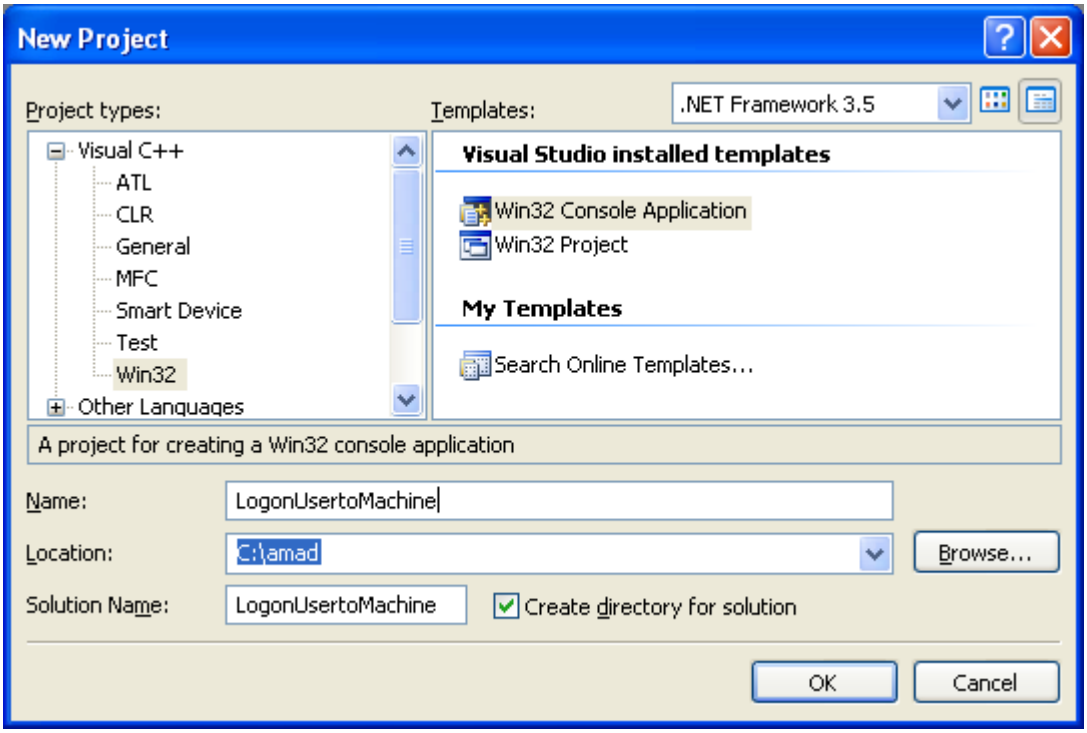
More Program Examples:

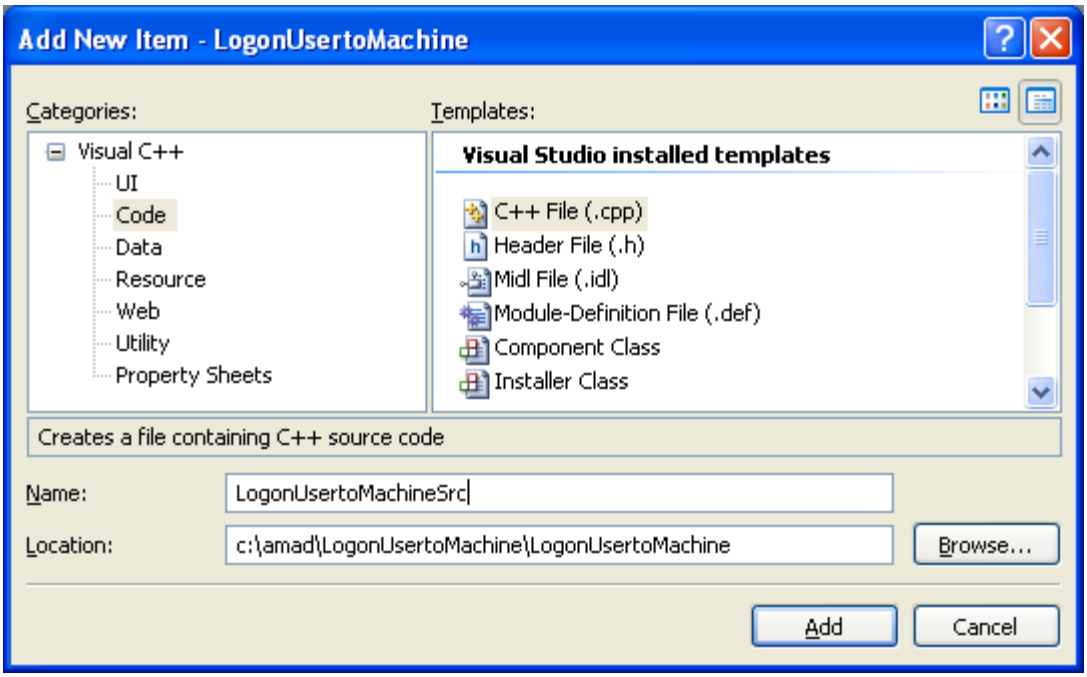
1. Log on a user to a machine Program Example
2. Simple Impersonation Program Example
3. Creating a Security Descriptor from Scratch for a New Object - a Registry key Example
4. Retrieving current user and domain names on Windows NT, Windows 2000, or Windows XP Code Example

Log on a user to a machine Program Example

The following program example demonstrates how to log on a user to a machine programmatically. It uses the LogonUser() function.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.



Next, add the following source code.

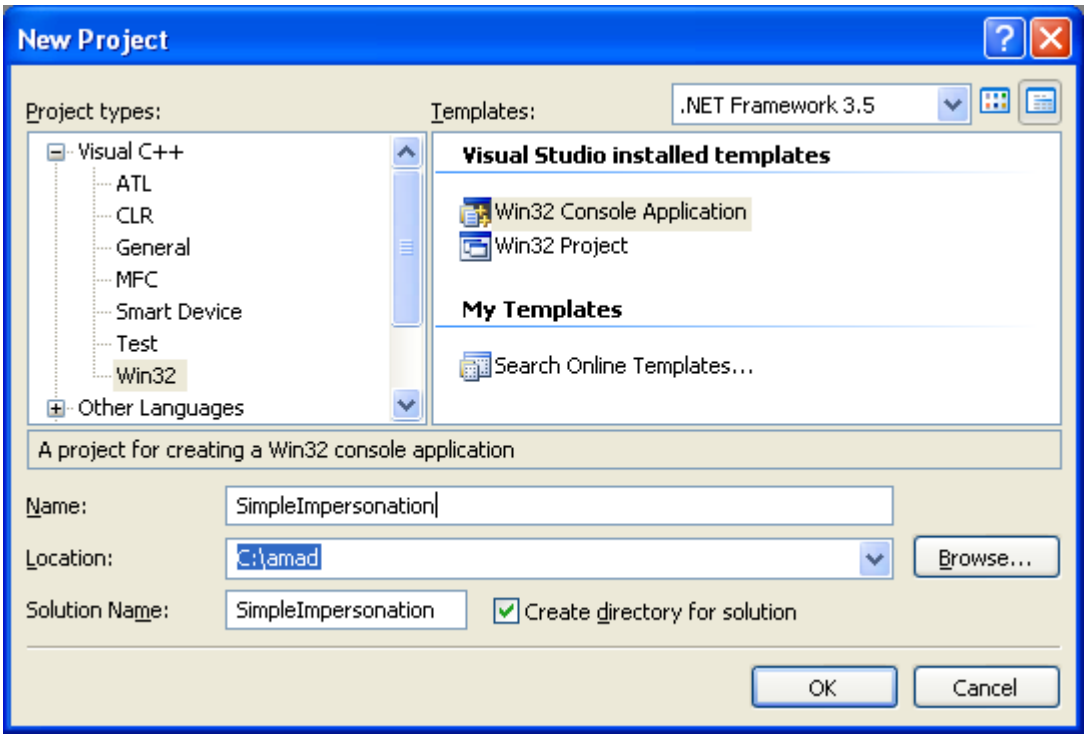
```
// Log a user on to the local computer. This computer is logged on as Mike
spoon,
// a user with Administrators group, then this program try to log
// on a restricted user named "usergedik" another valid user account
// created in the same machine
#include <windows.h>
#include <stdio.h>

int main(int argc, WCHAR **argv)
{
	// "usergedik" is just a restrictive user created
	// in the XP machine that runs this program
	LPTSTR lpszUsername = L"usergedik";
	// Local account database
	LPTSTR lpszDomain = L".";
	LPTSTR lpszPassword = L"123";
	DWORD dwLogonType = LOGON32_LOGON_INTERACTIVE;
	DWORD dwLogonProvider = LOGON32_PROVIDER_DEFAULT;
	HANDLE hToken;

	if(LogonUser(
		lpszUsername,	// Username
		lpszDomain,	// Domain or server where the Username is
reside
		lpszPassword,	// Plaintext password
		dwLogonType,	// Type of logon
		dwLogonProvider,	// The logon provider
		&hToken			// Pointer to handle that received the
token
		))
```

```
            wprintf(L"Well, \"%s\" user logged on to this machine
successfully!\n", lpszUsername);
      else
      {
            wprintf(L"%s failed to log on to this machine! error %u\n",
lpszUsername, GetLastError());
            exit(-1);
      }

      ////////////////
      // TODO: Use the logon token for the desired tasks
      ////////////////

      if(CloseHandle(hToken) != 0)
            wprintf(L"The handle that received the token has been
closed.\n");
      else
            wprintf(L"Something wrong, the handle cannot be closed! error:
%u\n", GetLastError());

      return 0;
}
```

