsNullOrEmpty(organizationUri.Value))
                    newConfig.OrganizationUri = new Uri(organizationUri.Value);
            var homeRealmUri = config.Element("HomeRealmUri");
            if (homeRealmUri != null)
                if (!String.IsNullOrEmpty(homeRealmUri.Value))
                    newConfig.HomeRealmUri = new Uri(homeRealmUri.Value);

            var vendpointType = config.Element("EndpointType");
            if (vendpointType != null)
                newConfig.EndpointType =
                        RetrieveAuthenticationType(vendpointType.Value);
            if (config.Element("Credentials").HasElements)
            {
                newConfig.Credentials =
                    ParseInCredentials(config.Element("Credentials"),
                    newConfig.EndpointType,
```

```
                    newConfig.ServerAddress + ":" + newConfig.OrganizationName
                }
                if (newConfig.EndpointType == AuthenticationProviderType.LiveId)
                {
                    newConfig.DeviceCredentials = GetDeviceCredentials();
                }
                var userPrincipalName = config.Element("UserPrincipalName");
                if (userPrincipalName != null)
                    if (!String.IsNullOrWhiteSpace(userPrincipalName.Value))
                        newConfig.UserPrincipalName = userPrincipalName.Value;
                configurations.Add(newConfig);
            }
        }

        if (configurations.Count > 0)
            isConfigExist = true;

        return isConfigExist;
    }

    /// <summary>
    /// Writes all server configurations to a file.
    /// </summary>
    /// <remarks>If the file exists, it is overwritten.</remarks>
    public void SaveConfigurations()
    {
        if (configurations == null)
            throw new NullReferenceException("No server connection configurations w

        FileInfo file = new FileInfo(CrmServiceHelperConstants.ServerCredentialsFil

        // Create directory if it does not exist.
        if (!file.Directory.Exists)
            file.Directory.Create();

        // Replace the file if it exists.
        using (FileStream fs = file.Open(FileMode.Create, FileAccess.Write, FileSha
        {
            using (XmlTextWriter writer = new XmlTextWriter(fs, Encoding.UTF8))
            {
                writer.Formatting = Formatting.Indented;
                writer.WriteStartDocument();
                writer.WriteStartElement("Configurations");
                writer.WriteFullEndElement();
                writer.WriteEndDocument();
            }
        }

        foreach (Configuration config in configurations)
            SaveConfiguration(CrmServiceHelperConstants.ServerCredentialsFile, conf
    }

    /// <summary>
    /// Writes a server configuration to a file.
    /// </summary>
    /// <param name="pathname">The file name and system path of the output configur
    /// <param name="config">A server connection configuration.</param>
    /// <param name="append">If true, the configuration is appended to the file, ot
    /// is created.</param>
    public void SaveConfiguration(String pathname, Configuration config, bool appen
    {
        if (String.IsNullOrWhiteSpace(pathname)) throw new ArgumentNullException("p
        if (config == null) throw new ArgumentNullException("config");
        // Target is the key with which associated credentials can be fetched from
        String target = config.ServerAddress + ":" + config.OrganizationName;
        if(null != config.Credentials)
        {
            switch(config.EndpointType)
            {
                case AuthenticationProviderType.ActiveDirectory:
                    target = target + ":" + config.Credentials.Windows.ClientCreden
                    break;
                case AuthenticationProviderType.LiveId:
                case AuthenticationProviderType.Federation:
                case AuthenticationProviderType.OnlineFederation:
                    target = target + ":" + config.Credentials.UserName.UserName;
                    break;
                default:
                    target = String.Empty;
                    break;
```

```
            }
        }

        XElement configurationsFromFile = XElement.Load(pathname);
        XElement newConfig =
            new XElement("Configuration",
                new XElement("ServerAddress", config.ServerAddress),
                new XElement("OrganizationName", config.OrganizationName),
                new XElement("DiscoveryUri",
                    (config.DiscoveryUri != null)
                    ? config.DiscoveryUri.OriginalString
                    : String.Empty),
                new XElement("OrganizationUri",
                    (config.OrganizationUri != null)
                    ? config.OrganizationUri.OriginalString
                    : String.Empty),
                new XElement("HomeRealmUri",
                    (config.HomeRealmUri != null)
                    ? config.HomeRealmUri.OriginalString
                    : String.Empty),
                ParseOutCredentials(config.Credentials, config.EndpointType, target
                new XElement("EndpointType", config.EndpointType.ToString()),
                new XElement("UserPrincipalName",
                    (config.UserPrincipalName != null)
                    ? config.UserPrincipalName
                    : String.Empty)
            );

        if (append)
        {
            configurationsFromFile.Add(newConfig);
        }
        else
        {
            configurationsFromFile.ReplaceAll(newConfig);
        }

        using (XmlTextWriter writer = new XmlTextWriter(pathname, Encoding.UTF8))
        {
            writer.Formatting = Formatting.Indented;
            configurationsFromFile.Save(writer);
        }
    }

