Printable Version

Using the Network Functions in C# (Part I - User Functions)

By Michael Bright

Welcome to another "How-to" article in Visual C#.NET. This article will focus on the Network functions within the Win32 API library associated with Network Management. The first thing is to point out is that there are two ways to manage users using the .NET Framework, the first being the Active Directory method, as you will notice you need to have Active Directory Installed and if you were managing users on a small network or on a stand alone station Active Directory won't be install and it would be not worth installing it. Therefore the other method and focus of this article is using the Platform Invoke layer and the Network functions of the Win32 API library. In this article we will look at how to Add, Delete and Modify Users and Groups using C# and also how we can query User and Network information for a machine or network, we will look at the following functions

- NetUserAdd
- NetUserDel
- NetUserGetInfo
- NetUserSetInfo
- NetUserChangePassword
- NetUserEnum
- NetUserGetLocalGroups

Getting Started

First of all as I'm sure most C# developers out there know we have to include the InteropServices namespace in our project to allow us to access functions from DLL's and this can be done using the following snippet of code:

```
///// CODE SNIPPET 1.0

using System.Runtime.InteropServices;

//// END OF CODE
```

Once we have included the access we can now include the declarations of the DLL's we are going to use and the Structures (structs) associated with them. Each of the calls needed will be discussed under the headings relating to what they do.

Adding a User using C#

One of the most important operations within the Network functions is that of

Adding Users to either a network or Computer. To Add a user via C# we will need to use the NetAddUser function, using this function allows us to specify a machine to add the user to or we can leave that null an the User is added to the local machine. In the below code snippet we can see how we can declare and use the NetUserAdd function. One important item associated with the NetUserAdd function is that the user you are added must be in the form of a specific structure. The structures (structs) that are used in association with this call are USER INFO 1, USER INFO 2, USER INFO 3 or USER INFO 4. In this practical example we shall use USER_INFO_1 to define our UserArea to add.

///// CODE SNIPPET 1.1 Declaration

```
[DllImport("Netapi32.dll")]
extern static int NetUserAdd([MarshalAs
(UnmanagedType.LPWStr)] string servername, int level, ref
USER_INFO_1 buf, int parm_err);
```

//// END OF CODE

You should also note that when i used this code, i didn't need to know about the errors returned, hence the use of an int as the last parameter, if you did want to know about errors returned you would need to amend this code. Now that we have declared our external API to use we should also include our USER_INFO_1 structure, using the following code snippet:

///// CODE SNIPPET 1.2 Structure Declaration

```
[StructLayout(LayoutKind.Sequential,
CharSet=CharSet.Unicode)]
public struct USER_INFO_1
{
public string usri1_name;
public string usri1_password;
public int usri1_password_age;
public int usri1_priv;
public string usri1_home_dir;
public string comment;
public int usri1_flags;
public string usri1_script_path;
}
```

//// END OF CODE

Once we have both of these block of code declare we can now use the call from within our program, i should also note that in this case we have used the structure about you can use any of the other three, and if you click the link you can convert the code from C++ to C# pretty easily. Below is a code snippet to show the usage of the NetUserAdd function.

///// CODE SNIPPET 1.3 NetUserAdd

```
USER_INFO_1 NewUser = new USER_INFO_1(); // Create an new
instance of the USER_INFO_1 struct

NewUser.usri1_name = "UserTestOne"; // Allocates the username
NewUser.usri1_password = "password"; // allocates the
password
```

```
NewUser.usri1_priv = 1; // Sets the account type to
USER_PRIV_USER
NewUser.usri1_home_dir = null; // We didn't supply a Home
Directory
NewUser.comment = "My First User Made through C#"; // Comment
on the User
NewUser.usri1_script_path = null; // We didn't supply a Logon
Script Path

if(NetUserAdd(null ,1 ,ref NewUser, 0)!=0) // If the call
fails we get a non-zero value
{
MessageBox.Show("Error Adding
User","Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}

//// END OF CODE
```

So as we discussed the above code will add a user to the machine you are currently on, but as I said if you want to add the user to a different machine on you network you can replace the first null parameter of the call with the name of the machine.

Removing a User using C#

Compared to the previous function removing a user is far easier, as with the above code the function returns a non-zero value if it fails, to remove a user from the local machine you can use the following declaration in the below code snippet.

```
///// CODE SNIPPET 1.4 Declaration

[DllImport("Netapi32.dll")]
extern static int NetUserDel([MarshalAs
(UnmanagedType.LPWStr)] string servername, [MarshalAs
(UnmanagedType.LPWStr)] string username);

//// END OF CODE
```

The usage for the NetUserDel called would be as shown below in the code snippet.

```
///// CODE SNIPPET 1.5 NetUserDel

if(NetUserDel(null ,"UserTestOne")!=0) // If the call fails
we get a non-zero value
{
MessageBox.Show("Error Removing
User","Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}

//// END OF CODE
```

As also with the NetUserAdd call, the user can make this call to remote machines by replacing the null string in the first parameter with that of the unicode representation or the computers name.