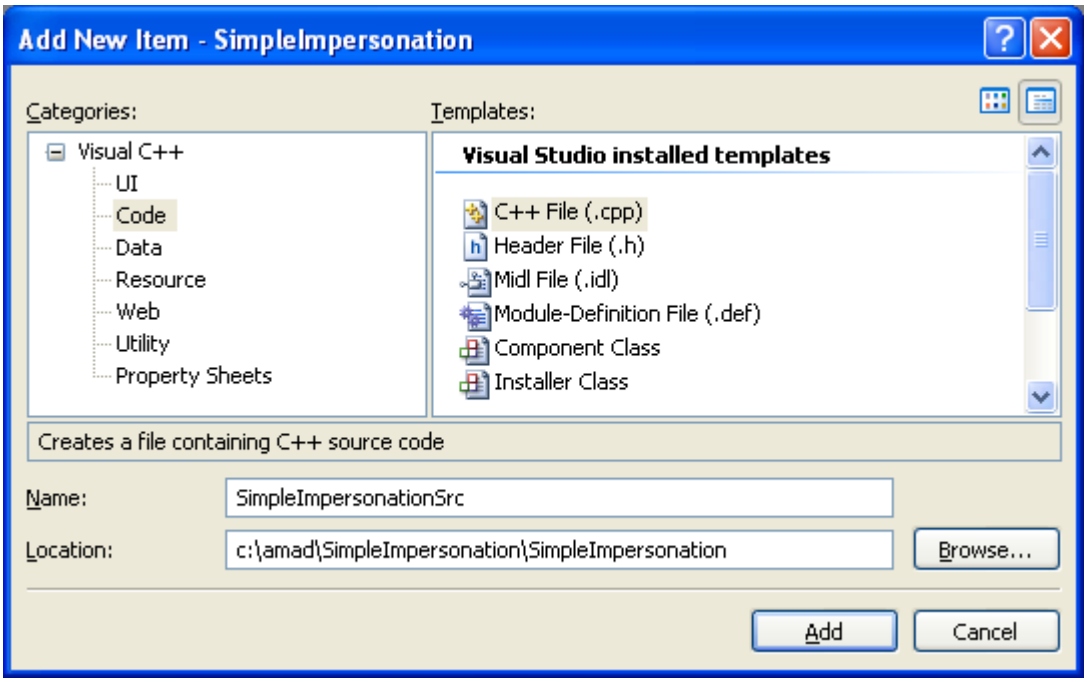
Build and run the project. The following screenshot is a sample output.

A Simple Impersonation Program Example

The following example shows simple impersonation using ImpersonateLoggedOnUser(). Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.

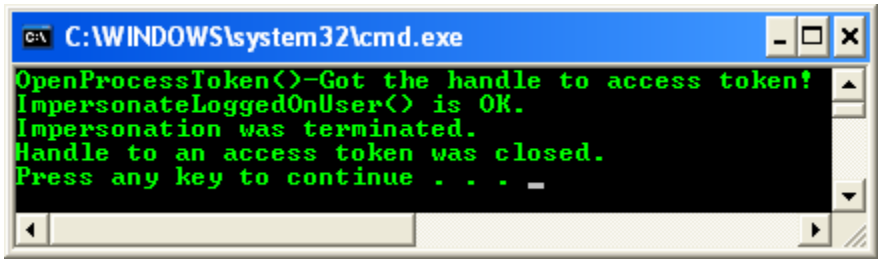


Next, add the following source code.

```
#include <windows.h>
#include <stdio.h>
```

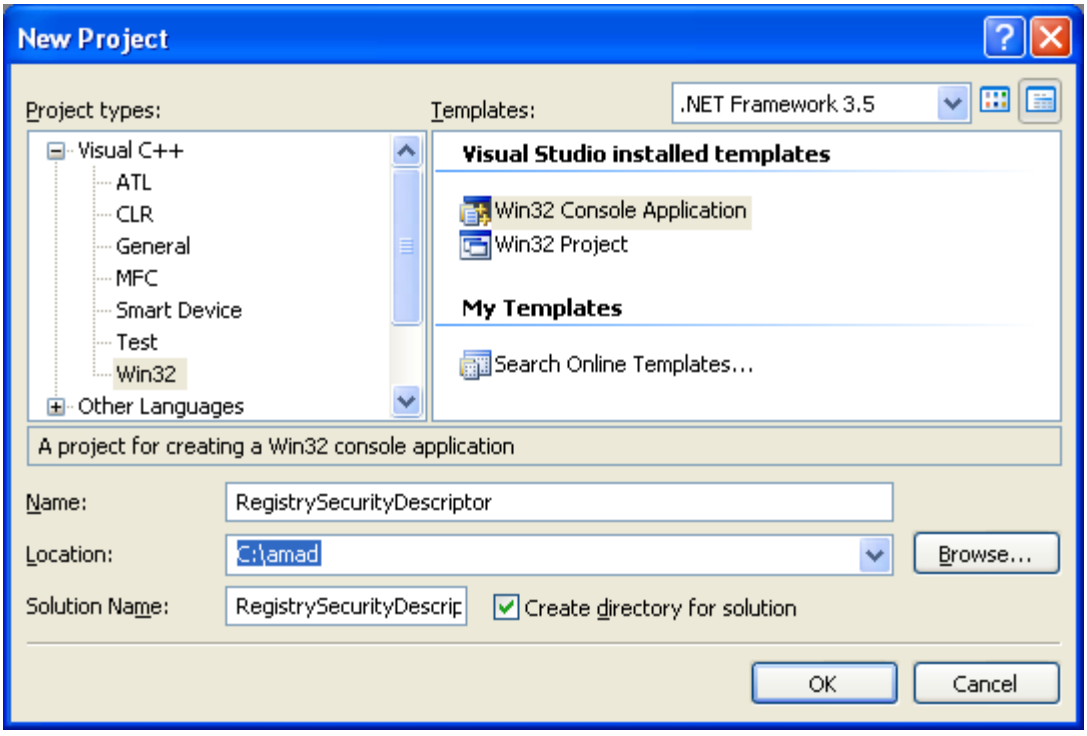
```
int wmain(int argc, WCHAR **argv)
{
	// Handle to token
	HANDLE hToken;

	// Open a handle to the access token for the
	// calling process that is the currently login access token
	if(!OpenProcessToken(GetCurrentProcess(), TOKEN_ALL_ACCESS, &hToken))
	{
		wprintf(L"OpenProcessToken()-Getting the handle to access token
failed, error %u\n", GetLastError());
	}
	else
		wprintf(L"OpenProcessToken()-Got the handle to access token!\n");

	// Lets the calling process impersonate the security context of a
logged-on user.
	if(ImpersonateLoggedOnUser(hToken))
		wprintf(L"ImpersonateLoggedOnUser() is OK.\n");
	else
	{
		wprintf(L"ImpersonateLoggedOnUser() failed, error %u.\n",
GetLastError());
		exit(-1);
	}

	/////////////////
	// TODO: Do other desired tasks
	/////////////////

	// Terminates the impersonation of a client.
	if(RevertToSelf())
		wprintf(L"Impersonation was terminated.\n");

	// Close the handle
	if(CloseHandle(hToken))
		wprintf(L"Handle to an access token was closed.\n");
	else
		wprintf(L"Failed to close the hToken handle! error %u\n",
GetLastError());

	return 0;
}
```

Build and run the project. The following screenshot is a sample output.

