Utilizzando le api della libreria di sistema Wtsapi32.dll è possibile elencare e gestire le sessioni aperte in un server Windows sia locale che remoto.

Il prerequisito è ovviamente che l'utente abbia garantito l'accesso a quel server, in caso contrario sarebbe necessario avviare le api impersonando un utente che abbia

le autorizzazioni necessarie ad accedere al server remoto.

Le funzioni principali (e come si integrano un C#) sono le seguenti:

```
[DllImport(&quot;Wtsapi32.dll&quot;)]
static extern bool WTSQuerySessionInformation(
System.IntPtr hServer, int sessionId, WTS_INFO_CLASS wtsInfoClass, out System.IntPtr ppBuffer, out uint pBytesReturned);

[DllImport(&quot;wtsapi32.dll&quot;, SetLastError=true)]
static extern IntPtr WTSOpenServer([MarshalAs(UnmanagedType.LPStr)] String pServerName);

[DllImport(&quot;wtsapi32.dll&quot;)]
static extern void WTSCloseServer(IntPtr hServer);

[DllImport(&quot;wtsapi32.dll&quot;, SetLastError = true)]
static extern bool WTSLogoffSession(IntPtr hServer, int SessionId, bool bWait);

[DllImport(&quot;wtsapi32.dll&quot;, SetLastError = true)]
static extern bool WTSDisconnectSession(IntPtr hServer, int SessionId, bool bWait);

[DllImport(&quot;wtsapi32.dll&quot;, SetLastError = true)]
static extern bool WTSTerminateProcess(IntPtr hServer, int ProcessId, bool bWait);
```

WTSQuerySessionInformation: recupera i dati delle sessioni presenti in quel momento sul server

WTSOpenServer: apre la connessione al server target

WTSCloseServer: chiude la connessione al server al termine dell'operazione di enumerazione delle sessioni

WTSLogoffSession: esegue il logoff della sessione specificata tramite SESSIONID

WTSDisconnectSession: esegue la disconnessione della sessione specificata

WTSTerminateProcess: termina un processo per quella sessione

Un esempio è mostrato nel listato seguente:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime.InteropServices;

namespace TerminalServices
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            List<String> ses = ListSessions(&amp;quot;ACERFAC&amp;quot;);
            foreach (String s in ses)
                // if(s.Contains(&amp;quot;ACERFAC&amp;quot;))
                Console.WriteLine(s);



            Console.ReadKey();

        }
```

```
    [DllImport(&amp;quot;Wtsapi32.dll&amp;quot;)]
    static extern bool WTSQuerySessionInformation(
    System.IntPtr hServer, int sessionId, WTS_INFO_CLASS wtsInfoClass, out System.IntPtr
ppBuffer, out uint pBytesReturned);


    [DllImport(&amp;quot;wtsapi32.dll&amp;quot;, SetLastError=true)]
    static extern IntPtr WTSOpenServer([MarshalAs(UnmanagedType.LPStr)] String
pServerName);

    [DllImport(&amp;quot;wtsapi32.dll&amp;quot;)]
    static extern void WTSCloseServer(IntPtr hServer);

    [DllImport(&amp;quot;wtsapi32.dll&amp;quot;, SetLastError = true)]
    static extern bool WTSLogoffSession(IntPtr hServer, int SessionId, bool bWait);

    [DllImport(&amp;quot;wtsapi32.dll&amp;quot;, SetLastError = true)]
    static extern bool WTSDisconnectSession(IntPtr hServer, int SessionId, bool bWait);

    [DllImport(&amp;quot;wtsapi32.dll&amp;quot;, SetLastError = true)]
    static extern bool WTSTerminateProcess(IntPtr hServer, int ProcessId, bool bWait);

    [DllImport(&amp;quot;kernel32.dll&amp;quot;, SetLastError = true)]
    static extern bool GetLastError();
```

```
[DllImport(&quot;wtsapi32.dll&quot;, SetLastError=true)]
static extern Int32 WTSEnumerateSessions(
    IntPtr hServer,
    [MarshalAs(UnmanagedType.U4)] Int32 Reserved,
    [MarshalAs(UnmanagedType.U4)] Int32 Version,
    ref IntPtr ppSessionInfo,
    [MarshalAs(UnmanagedType.U4)] ref Int32 pCount);

[DllImport(&quot;wtsapi32.dll&quot;)]
static extern void WTSFreeMemory(IntPtr pMemory);

[StructLayout(LayoutKind.Sequential)]
private struct WTS_SESSION_INFO
{
    public Int32 SessionID;

