



QDABRA ACTIVE DIRECTORY WEB SERVICE

Using Qdabra's Active Directory Web Service on an ASP.NET website

Qdabra's Active Directory Web Service (ADWS) works well with web service clients like Microsoft's InfoPath form editor. Qdabra markets its web services for InfoPath users, but the web services aren't limited to InfoPath. Any web 2.0 client can call Qdabra's web services. You can leverage ADWS and other web services from your website, Windows applications, or even command line utilities.

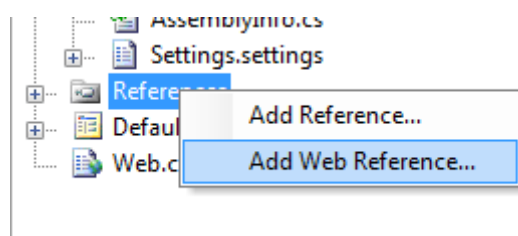
This document shows how to use ADWS from a website and makes the two following assumptions:

- you are designing your website in Visual Studio using an ASP.NET Web Application project,
- your website uses Windows authentication, and
- your users are authenticated on the pages where this code will be used.

This tutorial's code samples use the C# programming language, so if your site uses VB.NET or another language, you will need to produce the equivalent code.

GETTING STARTED

To use ADWS in your .NET application, you will have to add a reference to the web service. In the Project Explorer of your website project, right click the **References** icon and click **Add Web Reference...**



In the dialog box, type the address of the **AdUserInfo.asmx** page for your installation of ADWS and click **Go**. A list of the available web methods should appear.



URL: Go

AdUserInfo

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [FindUsersByAlias](#)
Attempts to find any users with an alias matching the specified search string.
- [FindUsersByName](#)
Attempts to find any users with a name matching the specified search string.
- [GetEmployeesForManager](#)
Returns a list of employees that have a manager with the specified alias.

Web services found at this URL:

1 Service Found:

- AdUserInfo

Web reference name:

Add Reference

If the **Web reference name** textbox populates with your server's hostname, you may want to give it a more descriptive name, such as "ADWSWebReference" and click **Add Reference**.

Finally, add a new ASPX web page to your site so that you can use it to experiment with ADWS. We're now ready to start working on the web page.

SOME PAGE SETUP AND CONVENIENCE METHODS

Your ASPX page should have a simple HTML structure, containing a `<form>` and `<div>` tag inside the page's body.

```
<body>
  <form id="form1" runat="server">
    <div>

    </div>
  </form>
</body>
```

Modify the `<div>` tag to have the **runat="server"** attribute and give it the ID "userInfoDiv" so that we can access this control from code:

```
<form id="form1" runat="server">
  <div runat="server" id="userInfoDiv">

  </div>
</form>
```

That's it. Those are the only changes you need to make to the page's HTML.

Next, let's add a few common helper methods that we will use throughout this tutorial. Switch to the code view of your page and add the following below the `Page_Load()` method:



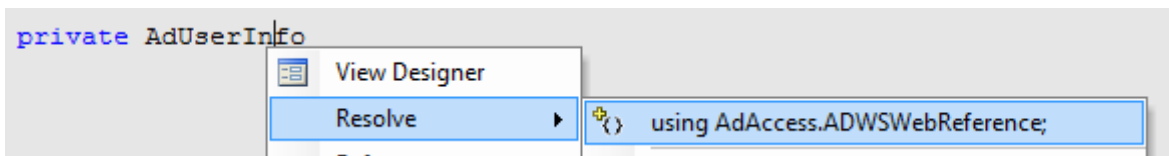
```
private void PrintLine(string text)
{
    HtmlGenericControl p = new HtmlGenericControl("p");
    p.InnerText = text;
    userInfoDiv.Controls.Add(p);
}
```

The PrintLine method will add a line of text (in the form of a <p> tag) to the userInfoDiv control. We'll use this method any time we want to display some text on the page.

```
private AdUserInfo GetADWSReference()
{
    AdUserInfo webReference = new AdUserInfo();
    webReference.Url = "http://          /QdabraWebService/AdUserInfo.asmx";
    webReference.UseDefaultCredentials = true;

    return webReference;
}
```

Visual Studio's Intellisense doesn't recognize the AdUserInfo class name, so after you type the words "private AdUserInfo", please right-click "AdUserInfo," select **Resolve**, and select the top option in the menu list (it may differ depending on the name of your website)



The GetADWSReference method returns a reference to your ADWS web service, which you can use to simplify calling any of its web methods from your code. We will use this method in a few places in this demo. The value you assign to **Url** should be the location of your ADWS web service. It's not strictly necessary that you set **Url** here if you will be using the service in the same location where you initially accessed it, but it's useful to have this line here in case you need to change the address at some point.