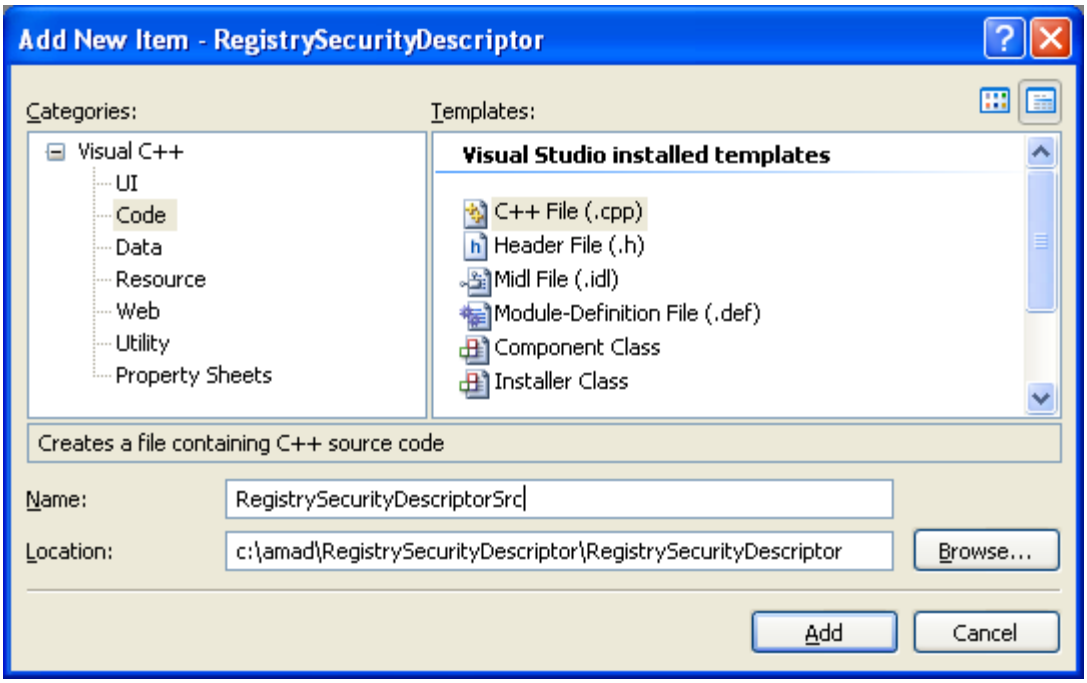


Creating a Security Descriptor from Scratch for a New Object, a Registry key Code Example

The following example creates a security descriptor for a new registry key using the following steps. Similar code can be used to create a security descriptor for other object types.
The example fills an array of EXPLICIT_ACCESS structures with the information for two ACEs. One ACE allows read access to everyone; the other ACE allows full access to administrators.
The EXPLICIT_ACCESS array is passed to the SetEntriesInAcl() function to create a DACL for the security descriptor.
After allocating memory for the security descriptor, the example calls the InitializeSecurityDescriptor() and SetSecurityDescriptorDacl() functions to initialize the security descriptor and attach the DACL.
The security descriptor is then stored in a SECURITY_ATTRIBUTES structure and passed to the RegCreateKeyEx() function, which attaches the security descriptor to the newly created key.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.



Next, add the following source code.

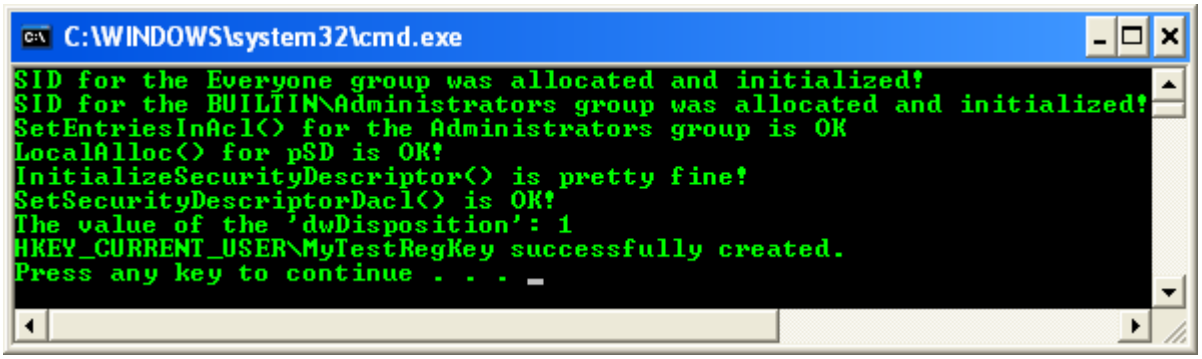
```
// Creating a security descriptor for a new object, a registry key...
#include <windows.h>
#include <stdio.h>
#include <aclapi.h>

// Buffer clean up routine
void Cleanup(PSID pEveryoneSID, PSID pAdminSID, PACL pACL,
PSECURITY_DESCRIPTOR pSD, HKEY hkSub)
{
      if(pEveryoneSID)
        FreeSid(pEveryoneSID);
    if(pAdminSID)
        FreeSid(pAdminSID);
    if(pACL)
        LocalFree(pACL);
    if(pSD)
        LocalFree(pSD);
    if(hkSub)
        RegCloseKey(hkSub);
}

int wmain(int argc, WCHAR **argv)
{
      DWORD dwRes, dwDisposition;
    PSID pEveryoneSID = NULL, pAdminSID = NULL;
    PACL pACL = NULL;
      PSECURITY_DESCRIPTOR pSD = {0};
    // An array of EXPLICIT_ACCESS structure
    EXPLICIT_ACCESS ea[2];
    SID_IDENTIFIER_AUTHORITY SIDAuthWorld = SECURITY_WORLD_SID_AUTHORITY;
```

```
    SID_IDENTIFIER_AUTHORITY SIDAuthNT = SECURITY_NT_AUTHORITY;
      SECURITY_ATTRIBUTES sa = {0};
    LONG lRes;
    HKEY hkSub = NULL;

    // Create a well-known SID for the Everyone group.
    if(!AllocateAndInitializeSid(&SIDAuthWorld, 1, SECURITY_WORLD_RID, 0, 0,
0, 0, 0, 0, 0, &pEveryoneSID))
    {
            wprintf(L"AllocateAndInitializeSid() failed, error %u\n",
GetLastError());
            Cleanup(pEveryoneSID, pAdminSID, pACL, pSD, hkSub);
    }
    else
        wprintf(L"SID for the Everyone group was allocated and
initialized!\n");

    // Initialize an EXPLICIT_ACCESS structure for an ACE.
    // The ACE will allow Everyone read access to the key.
    ZeroMemory(&ea, 2 * sizeof(EXPLICIT_ACCESS));
    ea[0].grfAccessPermissions = KEY_READ;
    ea[0].grfAccessMode = SET_ACCESS;
    ea[0].grfInheritance= NO_INHERITANCE;
    ea[0].Trustee.TrusteeForm = TRUSTEE_IS_SID;
    ea[0].Trustee.TrusteeType = TRUSTEE_IS_WELL_KNOWN_GROUP;
    ea[0].Trustee.ptstrName  = (LPTSTR) pEveryoneSID;

    // Create a SID for the BUILTIN\Administrators group.
    if(!AllocateAndInitializeSid(&SIDAuthNT, 2, SECURITY_BUILTIN_DOMAIN_RID,
            DOMAIN_ALIAS_RID_ADMINS,
            0, 0, 0, 0, 0, 0,
            &pAdminSID))
      {
        wprintf(L"AllocateAndInitializeSid() failed, error %u\n",
GetLastError());
        Cleanup(pEveryoneSID, pAdminSID, pACL, pSD, hkSub);
    }
    else
        wprintf(L"SID for the BUILTIN\\Administrators group was allocated and
initialized!\n");

    // Initialize an EXPLICIT_ACCESS structure for an ACE.
    // The ACE will allow the Administrators group full access to the key.
    ea[1].grfAccessPermissions = KEY_ALL_ACCESS;
    ea[1].grfAccessMode = SET_ACCESS;
    ea[1].grfInheritance= NO_INHERITANCE;
    ea[1].Trustee.TrusteeForm = TRUSTEE_IS_SID;
    ea[1].Trustee.TrusteeType = TRUSTEE_IS_GROUP;
    ea[1].Trustee.ptstrName  = (LPTSTR) pAdminSID;

    // Create a new ACL that contains the new ACEs.
    dwRes = SetEntriesInAcl(2, ea, NULL, &pACL);
    if(dwRes != ERROR_SUCCESS)
    {
        wprintf(L"SetEntriesInAcl() failed, error %u\n", GetLastError());
        Cleanup(pEveryoneSID, pAdminSID, pACL, pSD, hkSub);
    }
```