Getting User Information then modifying it using C#

To obtain user information within Visual C#.NET we need to used the native call NetUserGetInfo, this call as with NetUserAdd uses a structure (struct) to manage the data, however in this case it returns data to a struct as opposed to taking it out. In association with NetUserGetInfo, to change the information you have obtained you can use the function NetUserSetInfo. You should also note that these functions rely on each other to allow changes, for example to use the NetUserSetInfo function you must know the users privilege level, which is obtained via the NetUserGetInfo function. In the below code snippet we have the required declarations for both calls and please NOTE for this function we are using the USER_INFO_1 structure from above again.

```
///// CODE SNIPPET 1.6 Declarations

[DllImport("Netapi32.dll")]
extern static int NetUserGetInfo([MarshalAs
(UnmanagedType.LPWStr)] string servername,[MarshalAs
(UnmanagedType.LPWStr)] string username,int level,out IntPtr
bufptr);

[DllImport("Netapi32.dll")]
extern static int NetUserSetInfo([MarshalAs
(UnmanagedType.LPWStr)] string servername,[MarshalAs
(UnmanagedType.LPWStr)] string username,int level,ref
USER_INFO_1 buf, int error);

//// END OF CODE
```

Using these declarations we can obtain and modify a users settings, with ease, if we look at the next code snippet we will obtain the user information for the user we made earlier "UserTestOne" and then we will change some user information.

```
///// CODE SNIPPET 1.7 NetUserGetInfo

IntPtr bufPtr;
USER_INFO_1 User = new USER_INFO_1();
if(NetUserGetInfo(null, "Administrator",1,out bufPtr)!=0)
{
MessageBox.Show("Error Getting User
Info","Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}
User = (USER_INFO_1)Marshal.PtrToStructure(bufPtr, typeof
(USER_INFO_1));
MessageBox.Show("Users Name: " + User.usri1_name + " Users
Comments: " + User.comment + " Users Privilege Level: " +
User.usri1_priv);

// In this case we have used Marshaling to obtain the data
this is the only method i found that
// work but im sure someone can correct me on that?

//// END OF CODE

///// CODE SNIPPET 1.8 NetUsetSetInfo
```

```
USER_INFO_1 Update = new USER_INFO_1();
Update.comment = "This is Our C# Updated Comment";
Update.usri1_priv = 2; // Note that this can only be obtained
programmatically using NetUserGetInfo
if(NetUserSetInfo(null, "UserTestOne",1,ref Update,0)!=0)
{
MessageBox.Show("Error Setting User
Info","Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}
```

//// END OF CODE

To summarize, NetUserSetInfo relies on NetUserGetInfo for it to work correctly, and as with all the other Network functions, if you change the null string in the first parameter to that of a computer, then you can use the function remotely.

Changing a Users Password using C#

Another very important function of a network management program is to allow the changing or passwords. The function NetUserChangePassword, allows us to do this bascially providing we have the user specified current password. So this function could be considered to be used as part of a program that allows a user to change their own password. Anyway to declare the NetUserChangePassword function we use the following code snippet.

///// CODE SNIPPET 1.9 Declarations

```
[DllImport("Netapi32.dll")]
extern static int NetUserChangePassword([MarshalAs
(UnmanagedType.LPWStr)] string domainname,[MarshalAs
(UnmanagedType.LPWStr)] string username,[MarshalAs
(UnmanagedType.LPWStr)] string oldpassword,[MarshalAs
(UnmanagedType.LPWStr)] string newpassword);
```

//// END OF CODE

Using this declaration we can now manipulate a set users password, providing we know it already. Again we can use the below code remotely changing the null string parameter to that of the machine to manage.

///// CODE SNIPPET 2.0 NetUserChangePassword

```
if(NetUserChangePassword(null, "UserTestOne", "password",
"ournewpassword")!=0)
{
MessageBox.Show("Error Changing User
Password","Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
}
```