    /// <summary>
    /// Obtains the user's logon credentials for the target server.
    /// </summary>
    /// <param name="config">An instance of the Configuration.</param>
    /// <returns>Logon credentials of the user.</returns>
    public static ClientCredentials GetUserLogonCredentials(ServerConnection.Config
    {
        ClientCredentials credentials = new ClientCredentials();
        String userName;
        SecureString password;
        String domain;
        Boolean isCredentialExist = (config.Credentials != null) ? true : false;
        switch (config.EndpointType)
        {
            // An on-premises Microsoft Dynamics CRM server deployment.
            case AuthenticationProviderType.ActiveDirectory:
                // Uses credentials from windows credential manager for earlier sav
                if (isCredentialExist && !String.IsNullOrWhiteSpace(config.Organiza
                {
                    domain = config.Credentials.Windows.ClientCredential.Domain;
                    userName = config.Credentials.Windows.ClientCredential.UserName
                    if (String.IsNullOrWhiteSpace(config.Credentials.Windows.Client
                    {
                        Console.Write("\nEnter domain\\username: ");
                        Console.WriteLine(
                        config.Credentials.Windows.ClientCredential.Domain + "\\"
                        + config.Credentials.Windows.ClientCredential.UserName);

                        Console.Write("       Enter Password: ");
                        password = ReadPassword();
                    }
                    else
                    {
                        password = config.Credentials.Windows.ClientCredential.Secu
                    }
                }
```

```
            // Uses default credentials saved in windows credential manager for
            else if (!isCredentialExist && !String.IsNullOrWhiteSpace(config.Or
            {
                return null;
            }
            // Prompts users to enter credential for current organization.
            else
            {
                String[] domainAndUserName;
                do
                {
                    Console.Write("\nEnter domain\\username: ");
                    domainAndUserName = Console.ReadLine().Split('\\');

                    // If user do not choose to enter user name,
                    // then try to use default credential from windows credenti
                    if (domainAndUserName.Length == 1 && String.IsNullOrWhiteSp
                    {
                        return null;
                    }
                }
                while (domainAndUserName.Length != 2 || String.IsNullOrWhiteSpa
                    || String.IsNullOrWhiteSpace(domainAndUserName[1]));

                domain = domainAndUserName[0];
                userName = domainAndUserName[1];

                Console.Write("       Enter Password: ");
                password = ReadPassword();
            }
            if (null != password)
            {
                credentials.Windows.ClientCredential =
                    new System.Net.NetworkCredential(userName, password, domain
            }
            else
            {
                credentials.Windows.ClientCredential = null;
            }

            break;
        // A Microsoft Dynamics CRM Online server deployment.
        case AuthenticationProviderType.LiveId:
        // An internet-facing deployment (IFD) of Microsoft Dynamics CRM.
        case AuthenticationProviderType.Federation:
        // Managed Identity/Federated Identity users using Microsoft Office 365
        case AuthenticationProviderType.OnlineFederation:
            // Use saved credentials.
            if (isCredentialExist)
            {
                userName = config.Credentials.UserName.UserName;
                if (String.IsNullOrWhiteSpace(config.Credentials.UserName.Passw
                {
                    Console.Write("\n Enter Username: ");
                    Console.WriteLine(config.Credentials.UserName.UserName);

                    Console.Write(" Enter Password: ");
                    password = ReadPassword();
                }
                else
                {
                    password = ConvertToSecureString(config.Credentials.UserNam
                }
            }
            // For OnlineFederation environments, initially try to authenticate
            // for single sign-on scenario.
            else if (config.EndpointType == AuthenticationProviderType.OnlineFe
                && config.AuthFailureCount == 0
                && !String.IsNullOrWhiteSpace(UserPrincipal.Current.UserPrincip
            {
                config.UserPrincipalName = UserPrincipal.Current.UserPrincipalN
                return null;
            }
            // Otherwise request username and password.
            else
            {
                config.UserPrincipalName = String.Empty;
                if (config.EndpointType == AuthenticationProviderType.LiveId)
                    Console.Write("\n Enter Microsoft account: ");
                else
```

```
                    Console.Write("\n Enter Username: ");
                userName = Console.ReadLine();
                if (string.IsNullOrWhiteSpace(userName))
                {
                    return null;
                }

                Console.Write(" Enter Password: ");
                password = ReadPassword();
            }
            credentials.UserName.UserName = userName;
            credentials.UserName.Password = ConvertToUnsecureString(password);
            break;
        default:
            credentials = null;
            break;
    }
    return credentials;
}

/// <summary>
/// Prompts user to enter password in console window
/// and capture the entered password into SecureString.
/// </summary>
/// <returns>Password stored in a secure string.</returns>
public static SecureString ReadPassword()
{
    SecureString ssPassword = new SecureString();

    ConsoleKeyInfo info = Console.ReadKey(true);
    while (info.Key != ConsoleKey.Enter)
    {
        if (info.Key == ConsoleKey.Backspace)
        {
            if (ssPassword.Length != 0)
            {
                ssPassword.RemoveAt(ssPassword.Length - 1);
                Console.Write("\b \b");     // erase last char
            }
        }
        else if (info.KeyChar >= ' ')           // no control chars
        {
            ssPassword.AppendChar(info.KeyChar);
            Console.Write("*");
        }
        info = Console.ReadKey(true);
    }

    Console.WriteLine();
    Console.WriteLine();

    // Lock the secure string password.
    ssPassword.MakeReadOnly();

    return ssPassword;
}

/// <summary>
/// Generic method to obtain discovery/organization service proxy instance.
/// </summary>
/// <typeparam name="TService">
/// Set IDiscoveryService or IOrganizationService type
/// to request respective service proxy instance.
/// </typeparam>
/// <typeparam name="TProxy">
/// Set the return type to either DiscoveryServiceProxy
/// or OrganizationServiceProxy type based on TService type.
/// </typeparam>
/// <param name="currentConfig">An instance of existing Configuration</param>
/// <returns>An instance of TProxy
/// i.e. DiscoveryServiceProxy or OrganizationServiceProxy</returns>
public static TProxy GetProxy<TService, TProxy>(ServerConnection.Configuration
    where TService : class
    where TProxy : ServiceProxy<TService>
{
    // Check if it is organization service proxy request.
    Boolean isOrgServiceRequest = typeof(TService).Equals(typeof(IOrganizationS