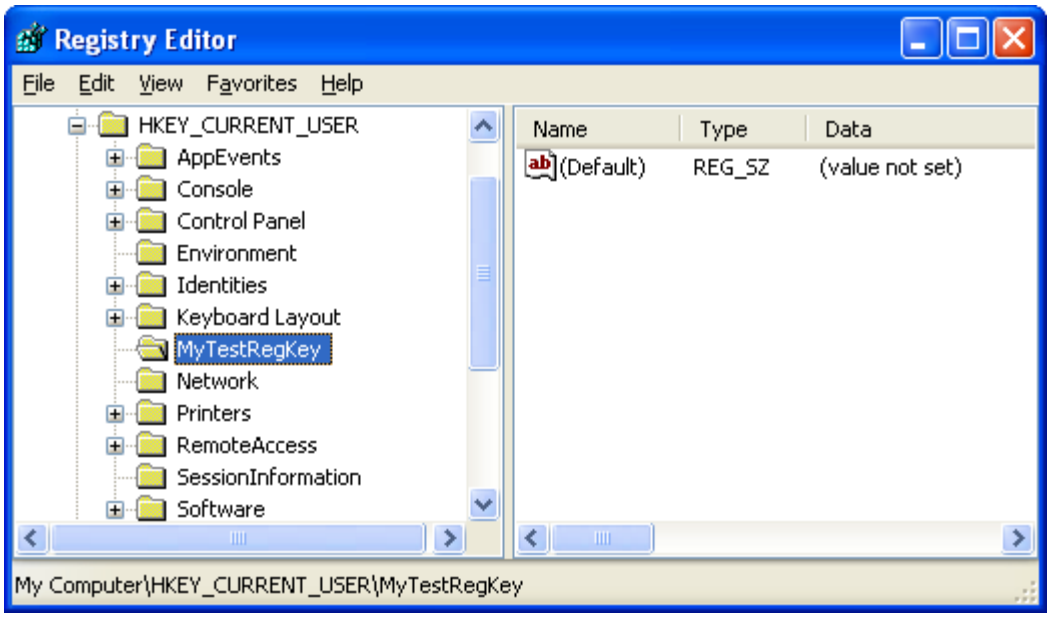
```
    else
       wprintf(L"SetEntriesInAcl() for the Administrators group is OK\n");

    // Initialize a security descriptor.
    pSD = (PSECURITY_DESCRIPTOR) LocalAlloc(LPTR,
SECURITY_DESCRIPTOR_MIN_LENGTH);
    if(pSD == NULL)
    {
        wprintf(L"LocalAlloc() for pSD failed, error %u\n", GetLastError());
        Cleanup(pEveryoneSID, pAdminSID, pACL, pSD, hkSub);
    }
    else
            wprintf(L"LocalAlloc() for pSD is OK!\n");

    if(!InitializeSecurityDescriptor(pSD, SECURITY_DESCRIPTOR_REVISION))
    {
        wprintf(L"InitializeSecurityDescriptor() failed, error %u\n",
GetLastError());
        Cleanup(pEveryoneSID, pAdminSID, pACL, pSD, hkSub);
    }
    else
       wprintf(L"InitializeSecurityDescriptor() is pretty fine!\n");

    // Add the ACL to the security descriptor.
    if(!SetSecurityDescriptorDacl(pSD,
            TRUE,       // bDaclPresent flag
            pACL,
            FALSE))           // not a default DACL
    {
        wprintf(L"SetSecurityDescriptorDacl() failed, error %u\n",
GetLastError());
        Cleanup(pEveryoneSID, pAdminSID, pACL, pSD, hkSub);
    }
    else
       wprintf(L"SetSecurityDescriptorDacl() is OK!\n");

    // Initialize a security attributes structure.
    sa.nLength = sizeof (SECURITY_ATTRIBUTES);
    sa.lpSecurityDescriptor = pSD;
    sa.bInheritHandle = FALSE;
    // Use the security attributes to set the security descriptor
      // when you create a registry key.
    lRes = RegCreateKeyEx(
            HKEY_CURRENT_USER,      // handle to an open key
            L"MyTestRegKey",  // name of the subkey
            0,                              // Reserved
            L"",                    // class or object type of this key, may
be ignored
            0,                              // Options
            KEY_READ | KEY_WRITE, // Access right for the key
            &sa,                    // Pointer to security attribute
structure,
                                            // can be inherited or not. NULL is
not inherited
            &hkSub,                         // variable that receives a handle
to the opened or created key
            &dwDisposition);  // variable that receives:
```

```
                                                // REG_CREATED_NEW_KEY - create new
key (non-exist)
                                                // REG_OPENED_EXISTING_KEY - just
open the existing key (already exist)
      // If successful
      if(lRes == 0)
      {
            wprintf(L"The value of the \'dwDisposition\': %u\n",
dwDisposition);
            wprintf(L"HKEY_CURRENT_USER\\MyTestRegKey successfully
created.\n");
      }
      else
            wprintf(L"Creating and opening HKEY_CURRENT_USER\\MyTestRegKey
was failed, error %u.\n", GetLastError());

    return 0;
}
```

Build and run the project. The following screenshot is a sample output.



Then, let verify the result using the regedt32/regedit registry tool.



Don't forget to delete the registry key that has been created.

Validate User Credentials on Microsoft Operating Systems Program Example

The sample code provided below shows how to call the SSPI services to perform credential validation. To use this method on Windows 95, Windows 98, and Windows Millennium Edition, you also have to enable the NTLM security services by opening Control Panel, Network, Access Control, and then selecting User-level access control. On Windows XP, the ForceGuest registry value is set to 1 by default in the following registry key:

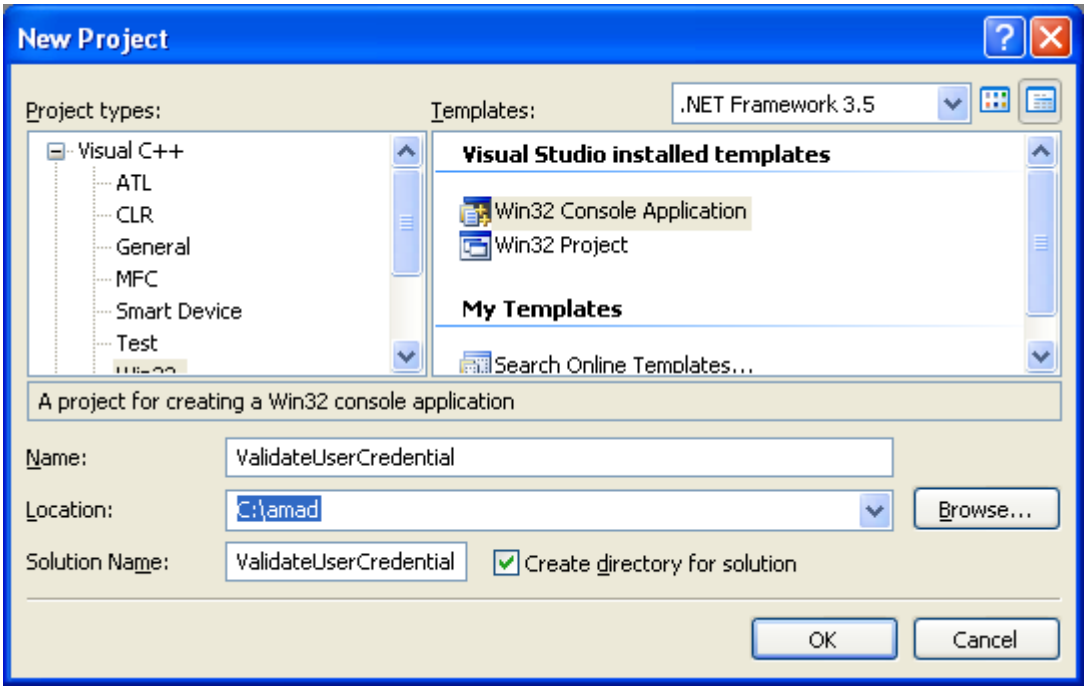
```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa
```