    [MarshalAs(UnmanagedType.LPStr)]
    public String pWinStationName;

    public WTS_CONNECTSTATE_CLASS State;
}
public enum WTS_INFO_CLASS
{
    WTSInitialProgram,
    WTSApplicationName,
    WTSWorkingDirectory,
    WTSOEMId,
    WTSSessionId,
    WTSUserName,
    WTSWinStationName,
    WTSDomainName,
    WTSConnectState,
    WTSClientBuildNumber,
    WTSClientName,
    WTSClientDirectory,
    WTSClientProductId,
    WTSClientHardwareId,
    WTSClientAddress,
    WTSClientDisplay,
    WTSClientProtocolType
}
public enum WTS_CONNECTSTATE_CLASS
{
    WTSActive,
    WTSConnected,
    WTSConnectQuery,
```

```
        WTSShadow,
        WTSDisconnected,
        WTSIdle,
        WTSListen,
        WTSReset,
        WTSDown,
        WTSInit
    }

    public static IntPtr OpenServer(String Name)
    {
        IntPtr server = WTSOpenServer(Name);
        return server;
    }
    public static void CloseServer(IntPtr ServerHandle)
    {
        WTSCloseServer(ServerHandle);
    }

    public static IntPtr serverHandle = IntPtr.Zero;
    public static List<String> ListSessions(String ServerName)
    {

        List<String> resultList = new List<string>();
        serverHandle = OpenServer(ServerName);

        try
        {
        IntPtr SessionInfoPtr = IntPtr.Zero;
        IntPtr userPtr = IntPtr.Zero;
        IntPtr domainPtr = IntPtr.Zero;
        IntPtr ClientName = IntPtr.Zero;
        IntPtr ClientDisp = IntPtr.Zero;

        Int32 sessionCount = 0;
        Int32 retVal = WTSEnumerateSessions(serverHandle, 0, 1, ref SessionInfoPtr, ref
sessionCount);
        Int32 dataSize = Marshal.SizeOf(typeof(WTS_SESSION_INFO));
        Int32 currentSession = (int)SessionInfoPtr;
        uint bytes = 0;

        if (retVal != 0)
        {
            for (int i = 0; i < sessionCount; i++)
            {
            WTS_SESSION_INFO si =
```

```
(WTS_SESSION_INFO)Marshal.PtrToStructure((System.IntPtr)currentSession,
typeof(WTS_SESSION_INFO));
            currentSession += dataSize;

            WTSQuerySessionInformation(serverHandle, si.SessionID,
WTS_INFO_CLASS.WTSUserName, out userPtr, out bytes);
            WTSQuerySessionInformation(serverHandle, si.SessionID,
WTS_INFO_CLASS.WTSDomainName, out domainPtr, out bytes);
            WTSQuerySessionInformation(serverHandle, si.SessionID,
WTS_INFO_CLASS.WTSClientName, out ClientName, out bytes);
            WTSQuerySessionInformation(serverHandle, si.SessionID,
WTS_INFO_CLASS.WTSApplicationName, out ClientDisp, out bytes);

            resultList.Add(&amp;quot;Domain and User: &amp;quot; + Marshal.PtrToStringAnsi(domainPtr)
+ &amp;quot;\&amp;quot; + Marshal.PtrToStringAnsi(userPtr)
                + &amp;quot; Session ID: &amp;quot; + si.SessionID + &amp;quot; Tipo: &amp;quot; +
si.pWinStationName + &amp;quot; Stato : &amp;quot; + si.State
                + &amp;quot; Client: &amp;quot; + Marshal.PtrToStringAnsi(ClientName)
                + &amp;quot; Client Disp: &amp;quot; + Marshal.PtrToStringAnsi(ClientDisp));

                // cancello i puntatori
            WTSFreeMemory(userPtr);
            WTSFreeMemory(domainPtr);
            WTSFreeMemory(ClientName);
            WTSFreeMemory(ClientDisp);
            }

            WTSFreeMemory(SessionInfoPtr);
        }
        }
        finally
        {



        //Terminate_Session();

        CloseServer(serverHandle);
        }
            return resultList;

    }


    // TERMINA LE CONNESSIONI PARTENDO DALL'HANDLE DEL SERVER E DALLA
SESSIONID DEL UTENTE
```

```
    private static void Terminate_Session()
    {

        Boolean res = LogOffUser(3, serverHandle);
        Console.WriteLine(res.ToString());

        if (!res)
            Console.WriteLine(Marshal.GetLastWin32Error());

        Console.ReadKey();
    }


    internal static bool LogOffUser(int sessionid, IntPtr server)
    {


        return WTSDisconnectSession(server, sessionid, true);

    }




     }


}
```