    // Get appropriate Uri from Configuration.
    Uri serviceUri = isOrgServiceRequest ?
```

```
        currentConfig.OrganizationUri : currentConfig.DiscoveryUri;

    // Set service management for either organization service Uri or discovery
    // For organization service Uri, if service management exists
    // then use it from cache. Otherwise create new service management for curr
    IServiceManagement<TService> serviceManagement =
        (isOrgServiceRequest && null != currentConfig.OrganizationServiceManage
        (IServiceManagement<TService>)currentConfig.OrganizationServiceManageme
        ServiceConfigurationFactory.CreateManagement<TService>(
        serviceUri);

    if (isOrgServiceRequest)
    {
        if (currentConfig.OrganizationTokenResponse == null)
        {
            currentConfig.OrganizationServiceManagement =
                (IServiceManagement<IOrganizationService>)serviceManagement;
        }
    }
    // Set the EndpointType in the current Configuration object
    // while adding new configuration using discovery service proxy.
    else
    {
        // Get the EndpointType.
        currentConfig.EndpointType = serviceManagement.AuthenticationType;
        // Get the logon credentials.
        currentConfig.Credentials = GetUserLogonCredentials(currentConfig);
    }

    // Set the credentials.
    AuthenticationCredentials authCredentials = new AuthenticationCredentials()

    // If UserPrincipalName exists, use it. Otherwise, set the logon credential
    if (!String.IsNullOrWhiteSpace(currentConfig.UserPrincipalName))
    {
        // Single sing-on with the Federated Identity organization using curren
        authCredentials.UserPrincipalName = currentConfig.UserPrincipalName;
    }
    else
    {
        authCredentials.ClientCredentials = currentConfig.Credentials;
    }

    Type classType;

    // Obtain discovery/organization service proxy for Federated,
    // Microsoft account and OnlineFederated environments.
    if (currentConfig.EndpointType !=
        AuthenticationProviderType.ActiveDirectory)
    {
        if (currentConfig.EndpointType == AuthenticationProviderType.LiveId)
        {
            authCredentials.SupportingCredentials = new AuthenticationCredentia
            authCredentials.SupportingCredentials.ClientCredentials =
                currentConfig.DeviceCredentials;
        }

        AuthenticationCredentials tokenCredentials =
            serviceManagement.Authenticate(
                authCredentials);

           if (isOrgServiceRequest)
        {
            // Set SecurityTokenResponse for the current organization.
            currentConfig.OrganizationTokenResponse = tokenCredentials.Security
            // Set classType to ManagedTokenOrganizationServiceProxy.
            classType = typeof(ManagedTokenOrganizationServiceProxy);

        }
        else
        {
            // Set classType to ManagedTokenDiscoveryServiceProxy.
            classType = typeof(ManagedTokenDiscoveryServiceProxy);
        }

        // Invokes ManagedTokenOrganizationServiceProxy or ManagedTokenDiscover
        // (IServiceManagement<TService>, SecurityTokenResponse) constructor.
        return (TProxy)classType
        .GetConstructor(new Type[]
            {
```

```
                typeof(IServiceManagement<TService>),
                typeof(SecurityTokenResponse)
            })
        .Invoke(new object[]
            {
                serviceManagement,
                tokenCredentials.SecurityTokenResponse
            });
    }

    // Obtain discovery/organization service proxy for ActiveDirectory environm
    if (isOrgServiceRequest)
    {
        classType = typeof(ManagedTokenOrganizationServiceProxy);
    }
    else
    {
        classType = typeof(ManagedTokenDiscoveryServiceProxy);
    }

    // Invokes ManagedTokenDiscoveryServiceProxy or ManagedTokenOrganizationSer
    // (IServiceManagement<TService>, ClientCredentials) constructor.
    return (TProxy)classType
        .GetConstructor(new Type[]
           {
               typeof(IServiceManagement<TService>),
               typeof(ClientCredentials)
           })
       .Invoke(new object[]
           {
               serviceManagement,
               authCredentials.ClientCredentials
           });
}

/// <summary>
/// Convert SecureString to unsecure string.
/// </summary>
/// <param name="securePassword">Pass SecureString for conversion.</param>
/// <returns>unsecure string</returns>
public static String ConvertToUnsecureString(SecureString securePassword)
{
    if (securePassword == null)
        throw new ArgumentNullException("securePassword");

    IntPtr unmanagedString = IntPtr.Zero;
    try
    {
        unmanagedString = Marshal.SecureStringToGlobalAllocUnicode(securePasswo
        return Marshal.PtrToStringUni(unmanagedString);
    }
    finally
    {
        Marshal.ZeroFreeGlobalAllocUnicode(unmanagedString);
    }
}

/// <summary>
/// Convert unsecure string to SecureString.
/// </summary>
/// <param name="password">Pass unsecure string for conversion.</param>
/// <returns>SecureString</returns>
public static SecureString ConvertToSecureString(string password)
{
    if (password == null)
        throw new ArgumentNullException("password");

    var securePassword = new SecureString();
    foreach (char c in password)
        securePassword.AppendChar(c);
    securePassword.MakeReadOnly();
    return securePassword;
}
#endregion Public methods