On a Windows XP computer that is a member of a workgroup:

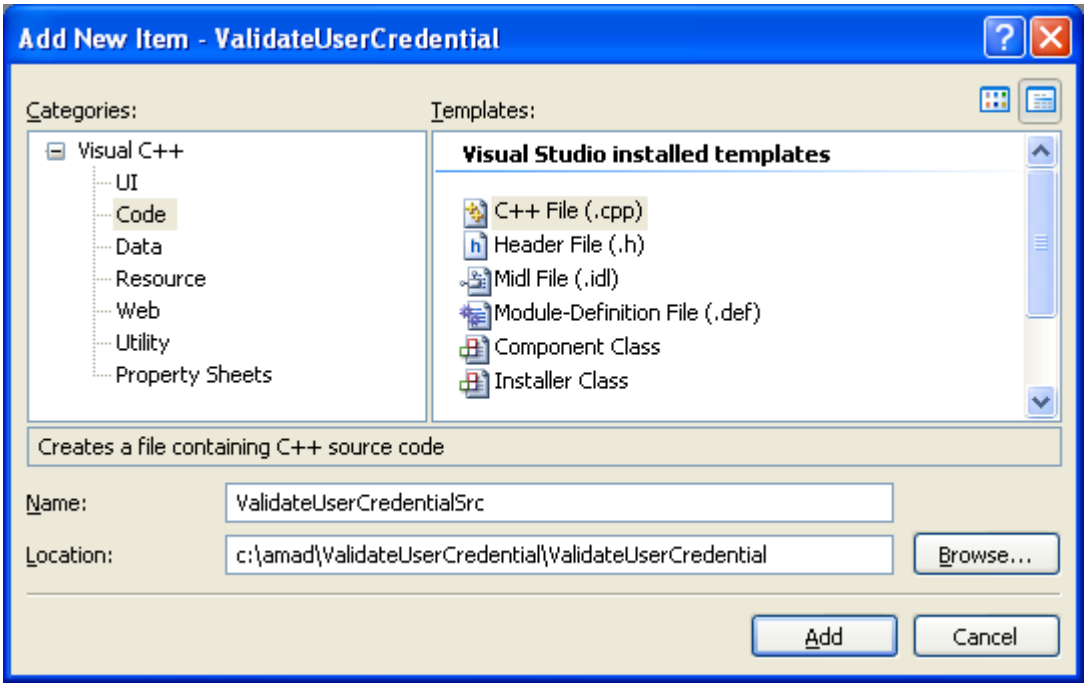
1. If ForceGuest is enabled (set to 1), SSPI will always try to log on using the Guest account.
2. If the Guest account is enabled, an SSPI logon will succeed as Guest for any user credentials.
3. If the Guest account is disabled, an SSPI logon will fail even for valid credentials.
4. If ForceGuest is disabled (set to 0), SSPI will log on as the specified user.

Additionally, if the Guest account is enabled, SSPI logon may succeed as Guest for user credentials that are not valid. The C sample code in this article demonstrates how you can check the access token of the established security context. The IsGuest() helper function in the sample code shows how you can verify that the logon occurred as the specified user or as Guest.

Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.