GetUserInfo()

The first ADWS web method that we will demo using is GetUserInfo(), which returns a list of Active Directory properties for a specific user. First let's create a helper method that will display the properties on the web page (*Note: It's unlikely that you would want to simply dump a list of a user's Active Directory properties on a web page like this on an actual website, so this is primarily for demonstrational purposes. In actual practice it's more likely that you would want to iterate through the array, find the property that you seek, and use this as necessary. See the GetMyInfo() section for an example of this.*).



```
private void ShowADProperties(ADProp[] properties)
{
    if (properties == null)
    {
        PrintLine("No properties to display.");
    }
    else
    {
        foreach (ADProp property in properties)
        {
            PrintLine(string.Format("{0}: {1}", property.Key, property.Value));
        }
    }
}
```

The ShowADProperties method iterates through an array of ADProp objects (GetUserInfo() returns an array of ADProp objects), and displays each pair of property names and property values in the web page. It also checks whether the array is null and displays an appropriate message in that situation, as GetUserInfo() will return a null array if the username passed to it is invalid. Now we need some code to actually call the web method and then pass the values to ShowADProperties:

```
private void CallGetUserInfo(string userName)
{
    AdUserInfo webReference = GetADWSReference();
    ADProp[] properties = webReference.GetUserInfo(userName);

    ShowADProperties(properties);
}
```

As you can see here, CallGetUserInfo uses the GetADWSReference() helper method we wrote earlier to obtain a reference to the webservice, and then calls GetUserInfo(). It passes the result to ShowADProperties() to display the result in the web page. The last piece of the puzzle is to add some code to call *this* CallGetUserInfo() method, and pass it a *userName* value to use. The easiest way to make the code run is to add it to the page's existing Page_Load() method, so let's just do that, and add it there. Before adding the new method, please make sure to add a **using** statement list of **using** statements at the top of your file, to reference the System.Security.Principal namespace:

```
using System.Security.Principal;
```



```
protected void Page_Load(object sender, EventArgs e)
{
    try
    {
        IIdentity userIdentity = HttpContext.Current.User.Identity;
        if (userIdentity == null)
        {
            PrintLine("No user is logged in!");
        }
        else
        {
            CallGetUserInfo(userIdentity.Name);
        }
    }
    catch (Exception ex)
    {
        PrintLine("An error occurred: " + ex.Message);
    }
}
```

The Page_Load method attempts to retrieve an IIdentity object for the current user and if none is available, displays an appropriate message. It also employs a try...catch block to handle any exceptions that may occur. If everything goes well (no exceptions), this method passes the **Name** property of the userIdentity object to CallGetUserInfo(). Viewing the page at this point should reveal a list of the current user's active directory properties:

objectClass: top
cn: James Rishe
sn: Rishe
c: US
l: Kirkland
st: WA
title: Software Engineer
postalCode: 98033
givenName: James

GetMyInfo() – Impersonating the user

The GetMyInfo() web method assumes that the end user is calling the web service directly. This is a simple matter when the user is filling out an InfoPath form that calls these web methods (because InfoPath



impersonates the logged in user), but web pages don't automatically impersonate end users. In fact, most ASP.NET webpage code runs under the identity of a network service account, so when the code calls one of ADWS' web methods, the web method will actually think that the network service account is calling the web method and not the webpage user.

We can demonstrate this by working up a new method similar to CallGetUserInfo(), that calls GetMyInfo instead. Here is the code:

```
private void CallGetMyInfo()  
{  
    AdUserInfo webReference = GetADWSReference();  
    PrintLine("Calling GetMyInfo() as " + Environment.UserName);  
    ADProp[] properties = webReference.GetMyInfo();  
  
    ShowADProperties(properties);  
}
```

After modifying Page_Load to call this method and re-opening the page, we encounter the situation that I described above:

Calling GetMyInfo() as NETWORK SERVICE

No properties to display.