#region Protected methods

/// <summary>
/// Obtains the name and port of the server running the Microsoft Dynamics CRM
/// Discovery service.
```

```
        /// </summary>
        /// <returns>The server's network name and optional TCP/IP port.</returns>
        protected virtual String GetServerAddress(out bool ssl)
        {
            ssl = false;

            Console.Write("Enter a CRM server name and port [crm.dynamics.com]: ");
            String server = Console.ReadLine();

            if (server.EndsWith(".dynamics.com") || String.IsNullOrWhiteSpace(server))
            {
                ssl = true;
            }
            else
            {
                Console.Write("Is this server configured for Secure Socket Layer (https
                String answer = Console.ReadLine();

                if (answer == "y" || answer == "Y")
                    ssl = true;
            }

            return server;
        }

        /// <summary>
        /// Is this organization provisioned in Microsoft Office 365?
        /// </summary>
        /// <param name="server">The server's network name.</param>
        protected virtual Boolean GetOrgType(String server)
        {
            Boolean isO365Org = false;
            if (String.IsNullOrWhiteSpace(server))
                return isO365Org;
            if (server.IndexOf('.') == -1)
                return isO365Org;

            Console.Write("Is this organization provisioned in Microsoft Office 365 (y/
            String answer = Console.ReadLine();

            if (answer == "y" || answer == "Y")
                isO365Org = true;

            return isO365Org;
        }

        /// <summary>
        /// Obtains the web address (Uri) of the target organization.
        /// </summary>
        /// <returns>Uri of the organization service or an empty string.</returns>
        protected virtual Uri GetOrganizationAddress()
        {
            using (DiscoveryServiceProxy serviceProxy = GetDiscoveryProxy())
            {
                // Obtain organization information from the Discovery service.
                if (serviceProxy != null)
                {
                    // Obtain information about the organizations that the system user
                    OrganizationDetailCollection orgs = DiscoverOrganizations(servicePr

                    if (orgs.Count > 0)
                    {
                        Console.WriteLine("\nList of organizations that you belong to:"
                        for (int n = 0; n < orgs.Count; n++)
                        {
                            Console.Write("\n({0}) {1} ({2})\t", n + 1, orgs[n].Friendl
                        }

                        Console.Write("\n\nSpecify an organization number (1-{0}) [1]:
                        String input = Console.ReadLine();
                        if (input == String.Empty)
                        {
                            input = "1";
                        }
                        int orgNumber;
                        Int32.TryParse(input, out orgNumber);
                        if (orgNumber > 0 && orgNumber <= orgs.Count)
                        {
                            config.OrganizationName = orgs[orgNumber - 1].FriendlyName;
                            // Return the organization Uri.
```

```
                    return new System.Uri(orgs[orgNumber - 1].Endpoints[Endpoin
                }
                else
                    throw new InvalidOperationException("The specified organiza
            }
            else
            {
                Console.WriteLine("\nYou do not belong to any organizations on
                return new System.Uri(String.Empty);
            }
        }
        else
            throw new InvalidOperationException("An invalid server name was spe
    }
}

/// <summary>
/// Get the device credentials by either loading from the local cache
/// or request new device credentials by registering the device.
/// </summary>
/// <returns>Device Credentials.</returns>
protected virtual ClientCredentials GetDeviceCredentials()
{
    return Microsoft.Crm.Services.Utility.DeviceIdManager.LoadOrRegisterDevice(
}

/// <summary>
/// Get the discovery service proxy based on existing configuration data.
/// Added new way of getting discovery proxy.
/// Also preserving old way of getting discovery proxy to support old scenarios
/// </summary>
/// <returns>An instance of DiscoveryServiceProxy</returns>
private DiscoveryServiceProxy GetDiscoveryProxy()
{
    try
    {
        // Obtain the discovery service proxy.
        DiscoveryServiceProxy discoveryProxy = GetProxy<IDiscoveryService, Disc
        // Checking authentication by invoking some SDK methods.
        discoveryProxy.Execute(new RetrieveOrganizationsRequest());
        return discoveryProxy;
    }
    catch (System.ServiceModel.Security.SecurityAccessDeniedException ex)
    {
            // If authentication failed using current UserPrincipalName,
            // request UserName and Password to try to authenticate using user
            if (!String.IsNullOrWhiteSpace(config.UserPrincipalName) &&
                ex.Message.Contains("Access is denied."))
            {
                config.AuthFailureCount += 1;
            }
            else
            {
                throw ex;
            }
    }
    // You can also catch other exceptions to handle a specific situation in yo
    //      System.ServiceModel.Security.ExpiredSecurityTokenException
    //      System.ServiceModel.Security.MessageSecurityException
    //      System.ServiceModel.Security.SecurityNegotiationException

    // Second trial to obtain the discovery service proxy in case of single sig
    return GetProxy<IDiscoveryService, DiscoveryServiceProxy>(this.config);

}

/// <summary>
/// Verify passed strings with the supported AuthenticationProviderType.
/// </summary>
/// <param name="authType">String AuthenticationType</param>
/// <returns>Supported AuthenticatoinProviderType</returns>
private AuthenticationProviderType RetrieveAuthenticationType(String authType)
{
    switch (authType)
    {
        case "ActiveDirectory":
            return AuthenticationProviderType.ActiveDirectory;
        case "LiveId":
            return AuthenticationProviderType.LiveId;
        case "Federation":
```