Next, add the following source code.

```
//////////////////////////////////////////////////////////////////////////
//
//  SSPI Authentication Sample
//
//  This program demonstrates how to use SSPI to authenticate user
credentials.
//
//  THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
//  ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED
//  TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
//  PARTICULAR PURPOSE.
//
//  Copyright (C) 2007.  Microsoft Corporation.  All rights reserved.
//////////////////////////////////////////////////////////////////////////
//

#define SECURITY_WIN32
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <sspi.h>
#include <lm.h>
#include <lmcons.h>

// Link to netapi32.lib
#pragma comment(lib, "netapi32.lib")
// Older versions of WinError.h do not have SEC_I_COMPLETE_NEEDED #define.
// So, in such an SDK environment setup, we will include issperr.h which has
the
// definition for SEC_I_COMPLETE_NEEDED. Include issperr.h only if
// SEC_I_COMPLETE_NEEDED is not defined.
#ifndef SEC_I_COMPLETE_NEEDED
#include <issperr.h>
#endif

typedef struct _AUTH_SEQ {
   BOOL fInitialized;
   BOOL fHaveCredHandle;
   BOOL fHaveCtxtHandle;
   CredHandle hcred;
   struct _SecHandle hctxt;
} AUTH_SEQ, *PAUTH_SEQ;


// Function pointers
ACCEPT_SECURITY_CONTEXT_FN        _AcceptSecurityContext     = NULL;
ACQUIRE_CREDENTIALS_HANDLE_FN     _AcquireCredentialsHandle  = NULL;
COMPLETE_AUTH_TOKEN_FN            _CompleteAuthToken         = NULL;
DELETE_SECURITY_CONTEXT_FN        _DeleteSecurityContext     = NULL;
FREE_CONTEXT_BUFFER_FN            _FreeContextBuffer         = NULL;
FREE_CREDENTIALS_HANDLE_FN        _FreeCredentialsHandle     = NULL;
INITIALIZE_SECURITY_CONTEXT_FN    _InitializeSecurityContext = NULL;
QUERY_SECURITY_PACKAGE_INFO_FN    _QuerySecurityPackageInfo  = NULL;
QUERY_SECURITY_CONTEXT_TOKEN_FN   _QuerySecurityContextToken = NULL;
```

```
#define CheckAndLocalFree(ptr) \
            if (ptr != NULL) \
            { \
               LocalFree(ptr); \
               ptr = NULL; \
            }

LPVOID RetrieveTokenInformationClass(HANDLE hToken,TOKEN_INFORMATION_CLASS
InfoClass,LPDWORD lpdwSize)
{
   LPVOID pInfo = NULL;
   BOOL fSuccess = FALSE;

   __try
   {
      *lpdwSize = 0;

      // Determine the size of the buffer needed
      GetTokenInformation(hToken,InfoClass,NULL,*lpdwSize, lpdwSize);

      if (GetLastError() != ERROR_INSUFFICIENT_BUFFER)
      {
         wprintf(L"GetTokenInformation() failed, error %d\n",
GetLastError());
         __leave;
      }

      // Allocate a buffer for getting token information
      pInfo = LocalAlloc(LPTR, *lpdwSize);
      if (pInfo == NULL)
      {
         wprintf(L"LocalAlloc() failed, error %d\n", GetLastError());
         __leave;
      }

      if (!GetTokenInformation(hToken,InfoClass,pInfo,*lpdwSize, lpdwSize))
      {
         wprintf(L"GetTokenInformation() failed, error %d\n",
GetLastError());
         __leave;
      }

      fSuccess = TRUE;
   }
   __finally
   {
      // Free pDomainAndUserName only if failed
      // Otherwise, the caller has to free after use
      if (fSuccess == FALSE)
      {
         CheckAndLocalFree(pInfo);
      }
   }
   return pInfo;
}
```

```
PSID GetUserSidFromWellKnownRid(DWORD Rid)
{
    PUSER_MODALS_INFO_2 umi2;
    NET_API_STATUS nas;
    UCHAR SubAuthorityCount;
    PSID pSid = NULL;
    BOOL bSuccess = FALSE; // assume failure

    nas = NetUserModalsGet(NULL, 2, (LPBYTE *)&umi2);

    if (nas != NERR_Success)
    {
        wprintf(L"NetUserModalsGet() failed, error %d\n", nas);
        SetLastError(nas);
        return NULL;
    }

    SubAuthorityCount = *GetSidSubAuthorityCount(umi2->usrmod2_domain_id);

    // Allocate storage for new Sid. account domain Sid + account Rid
    pSid =
(PSID)LocalAlloc(LPTR,GetSidLengthRequired((UCHAR)(SubAuthorityCount + 1)));

    if (pSid != NULL)
    {
        if (InitializeSid(pSid,GetSidIdentifierAuthority(umi2-
>usrmod2_domain_id),(BYTE)(SubAuthorityCount+1)))
        {
            DWORD SubAuthIndex = 0;

            // Copy existing subauthorities from account domain Sid into new
Sid
            for (; SubAuthIndex < SubAuthorityCount ; SubAuthIndex++)
            {
                *GetSidSubAuthority(pSid, SubAuthIndex) =
*GetSidSubAuthority(umi2->usrmod2_domain_id,
                                    SubAuthIndex);
            }

            // Append Rid to new Sid
            *GetSidSubAuthority(pSid, SubAuthorityCount) = Rid;
        }
    }

    NetApiBufferFree(umi2);

    return pSid;
}

BOOL IsGuest(HANDLE hToken)
{
    BOOL fGuest = FALSE;
    PSID pGuestSid = NULL;
    PSID pUserSid = NULL;
    TOKEN_USER *pUserInfo = NULL;
    DWORD dwSize = 0;
```

```
    pGuestSid = GetUserSidFromWellKnownRid(DOMAIN_USER_RID_GUEST);
    if (pGuestSid == NULL)
        return fGuest;

    // Get user information
    pUserInfo = (TOKEN_USER *)RetrieveTokenInformationClass(hToken,
TokenUser, &dwSize);
    if (pUserInfo != NULL)
    {
        if (EqualSid(pGuestSid, pUserInfo->User.Sid))
            fGuest = TRUE;
    }

    CheckAndLocalFree(pUserInfo);
    CheckAndLocalFree(pGuestSid);

    return fGuest;
}

//////////////////////////////////////////////////////
void UnloadSecurityDll(HMODULE hModule) {

   if (hModule)
      FreeLibrary(hModule);

   _AcceptSecurityContext      = NULL;
   _AcquireCredentialsHandle   = NULL;
   _CompleteAuthToken          = NULL;
   _DeleteSecurityContext      = NULL;
   _FreeContextBuffer          = NULL;
   _FreeCredentialsHandle      = NULL;
   _InitializeSecurityContext  = NULL;
   _QuerySecurityPackageInfo   = NULL;
   _QuerySecurityContextToken  = NULL;
}

//////////////////////////////////////////////////////
HMODULE LoadSecurityDll()
{
   HMODULE hModule;
   BOOL    fAllFunctionsLoaded = FALSE;
   TCHAR   lpszDLL[MAX_PATH];
   OSVERSIONINFO VerInfo;

   //  Find out which security DLL to use, depending on
   //  whether we are on Windows NT or Windows 95, Windows 2000,
   //  Windows XP, or Windows Server 2003
   //  We have to use security.dll on Windows NT 4.0.
   //  All other operating systems, we have to use Secur32.dll
   VerInfo.dwOSVersionInfoSize = sizeof (OSVERSIONINFO);
   if (!GetVersionEx (&VerInfo))   // If this fails, something has gone wrong
   {
      return FALSE;
   }

   if (VerInfo.dwPlatformId == VER_PLATFORM_WIN32_NT &&
VerInfo.dwMajorVersion == 4 && VerInfo.dwMinorVersion == 0)
```

```
    {
        lstrcpy(lpszDLL, L"security.dll");
    }
    else
    {
        lstrcpy (lpszDLL, L"secur32.dll");
    }

    hModule = LoadLibrary(lpszDLL);
    if (!hModule)
        return NULL;

    __try {
          _AcceptSecurityContext =
(ACCEPT_SECURITY_CONTEXT_FN)GetProcAddress(hModule, "AcceptSecurityContext");
        if (!_AcceptSecurityContext)
            __leave;

#ifdef UNICODE
        _AcquireCredentialsHandle = (ACQUIRE_CREDENTIALS_HANDLE_FN)
              GetProcAddress(hModule, "AcquireCredentialsHandleW");
#else
        _AcquireCredentialsHandle = (ACQUIRE_CREDENTIALS_HANDLE_FN)
              GetProcAddress(hModule, "AcquireCredentialsHandleA");
#endif
        if (!_AcquireCredentialsHandle)
            __leave;

        // CompleteAuthToken is not present on Windows 9x Secur32.dll
        // Do not check for the availablity of the function if it is NULL;
        _CompleteAuthToken = (COMPLETE_AUTH_TOKEN_FN)GetProcAddress(hModule,
"CompleteAuthToken");

        _DeleteSecurityContext =
(DELETE_SECURITY_CONTEXT_FN)GetProcAddress(hModule, "DeleteSecurityContext");
        if (!_DeleteSecurityContext)
            __leave;

        _FreeContextBuffer = (FREE_CONTEXT_BUFFER_FN)GetProcAddress(hModule,
"FreeContextBuffer");
        if (!_FreeContextBuffer)
            __leave;

        _FreeCredentialsHandle =
(FREE_CREDENTIALS_HANDLE_FN)GetProcAddress(hModule, "FreeCredentialsHandle");
        if (!_FreeCredentialsHandle)
            __leave;

#ifdef UNICODE
        _InitializeSecurityContext =
(INITIALIZE_SECURITY_CONTEXT_FN)GetProcAddress(hModule,
"InitializeSecurityContextW");
#else
        _InitializeSecurityContext = (INITIALIZE_SECURITY_CONTEXT_FN)
              GetProcAddress(hModule, "InitializeSecurityContextA");
#endif
        if (!_InitializeSecurityContext)
```

```
            __leave;

#ifdef UNICODE
      _QuerySecurityPackageInfo =
(QUERY_SECURITY_PACKAGE_INFO_FN)GetProcAddress(hModule,
"QuerySecurityPackageInfoW");
#else
      _QuerySecurityPackageInfo = (QUERY_SECURITY_PACKAGE_INFO_FN)
            GetProcAddress(hModule, "QuerySecurityPackageInfoA");
#endif
      if (!_QuerySecurityPackageInfo)
         __leave;

      _QuerySecurityContextToken =
(QUERY_SECURITY_CONTEXT_TOKEN_FN)GetProcAddress(hModule,
"QuerySecurityContextToken");
      if (!_QuerySecurityContextToken)
         __leave;

      fAllFunctionsLoaded = TRUE;

   } __finally {

      if (!fAllFunctionsLoaded)
        {
         UnloadSecurityDll(hModule);
         hModule = NULL;
      }
   }
   return hModule;
}

//////////////////////////////////////////////////////////////////////////////
//
/*++
 Routine Description:
   Optionally takes an input buffer coming from the server and returns
   a buffer of information to send back to the server.  Also returns
   an indication of whether or not the context is complete.

 Return Value:
   Returns TRUE if successful; otherwise FALSE.
--*/
BOOL GenClientContext(PAUTH_SEQ pAS, PSEC_WINNT_AUTH_IDENTITY pAuthIdentity,
      PVOID pIn, DWORD cbIn, PVOID pOut, PDWORD pcbOut, PBOOL pfDone)
{
      SECURITY_STATUS ss;
   TimeStamp       tsExpiry;
   SecBufferDesc   sbdOut;
   SecBuffer       sbOut;
   SecBufferDesc   sbdIn;
   SecBuffer       sbIn;
   ULONG           fContextAttr;

   if (!pAS->fInitialized)
   {
      ss = _AcquireCredentialsHandle(NULL, L"NTLM",
```

```
            SECPKG_CRED_OUTBOUND, NULL, pAuthIdentity, NULL, NULL,
            &pAS->hcred, &tsExpiry);
    if (ss < 0) {
        fprintf(stderr, "AcquireCredentialsHandle failed with %08X\n", ss);
        return FALSE;
    }

    pAS->fHaveCredHandle = TRUE;
}

// Prepare output buffer
sbdOut.ulVersion = 0;
sbdOut.cBuffers = 1;
sbdOut.pBuffers = &sbOut;

sbOut.cbBuffer = *pcbOut;
sbOut.BufferType = SECBUFFER_TOKEN;
sbOut.pvBuffer = pOut;

// Prepare input buffer
if (pAS->fInitialized)
{
    sbdIn.ulVersion = 0;
    sbdIn.cBuffers = 1;
    sbdIn.pBuffers = &sbIn;

    sbIn.cbBuffer = cbIn;
    sbIn.BufferType = SECBUFFER_TOKEN;
    sbIn.pvBuffer = pIn;
}

ss = _InitializeSecurityContext(&pAS->hcred,
       pAS->fInitialized ? &pAS->hctxt : NULL, NULL, 0, 0,
       SECURITY_NATIVE_DREP, pAS->fInitialized ? &sbdIn : NULL,
       0, &pAS->hctxt, &sbdOut, &fContextAttr, &tsExpiry);
if (ss < 0)
{
    // <winerror.h>
    fwprintf(stderr, L"InitializeSecurityContext failed with %08X\n", ss);
    return FALSE;
}

pAS->fHaveCtxtHandle = TRUE;

// If necessary, complete token
if (ss == SEC_I_COMPLETE_NEEDED || ss == SEC_I_COMPLETE_AND_CONTINUE)
{
       if (_CompleteAuthToken)
       {
       ss = _CompleteAuthToken(&pAS->hctxt, &sbdOut);
       if (ss < 0)
           {
           fwprintf(stderr, L"CompleteAuthToken failed with %08X\n", ss);
           return FALSE;
       }
    }
    else
```

```
        {
         fwprintf (stderr, L"CompleteAuthToken not supported.\n");
         return FALSE;
      }
   }

   *pcbOut = sbOut.cbBuffer;

   if (!pAS->fInitialized)
      pAS->fInitialized = TRUE;

   *pfDone = !(ss == SEC_I_CONTINUE_NEEDED || ss ==
SEC_I_COMPLETE_AND_CONTINUE );

   return TRUE;
}

//////////////////////////////////////////////////////////////////////////
//
/*++
 Routine Description:
    Takes an input buffer coming from the client and returns a buffer
    to be sent to the client.  Also returns an indication of whether or
    not the context is complete.

 Return Value:
    Returns TRUE if successful; otherwise FALSE.
--*/
BOOL GenServerContext(PAUTH_SEQ pAS, PVOID pIn, DWORD cbIn, PVOID pOut,PDWORD
pcbOut, PBOOL pfDone)
{
   SECURITY_STATUS ss;
   TimeStamp       tsExpiry;
   SecBufferDesc   sbdOut;
   SecBuffer       sbOut;
   SecBufferDesc   sbdIn;
   SecBuffer       sbIn;
   ULONG           fContextAttr;

   if (!pAS->fInitialized)
   {
      ss = _AcquireCredentialsHandle(NULL, L"NTLM",
            SECPKG_CRED_INBOUND, NULL, NULL, NULL, NULL, &pAS->hcred,
&tsExpiry);
      if (ss < 0)
        {
         fwprintf(stderr, L"AcquireCredentialsHandle failed with %08X\n",
ss);
         return FALSE;
      }

      pAS->fHaveCredHandle = TRUE;
   }

   // Prepare output buffer
   sbdOut.ulVersion = 0;
   sbdOut.cBuffers = 1;
```

```
   sbdOut.pBuffers = &sbOut;

   sbOut.cbBuffer = *pcbOut;
   sbOut.BufferType = SECBUFFER_TOKEN;
   sbOut.pvBuffer = pOut;

   // Prepare input buffer
   sbdIn.ulVersion = 0;
   sbdIn.cBuffers = 1;
   sbdIn.pBuffers = &sbIn;

   sbIn.cbBuffer = cbIn;
   sbIn.BufferType = SECBUFFER_TOKEN;
   sbIn.pvBuffer = pIn;

   ss = _AcceptSecurityContext(&pAS->hcred,
         pAS->fInitialized ? &pAS->hctxt : NULL, &sbdIn, 0,
         SECURITY_NATIVE_DREP, &pAS->hctxt, &sbdOut, &fContextAttr,
&tsExpiry);
   if (ss < 0)
   {
      fwprintf(stderr, L"AcceptSecurityContext failed with %08X\n", ss);
      return FALSE;
   }

   pAS->fHaveCtxtHandle = TRUE;

   // If necessary, complete token
   if (ss == SEC_I_COMPLETE_NEEDED || ss == SEC_I_COMPLETE_AND_CONTINUE) {

      if (_CompleteAuthToken)
        {
         ss = _CompleteAuthToken(&pAS->hctxt, &sbdOut);
         if (ss < 0)
             {
            fwprintf(stderr, L"CompleteAuthToken failed with %08X\n", ss);
            return FALSE;
         }
      }
      else
        {
         fwprintf (stderr, L"CompleteAuthToken not supported.\n");
         return FALSE;
      }
   }

   *pcbOut = sbOut.cbBuffer;

   if (!pAS->fInitialized)
      pAS->fInitialized = TRUE;

   *pfDone = !(ss == SEC_I_CONTINUE_NEEDED || ss ==
SEC_I_COMPLETE_AND_CONTINUE);

   return TRUE;
}
```