//// END OF CODE

Obtaining the list of Users using C#

When managing a network it is also important what we have a accurate list of the users on the network, or local machine. To obtain this we must use the NetUserEnum function, which returns the names and data elements of each user

area to structures. In this case we are going to use the USER_INFO_0 structure to pass the return values into, simply because we are only trying to get usernames no other data i have added this in the declarations section. It is also important to note that this function uses the network buffer, which must be freed after, to free resources, this is done using the NetAPIBufferFree function also listed. To start with in this code snippet is the declarations.

```
///// CODE SNIPPET 2.1 Declarations

[StructLayout(LayoutKind.Sequential,
CharSet=CharSet.Unicode)]
public struct USER_INFO_0
{
public String Username;
}

[DllImport("Netapi32.dll")]
extern static int NetUserEnum([MarshalAs
(UnmanagedType.LPWStr)] string servername, int level, int
filter, out IntPtr bufptr, int prefmaxlen, out int
entriesread, out int totalentries, out int resume_handle);

[DllImport("Netapi32.dll")]
extern static int NetApiBufferFree(IntPtr Buffer);

//// END OF CODE
```

Once we have made the declaration we can go out and use our code, to create the collection of the users, to do this we use the below code snippet.

```
///// CODE SNIPPET 2.2 NetUserEnum (Some code attributed by
Nick Holmes)

int EntriesRead;
int TotalEntries;
int Resume;
IntPtr bufPtr;

NetUserEnum(null, 0, 2, out bufPtr, -1, out EntriesRead, out
TotalEntries, out Resume);

if(EntriesRead> 0)
{
USER_INFO_0[] Users = new USER_INFO_0[EntriesRead];
IntPtr iter = bufPtr;
for(int i=0; i < EntriesRead; i++)
{
Users[i] = (USER_INFO_0)Marshal.PtrToStructure(iter, typeof
(USER_INFO_0));
iter = (IntPtr)((int)iter + Marshal.SizeOf(typeof
(USER_INFO_0)));
MessageBox.Show(Users[i].Username);
}
NetworkAPI.NetApiBufferFree(bufPtr);
}

//// END OF CODE
```

The above code intern returns each use listed on the local machine via MessageBox, as with all other function you can chose a remote computer again by replacing the null string.

Identifying a users group membership using C#

The one last function we need to look at in this article is NetUserGetLocalGroups. This function allows us to determine what Groups the user is a member of and display them. As with the previous method we have to flush the NetAPIBuffer after using the function, using the same declaration as earlier. The declaration for NetUserGetLocalGroups also needs the LOCALGROUP_USERS_INFO_0 structure to return the names to this along with the declaration are shown in the below code snippet.

```
///// CODE SNIPPET 2.3 Declarations

[StructLayout(LayoutKind.Sequential,
CharSet=CharSet.Unicode)]
public struct LOCALGROUP_USERS_INFO_0
{
public string groupname;
}

[DllImport("Netapi32.dll")]
public extern static int NetUserGetLocalGroups([MarshalAs
(UnmanagedType.LPWStr)] string servername,[MarshalAs
(UnmanagedType.LPWStr)] string username, int level, int
flags, out IntPtr bufptr, int prefmaxlen, out int
entriesread, out int totalentries);

//// END OF CODE
```

With the above declaration we can now call it using similar code as in the previous functions, looking something like the below code snippet.

```
///// CODE SNIPPET 2.4 NetUserGetLocalGroups

int EntriesRead;
int TotalEntries;
IntPtr bufPtr;

NetUserGetLocalGroups(null, "Administrator",0,0,out
bufPtr,1024,out EntriesRead, out TotalEntries);

if(EntriesRead> 0)
{
LOCALGROUP_USERS_INFO_0[] RetGroups = new
LOCALGROUP_USERS_INFO_0[EntriesRead];
IntPtr iter = bufPtr;
for(int i=0; i < EntriesRead; i++)
{
RetGroups[i] = (LOCALGROUP_USERS_INFO_0)
Marshal.PtrToStructure(iter, typeof
(LOCALGROUP_USERS_INFO_0));
iter = (IntPtr)((int)iter + Marshal.SizeOf(typeof
(LOCALGROUP_USERS_INFO_0)));
MessageBox.Show(RetGroups[i].groupname);
```

```
}
NetApiBufferFree(bufPtr);
}

//// END OF CODE
```

The above example returns the groups to which the user Administrator belongs, again you can specify any user and any computer name. In this article we has looked at and discussed the User related Network Function available through Platform Invoke on the .NET Framework. So you can see that all this code does in fact work, i have included a sample Example application with all the function working in it. Available for download below.

Network Function Examples ~
Visual Studio.NET Project + Source (49K)
Runtime EXE (28K)

About the Author

My name is Michael Bright, I'm a university student studying a Bsc in Computer Science at University College Chester. I have a love for all thing computers and have only worked with C# for about 3 months. I am currently training for my MCP in it. My other interests are C, VB, HTML, ASP and also Flash. I have interests in Developing Network app's and Network Security. My Web site is csharp.brightweb.co.uk where you can find examples of my Work, including my Defender Security applications.

Saturday 25 January, 2003 2:52 PM