The page was unable to retrieve any user properties, because ADWS does not return properties for non-human accounts like the network service account.

THE SOLUTION

To get around this you need to have the ASP.NET webpage temporarily impersonate the logged-on user before calling the web method. If it does this, the web method will think that the user is calling the web method directly, and will return the correct data. Here is a method that does this correctly:



```
private ADProp[] GetMyProperties(IIdentity userIdentity)
{
    // Try to cast userIdentity to a WindowsIdentity
    WindowsIdentity winIdentity = userIdentity as WindowsIdentity;
    if (winIdentity == null)
    {
        PrintLine("Unable to obtain a windows identity for the current user.");
        return null;
    }
    else
    {
        WindowsImpersonationContext impersonation = null;
        try
        {
            AdUserInfo webReference = GetADWSReference();
            PrintLine("Current user before impersonating: " + Environment.UserName);
            impersonation = winIdentity.Impersonate();
            PrintLine("Now calling GetMyInfo as: " + Environment.UserName);
            return webReference.GetMyInfo();
        }
        finally
        {
            if (impersonation != null)
            {
                impersonation.Undo();
            }
        }
    }
}
```

The GetMyProperties method is more code but not at all complicated.. It first tries to convert the userIdentity object to a WindowsIdentity object and displays a message if this fails. It then impersonates the user, interspaced with some status messages, and calls the web method.

It's important to call the impersonation.Undo() method once we've called the web method, to stop impersonating the user when you're done. A try...finally block is added here to ensure we stop impersonating the user even if something else goes wrong in the process.

Here is the modified version of CallGetMyInfo(). Note that it now accepts an IIdentity parameter:

```
private void CallGetMyInfo(IIdentity userIdentity)
{
    AdUserInfo webReference = GetADWSReference();
    ADProp[] properties = GetMyProperties(userIdentity);

    ShowADProperties(properties);
}
```



And here is the code to call this method from the Page_Load() method. Here we have replaced CallGetUserInfo() with CallGetMyInfo():

```
IIdentity userIdentity = HttpContext.Current.User.Identity;
if (userIdentity == null)
{
    PrintLine("No user is logged in!");
}
else
{
    CallGetMyInfo(userIdentity);
}
```

Once this is all in place, we get the expected result:

Current user before impersonating: NETWORK SERVICE

Now calling GetMyInfo as: jrishe

objectClass: top

cn: James Rishe

sn: Rishe

c: US

l: Kirkland

st: WA

title: Software Engineer

A SLIGHTLY MORE PRACTICAL EXAMPLE – GETTING A SINGLE ACTIVE DIRECTORY PROPERTY

Now we will build on the information we've seen so far to write a method that retrieves a single requested Active Directory property. Below is a simple method that iterates through an array of ADProp objects to find a specific one:

This method will return null if the array is null, or if the specified property is not found.



```
private string GetPropertyValue(ADProp[] properties, string targetProperty)
{
    if (properties == null)
    {
        return null;
    }
    foreach (ADProp property in properties)
    {
        if (property.Key.Equals(targetProperty,
            StringComparison.InvariantCultureIgnoreCase))
        {
            return property.Value;
        }
    }

    return null;
}
```

And now here is a method that first retrieves the current user's AD properties by calling `GetMyProperties()`, then uses `GetPropertyValue()` to try to retrieve the *title* AD property, and display its value if it is found.

```
private void FindUserTitle(IIdentity userIdentity)
{
    ADProp[] properties = GetMyProperties(userIdentity);
    string title = GetPropertyValue(properties, "title");

    if (title == null)
    {
        PrintLine("Unable to retrieve the \"title\" property for the current user.");
    }
    else
    {
        PrintLine("Your job title is: " + title);
    }
}
```

Finally, we modify `Page_Load()` again to call this method:

```
IIdentity userIdentity = HttpContext.Current.User.Identity;
if (userIdentity == null)
{
    PrintLine("No user is logged in!");
}
else
{
    FindUserTitle(userIdentity);
}
```

Here is the final result:



Current user before impersonating: NETWORK SERVICE

Now calling GetMyInfo as: jrishe

Your job title is: Software Engineer