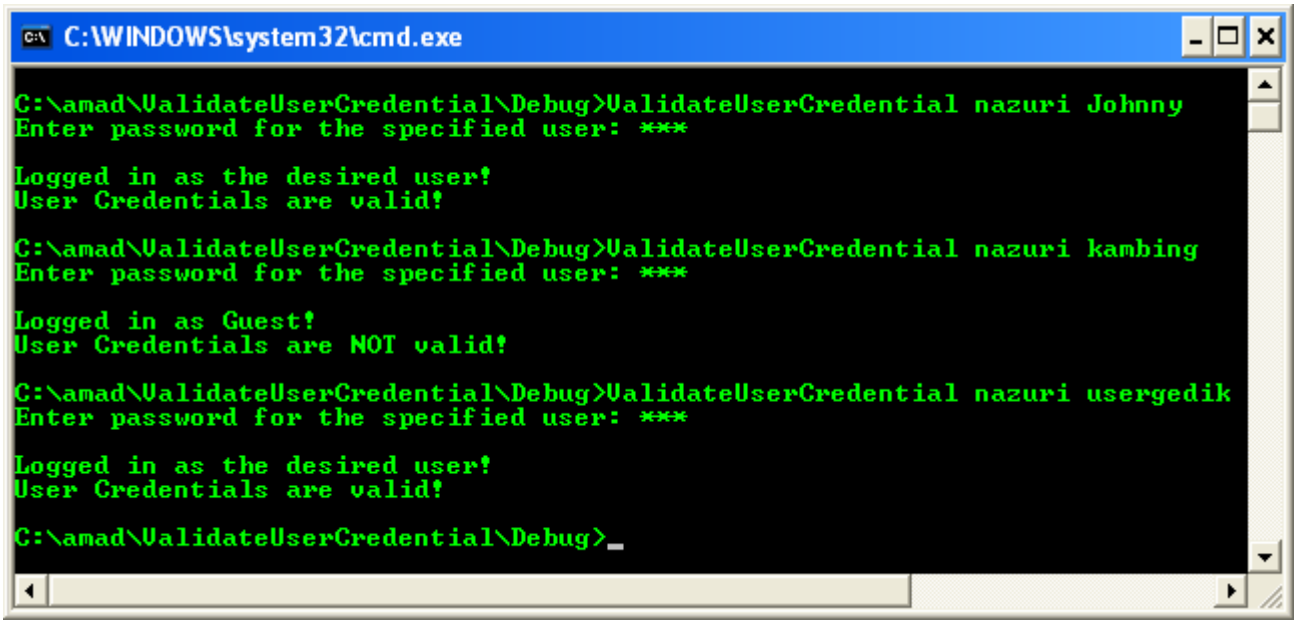
```
////////////////////////////////////////////////////////////////////////////
//
BOOL WINAPI SSPLogonUser(LPTSTR szDomain, LPTSTR szUser, LPTSTR szPassword) {

   AUTH_SEQ    asServer   = {0};
   AUTH_SEQ    asClient   = {0};
   BOOL        fDone      = FALSE;
   BOOL        fResult    = FALSE;
   DWORD       cbOut      = 0;
   DWORD       cbIn       = 0;
   DWORD       cbMaxToken = 0;
   PVOID       pClientBuf = NULL;
   PVOID       pServerBuf = NULL;
   PSecPkgInfo pSPI       = NULL;
   HMODULE     hModule    = NULL;

   // Enables passing a particular username and password to the run-time
library for
   // the purpose of authentication. The structure is valid for Windows and
Macintosh.
   SEC_WINNT_AUTH_IDENTITY ai;

   __try
   {
      hModule = LoadSecurityDll();
      if (!hModule)
         __leave;

      // Get max token size
      _QuerySecurityPackageInfo(L"NTLM", &pSPI);
      cbMaxToken = pSPI->cbMaxToken;
      _FreeContextBuffer(pSPI);

      // Allocate buffers for client and server messages
      pClientBuf = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, cbMaxToken);
      pServerBuf = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, cbMaxToken);

      // Initialize auth identity structure
      SecureZeroMemory(&ai, sizeof(ai));
        // The following are crapped!!!
        // We need to cast those back to ANSI!!!
        // http://msdn.microsoft.com/en-us/library/aa378664%28VS.85%29.aspx
#if defined(UNICODE) || defined(_UNICODE)
      ai.Domain = (unsigned short *)szDomain;
      ai.DomainLength = lstrlen(szDomain);
      ai.User = (unsigned short *)szUser;
      ai.UserLength = lstrlen(szUser);
      ai.Password = (unsigned short *)szPassword;
      ai.PasswordLength = lstrlen(szPassword);
      ai.Flags = SEC_WINNT_AUTH_IDENTITY_UNICODE;
#else
      ai.Domain = (unsigned short *)szDomain;
      ai.DomainLength = lstrlen(szDomain);
      ai.User = (unsigned short *)szUser;
      ai.UserLength = lstrlen(szUser);
      ai.Password = (unsigned short *)szPassword;
      ai.PasswordLength = lstrlen(szPassword);
```

```
      ai.Flags = SEC_WINNT_AUTH_IDENTITY_ANSI;
#endif

      // Prepare client message (negotiate) .
      cbOut = cbMaxToken;
      if (!GenClientContext(&asClient, &ai, NULL, 0, pClientBuf, &cbOut,
&fDone))
         __leave;

      // Prepare server message (challenge) .
      cbIn = cbOut;
      cbOut = cbMaxToken;
      if (!GenServerContext(&asServer, pClientBuf, cbIn, pServerBuf,
&cbOut,&fDone))
         __leave;

         // Most likely failure: AcceptServerContext fails with
SEC_E_LOGON_DENIED
         // in the case of bad szUser or szPassword.
         // Unexpected Result: Logon will succeed if you pass in a bad szUser
and
         // the guest account is enabled in the specified domain.

      // Prepare client message (authenticate) .
      cbIn = cbOut;
      cbOut = cbMaxToken;
      if (!GenClientContext(&asClient, &ai, pServerBuf, cbIn, pClientBuf,
&cbOut,&fDone))
         __leave;

      // Prepare server message (authentication) .
      cbIn = cbOut;
      cbOut = cbMaxToken;
      if (!GenServerContext(&asServer, pClientBuf, cbIn, pServerBuf,
&cbOut,&fDone))
         __leave;

      fResult = TRUE;

      {
         HANDLE hToken = NULL;

         if (_QuerySecurityContextToken(&asServer.hctxt, &hToken) == 0)
         {
            if (IsGuest(hToken))
            {
               wprintf(L"Logged in as Guest!\n");
               fResult = FALSE;
            }
            else
               wprintf(L"Logged in as the desired user!\n");
            CloseHandle(hToken);
         }
      }


   } __finally {
```

```
      // Clean up resources
      if (asClient.fHaveCtxtHandle)
         _DeleteSecurityContext(&asClient.hctxt);

      if (asClient.fHaveCredHandle)
         _FreeCredentialsHandle(&asClient.hcred);

      if (asServer.fHaveCtxtHandle)
         _DeleteSecurityContext(&asServer.hctxt);

      if (asServer.fHaveCredHandle)
         _FreeCredentialsHandle(&asServer.hcred);

      if (hModule)
         UnloadSecurityDll(hModule);

      HeapFree(GetProcessHeap(), 0, pClientBuf);
      HeapFree(GetProcessHeap(), 0, pServerBuf);

   }

   return fResult;
}

// The GetConsoleInput function gets an array of characters from the
// keyboard, while printing only asterisks to the screen.
void GetConsoleInput(TCHAR* strInput, int intMaxChars)
{
      char ch;
      char minChar = ' ';
      minChar++;

      ch = _getch();
      while (ch != '\r')
      {
            if (ch == '\b' && wcslen(strInput) > 0)
            {
                  strInput[wcslen(strInput)-1]   = '\0';
                  printf("\b \b");
            }
            else if (ch >= minChar && (int)wcslen(strInput) < intMaxChars)
            {
                  strInput[wcslen(strInput)+1] = '\0';
                  strInput[wcslen(strInput)]   = ch;
                  _putch('*');
            }
            ch = _getch();
      }
      _putch('\n');
}
int wmain(int argc, WCHAR **argv)
{
      WCHAR password[PWLEN+1];

   if (argc != 3)
      {
```