```
                return AuthenticationProviderType.Federation;
            case "OnlineFederation":
                return AuthenticationProviderType.OnlineFederation;
            default:
                throw new ArgumentException(String.Format("{0} is not a valid authe
        }
    }

    /// <summary>
    /// Parse credentials from an XML node to required ClientCredentials data type
    /// based on passed AuthenticationProviderType.
    /// </summary>
    /// <param name="credentials">Credential XML node.</param>
    /// <param name="endpointType">AuthenticationProviderType of the credential.</p
    /// <param name="target">Target is the key with which associated credentials ca
    /// <returns>Required ClientCredentials type.</returns>
    private ClientCredentials ParseInCredentials(XElement credentials, Authenticati
    {
        ClientCredentials result = new ClientCredentials();
        if (credentials.HasElements)
        {
            Credential cred = CredentialManager.ReadCredentials(target);
            switch (endpointType)
            {
                case AuthenticationProviderType.ActiveDirectory:
                    if (null != cred && cred.UserName.Contains("\\"))
                    {
                        String[] domainAndUser = cred.UserName.Split('\\');
                        result.Windows.ClientCredential = new System.Net.NetworkCre
                                                          {
                                                              UserName = domainAndUser[1],
                                                              Domain = domainAndUser[0],
                                                              Password = cred.Password
                                                          };
                    }
                    else
                    {
                        result.Windows.ClientCredential = new System.Net.NetworkCre
                        {
                            UserName = credentials.Element("UserName").Value,
                            Domain = credentials.Element("Domain").Value
                        };
                    }
                    break;
                case AuthenticationProviderType.LiveId:
                case AuthenticationProviderType.Federation:
                case AuthenticationProviderType.OnlineFederation:
                    if (null != cred)
                    {
                        result.UserName.UserName = cred.UserName;
                        result.UserName.Password = cred.Password;
                    }
                    else
                    {
                        result.UserName.UserName = credentials.Element("UserName").'
                    }
                    break;
                default:
                    break;
            }
        }
        else
            return null;

        return result;
    }

    /// <summary>
    /// Parse ClientCredentials into XML node.
    /// </summary>
    /// <param name="clientCredentials">ClientCredentials type.</param>
    /// <param name="endpointType">AuthenticationProviderType of the credentials.</
    /// <param name="target">Target is the key with which associated credentials ca
    /// <returns>XML node containing credentials data.</returns>
    private XElement ParseOutCredentials(ClientCredentials clientCredentials,
        AuthenticationProviderType endpointType, String target)
    {
        if (clientCredentials != null)
        {
            Credential cred = CredentialManager.ReadCredentials(target);
```

```
            switch (endpointType)
            {
                case AuthenticationProviderType.ActiveDirectory:
                    if (cred == null)
                    {
                        // Add entry in windows credential manager for future use.
                        if (!String.IsNullOrWhiteSpace(clientCredentials.Windows.Cl
                        {
                            CredentialManager.WriteCredentials(target,
                                new Credential(clientCredentials.Windows.ClientCred
                                    + clientCredentials.Windows.ClientCredential.Us
                                    clientCredentials.Windows.ClientCredential.Pass
                                true);
                        }
                    }
                    else
                    {
                        // Replace if the password has been changed.
                        if (!clientCredentials.Windows.ClientCredential.Password.Eq
                        {
                            CredentialManager.DeleteCredentials(target, false);
                            CredentialManager.WriteCredentials(target,
                                new Credential(clientCredentials.Windows.ClientCred
                                    + clientCredentials.Windows.ClientCredential.Us
                                    clientCredentials.Windows.ClientCredential.Pass
                                true);
                        }
                    }
                    return new XElement("Credentials",
                        new XElement("UserName", clientCredentials.Windows.ClientCr
                        new XElement("Domain", clientCredentials.Windows.ClientCred
                        );
                case AuthenticationProviderType.LiveId:
                case AuthenticationProviderType.Federation:
                case AuthenticationProviderType.OnlineFederation:
                    if (cred == null)
                    {
                        // Add entry in windows credential manager for future use.
                        if (!String.IsNullOrWhiteSpace(clientCredentials.UserName.P
                        {
                            CredentialManager.WriteCredentials(target,
                                new Credential(clientCredentials.UserName.UserName,
                                    clientCredentials.UserName.Password),
                                true);
                        }
                    }
                    else
                    {
                        // Replace if the password has been changed.
                        if (!clientCredentials.UserName.Password.Equals(cred.Passwo
                        {
                            CredentialManager.DeleteCredentials(target, false);
                            CredentialManager.WriteCredentials(target,
                               new Credential(clientCredentials.UserName.UserName,
                                   clientCredentials.UserName.Password),
                               true);
                        }
                    }
                    return new XElement("Credentials",
                       new XElement("UserName", clientCredentials.UserName.UserName
                       );
                default:
                    break;
            }
        }

        return new XElement("Credentials", "");
    }
    #endregion Private methods

    #region Private Classes
    /// <summary>
    /// private static class to store constants required by the CrmServiceHelper cl
    /// </summary>
    private static class CrmServiceHelperConstants
    {
        /// <summary>
        /// Credentials file path.
        /// </summary>
        public static readonly string ServerCredentialsFile = Path.Combine(
```

```
                Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Applic
                "Credentials.xml");
        }
        #endregion
    }

    #region Other Classes
    internal sealed class Credential
    {
        private SecureString _userName;
        private SecureString _password;

        internal Credential(CREDENTIAL_STRUCT cred)
        {
            _userName = ConvertToSecureString(cred.userName);
            int size = (int)cred.credentialBlobSize;
            if (size != 0)
            {
                byte[] bpassword = new byte[size];
                Marshal.Copy(cred.credentialBlob, bpassword, 0, size);
                _password = ConvertToSecureString(Encoding.Unicode.GetString(bpassword)
            }
            else
            {
                _password = ConvertToSecureString(String.Empty);
            }
        }

        public Credential(string userName, string password)
        {
            if (String.IsNullOrWhiteSpace(userName))
                throw new ArgumentNullException("userName");
            if (String.IsNullOrWhiteSpace(password))
                throw new ArgumentNullException("password");

            _userName = ConvertToSecureString(userName);
            _password = ConvertToSecureString(password);
        }

        public string UserName
        {
            get { return ConvertToUnsecureString(_userName); }
        }

        public string Password
        {
            get { return ConvertToUnsecureString(_password); }
        }

        /// <summary>
        /// This converts a SecureString password to plain text
        /// </summary>
        /// <param name="securePassword">SecureString password</param>
        /// <returns>plain text password</returns>
        private string ConvertToUnsecureString(SecureString secret)
        {
            if (secret == null)
                return string.Empty;

            IntPtr unmanagedString = IntPtr.Zero;
            try
            {
                unmanagedString = Marshal.SecureStringToGlobalAllocUnicode(secret);
                return Marshal.PtrToStringUni(unmanagedString);
            }
            finally
            {
                Marshal.ZeroFreeGlobalAllocUnicode(unmanagedString);
            }
        }

        /// <summary>
        /// This converts a string to SecureString
        /// </summary>
        /// <param name="password">plain text password</param>
        /// <returns>SecureString password</returns>
        private SecureString ConvertToSecureString(string secret)
        {
            if (string.IsNullOrEmpty(secret))
                return null;
```

```
            SecureString securePassword = new SecureString();
            char[] passwordChars = secret.ToCharArray();
            foreach (char pwdChar in passwordChars)
            {
                securePassword.AppendChar(pwdChar);
            }
            securePassword.MakeReadOnly();
            return securePassword;
        }


        /// <summary>
        /// This structure maps to the CREDENTIAL structure used by native code. We can
        /// </summary>
        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
        internal struct CREDENTIAL_STRUCT
        {
            public UInt32 flags;
            public UInt32 type;
            public string targetName;
            public string comment;
            public System.Runtime.InteropServices.ComTypes.FILETIME lastWritten;
            public UInt32 credentialBlobSize;
            public IntPtr credentialBlob;
            public UInt32 persist;
            public UInt32 attributeCount;
            public IntPtr credAttribute;
            public string targetAlias;
            public string userName;
        }

    }

    /// <summary>
    /// This class exposes methods to read, write and delete user credentials
    /// </summary>
    internal static class CredentialManager
    {
        /// <summary>
        /// Target Name against which all credentials are stored on the disk.
        /// </summary>
        public const string TargetName = "Microsoft_CRMSDK:";

        /// <summary>
        /// Cache containing secrets in-memory (used to improve performance and avoid I
        /// </summary>
        private static Dictionary<string, Credential> credentialCache = new Dictionary<

        public static Uri GetCredentialTarget(Uri target)
        {
            if (null == target)
                throw new ArgumentNullException("target");
            return new Uri(target.GetLeftPart(UriPartial.Authority));
        }

        private enum CRED_TYPE : int
        {
            GENERIC = 1,
            DOMAIN_PASSWORD = 2,
            DOMAIN_CERTIFICATE = 3,
            DOMAIN_VISIBLE_PASSWORD = 4,
            MAXIMUM = 5
        }

        internal enum CRED_PERSIST : uint
        {
            SESSION = 1,
            LOCAL_MACHINE = 2,
            ENTERPRISE = 3
        }

        private static class NativeMethods
        {
            [DllImport("advapi32.dll", SetLastError = true,
                EntryPoint = "CredReadW", CharSet = CharSet.Unicode)]
            [return: MarshalAs(UnmanagedType.Bool)]
            public static extern bool CredRead(string target, CRED_TYPE type, int reser
                [MarshalAs(UnmanagedType.CustomMarshaler,
                    MarshalTypeRef = typeof(CredentialMarshaler))] out Credential crede
```

```
        [DllImport("Advapi32.dll", SetLastError = true,
            EntryPoint = "CredWriteW", CharSet = CharSet.Unicode)]
        [return: MarshalAs(UnmanagedType.Bool)]
        public static extern bool CredWrite(ref Credential.CREDENTIAL_STRUCT creden

        [DllImport("Advapi32.dll", EntryPoint = "CredFree", SetLastError = true)]
        [return: MarshalAs(UnmanagedType.Bool)]
        public static extern bool CredFree(IntPtr cred);

        [DllImport("advapi32.dll", EntryPoint = "CredDeleteW", CharSet = CharSet.Un
        [return: MarshalAs(UnmanagedType.Bool)]
        public static extern bool CredDelete(string target, int type, int flags);
    }

    private sealed class CredentialMarshaler : ICustomMarshaler
    {
        private static CredentialMarshaler _instance;

        public void CleanUpManagedData(object ManagedObj)
        {
            // Nothing to do since all data can be garbage collected.
        }

        public void CleanUpNativeData(IntPtr pNativeData)
        {
            if (pNativeData == IntPtr.Zero)
            {
                return;
            }
            NativeMethods.CredFree(pNativeData);
        }

        public int GetNativeDataSize()
        {
            throw new NotImplementedException("The method or operation is not imple
        }

        public IntPtr MarshalManagedToNative(object obj)
        {
            throw new NotImplementedException("Not implemented yet");
        }

        public object MarshalNativeToManaged(IntPtr pNativeData)
        {
            if (pNativeData == IntPtr.Zero)
            {
                return null;
            }
            return new Credential((Credential.CREDENTIAL_STRUCT)Marshal.PtrToStruct
        }


        public static ICustomMarshaler GetInstance(string cookie)
        {
            if (null == _instance)
                _instance = new CredentialMarshaler();
            return _instance;
        }
    }

    public static Credential ReadCredentials(String target)
    {
        Credential cachedCredential;

        // Try to read the username from cache
        if (credentialCache.TryGetValue(TargetName + target, out cachedCredential))
        {
            return cachedCredential;
        }

        Credential credential;
        bool bSuccess = NativeMethods.CredRead(TargetName + target, CRED_TYPE.GENER
        // No match found.
        if (!bSuccess)
        {
            return null;
        }

        credentialCache[TargetName + target.ToString()] = credential;
```

```
        return credential;
    }

    public static Credential ReadWindowsCredential(Uri target)
    {
        Credential credential;
        bool bSuccess = NativeMethods.CredRead(target.Host, CRED_TYPE.DOMAIN_PASSWO
        if (!bSuccess)
        {
            throw new InvalidOperationException("Unable to read windows credentials
                new System.ComponentModel.Win32Exception(Marshal.GetLastWin32Error(
        }
        return credential;
    }
    /// <summary>
    /// Fetches the credentials.
    /// </summary>
    /// <param name="target">Target is the key with which associated credentials ca
    /// <param name="userCredentials">It is the in parameter which contains the use
    /// <param name="allowPhysicalStore">If allowPhysicalStore is true then the cre
    public static void WriteCredentials(String target, Credential userCredentials,
    {
        if (String.IsNullOrWhiteSpace(target))
            throw new ArgumentNullException("target");
        if (null == userCredentials)
            throw new ArgumentNullException("userCredentials");
        // Cache the username and password in memory
        credentialCache[TargetName + target] = userCredentials;

        // Store the credentials if allowed
        string passwordToStore = allowPhysicalStore ? userCredentials.Password : st
        Credential.CREDENTIAL_STRUCT credential = new Credential.CREDENTIAL_STRUCT(
        try
        {
            credential.targetName = TargetName + target;
            credential.type = (UInt32)CRED_TYPE.GENERIC;
            credential.userName = userCredentials.UserName;
            credential.attributeCount = 0;
            credential.persist = (UInt32)CRED_PERSIST.LOCAL_MACHINE;
            byte[] bpassword = Encoding.Unicode.GetBytes(passwordToStore);
            credential.credentialBlobSize = (UInt32)bpassword.Length;
            credential.credentialBlob = Marshal.AllocCoTaskMem(bpassword.Length);
            Marshal.Copy(bpassword, 0, credential.credentialBlob, bpassword.Length)
            if (!NativeMethods.CredWrite(ref credential, 0))
            {
                throw new System.ComponentModel.Win32Exception(Marshal.GetLastWin32
            }
        }
        finally
        {
            if (IntPtr.Zero != credential.credentialBlob)
                Marshal.FreeCoTaskMem(credential.credentialBlob);
        }
    }

    /// <summary>
    /// Deletes the credentials.
    /// </summary>
    /// <param name="target">Target is the key with which associated credentials ca
    /// <param name="softDelete">If a softDelete is done then credentials are delet
    /// They are completely removed otherwise.</param>
    public static void DeleteCredentials(String target, bool softDelete)
    {
        if (String.IsNullOrWhiteSpace(target))
            throw new ArgumentNullException("target");
        if (softDelete)
        {
            // Removes only the password
            try
            {
                Credential tempCredential = ReadCredentials(target);
                WriteCredentials(target, new Credential(tempCredential.UserName, St
            }
            catch (Exception)
            {
                // Do nothing
            }
        }
        else
        {
```

```
                // Removes the entry completely
                NativeMethods.CredDelete(TargetName + target, (int)CRED_TYPE.GENERIC, 0
                credentialCache.Remove(TargetName + target);
            }
        }
    }

    /// <summary>
    /// Wrapper class for DiscoveryServiceProxy to support auto refresh security token.
    /// </summary>
    internal sealed class ManagedTokenDiscoveryServiceProxy : DiscoveryServiceProxy
    {
        private AutoRefreshSecurityToken<DiscoveryServiceProxy, IDiscoveryService> _pro

        public ManagedTokenDiscoveryServiceProxy(Uri serviceUri, ClientCredentials user
            : base(serviceUri, null, userCredentials, null)
        {
            this._proxyManager = new AutoRefreshSecurityToken<DiscoveryServiceProxy, ID
        }

        public ManagedTokenDiscoveryServiceProxy(IServiceManagement<IDiscoveryService>
            SecurityTokenResponse securityTokenRes)
            : base(serviceManagement, securityTokenRes)
        {
            this._proxyManager = new AutoRefreshSecurityToken<DiscoveryServiceProxy, ID
        }

        public ManagedTokenDiscoveryServiceProxy(IServiceManagement<IDiscoveryService>
           ClientCredentials userCredentials)
            : base(serviceManagement, userCredentials)
        {
            this._proxyManager = new AutoRefreshSecurityToken<DiscoveryServiceProxy, ID
        }

        protected override SecurityTokenResponse AuthenticateDeviceCore()
        {
            return this._proxyManager.AuthenticateDevice();
        }

        protected override void AuthenticateCore()
        {
            this._proxyManager.PrepareCredentials();
            base.AuthenticateCore();
        }

        protected override void ValidateAuthentication()
        {
            this._proxyManager.RenewTokenIfRequired();
            base.ValidateAuthentication();
        }
    }

    /// <summary>
    /// Wrapper class for OrganizationServiceProxy to support auto refresh security tok
    /// </summary>
    internal sealed class ManagedTokenOrganizationServiceProxy : OrganizationServicePro
    {
        private AutoRefreshSecurityToken<OrganizationServiceProxy, IOrganizationService

        public ManagedTokenOrganizationServiceProxy(Uri serviceUri, ClientCredentials u
            : base(serviceUri, null, userCredentials, null)
        {
            this._proxyManager = new AutoRefreshSecurityToken<OrganizationServiceProxy,
        }

        public ManagedTokenOrganizationServiceProxy(IServiceManagement<IOrganizationSer
            SecurityTokenResponse securityTokenRes)
            : base(serviceManagement, securityTokenRes)
        {
            this._proxyManager = new AutoRefreshSecurityToken<OrganizationServiceProxy,
        }

        public ManagedTokenOrganizationServiceProxy(IServiceManagement<IOrganizationSer
            ClientCredentials userCredentials)
            : base(serviceManagement, userCredentials)
        {
            this._proxyManager = new AutoRefreshSecurityToken<OrganizationServiceProxy,
        }

        protected override SecurityTokenResponse AuthenticateDeviceCore()
```

```
        {
            return this._proxyManager.AuthenticateDevice();
        }

        protected override void AuthenticateCore()
        {
            this._proxyManager.PrepareCredentials();
            base.AuthenticateCore();
        }

        protected override void ValidateAuthentication()
        {
            this._proxyManager.RenewTokenIfRequired();
            base.ValidateAuthentication();
        }
    }

    /// <summary>
    /// Class that wraps acquiring the security token for a service
    /// </summary>
    public sealed class AutoRefreshSecurityToken<TProxy, TService>
        where TProxy : ServiceProxy<TService>
        where TService : class
    {
        private ClientCredentials _deviceCredentials;
        private TProxy _proxy;

        /// <summary>
        /// Instantiates an instance of the proxy class
        /// </summary>
        /// <param name="proxy">Proxy that will be used to authenticate the user</param
        public AutoRefreshSecurityToken(TProxy proxy)
        {
            if (null == proxy)
            {
                throw new ArgumentNullException("proxy");
            }

            this._proxy = proxy;
        }

        /// <summary>
        /// Prepares authentication before authen6ticated
        /// </summary>
        public void PrepareCredentials()
        {
            if (null == this._proxy.ClientCredentials)
            {
                return;
            }

            switch (this._proxy.ServiceConfiguration.AuthenticationType)
            {
                case AuthenticationProviderType.ActiveDirectory:
                    this._proxy.ClientCredentials.UserName.UserName = null;
                    this._proxy.ClientCredentials.UserName.Password = null;
                    break;
                case AuthenticationProviderType.Federation:
                case AuthenticationProviderType.LiveId:
                    this._proxy.ClientCredentials.Windows.ClientCredential = null;
                    break;
                default:
                    return;
            }
        }

        /// <summary>
        /// Authenticates the device token
        /// </summary>
        /// <returns>Generated SecurityTokenResponse for the device</returns>
        public SecurityTokenResponse AuthenticateDevice()
        {
            if (null == this._deviceCredentials)
            {
                this._deviceCredentials = DeviceIdManager.LoadOrRegisterDevice(
                    this._proxy.ServiceConfiguration.CurrentIssuer.IssuerAddress.Uri);
            }

            return this._proxy.ServiceConfiguration.AuthenticateDevice(this._deviceCred
        }
```

```
        /// <summary>
        /// Renews the token (if it is near expiration or has expired)
        /// </summary>
        public void RenewTokenIfRequired()
        {
            if (null != this._proxy.SecurityTokenResponse &&
                DateTime.UtcNow.AddMinutes(15) >= this._proxy.SecurityTokenResponse.Res
            {
                try
                {
                    this._proxy.Authenticate();
                }
                catch (CommunicationException)
                {
                    if (null == this._proxy.SecurityTokenResponse ||
                        DateTime.UtcNow >= this._proxy.SecurityTokenResponse.Response.L
                    {
                        throw;
                    }

                    // Ignore the exception
                }
            }
        }
    }
    #endregion
}
```

See Also

Tasks

Sample: Quick Start for Microsoft Dynamics CRM

Concepts

Use the Sample and Helper Code

Other Resources

Authenticate Users with Microsoft Dynamics CRM Web Services

Microsoft Dynamics CRM 2011

Send comments about this topic to Microsoft.

Did you find this helpful? ◯ Yes ◯ No

Community Additions ADD