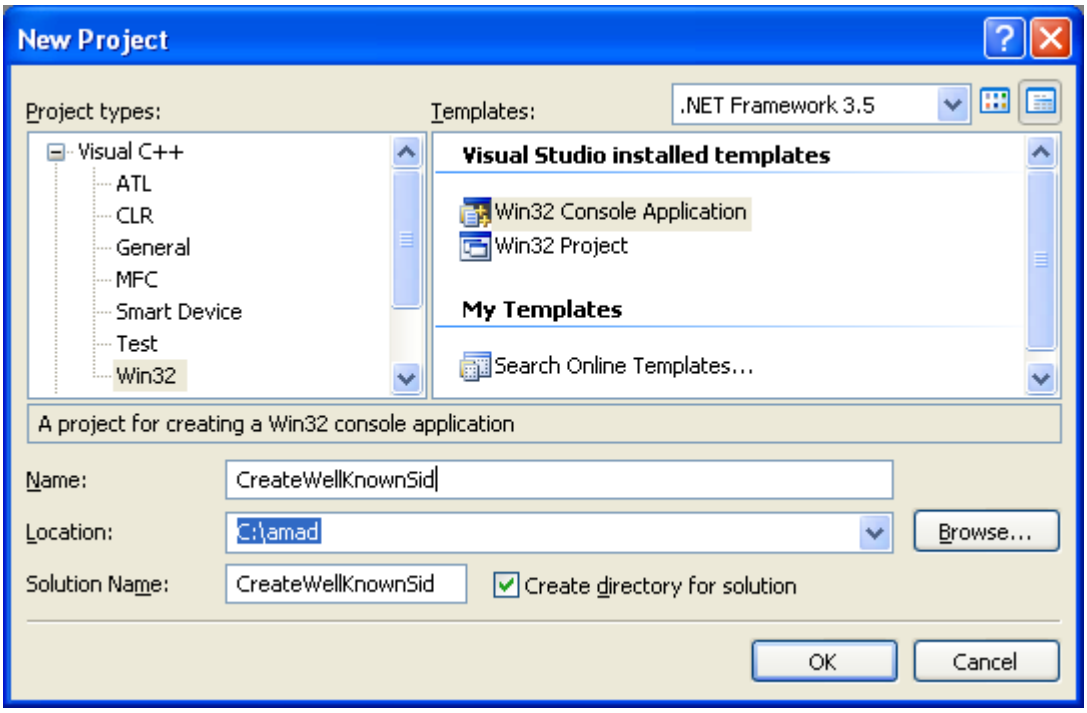
```
            wprintf(L"Usage: %s DomainName UserName\n", argv[0]);
            return 1;
        }

        wprintf(L"Enter password for the specified user: ");
        password[0] = 0;
        GetConsoleInput(password, PWLEN);
        wprintf(L"\n");
    // argv[1] - Domain Name
    // argv[2] - User Name
    if (SSPLogonUser(argv[1], argv[2], password))
    {
        wprintf(L"User Credentials are valid!\n");
    }
    else
        wprintf(L"User Credentials are NOT valid!\n");
}
```

Build and run the project. The following screenshot is a sample output.

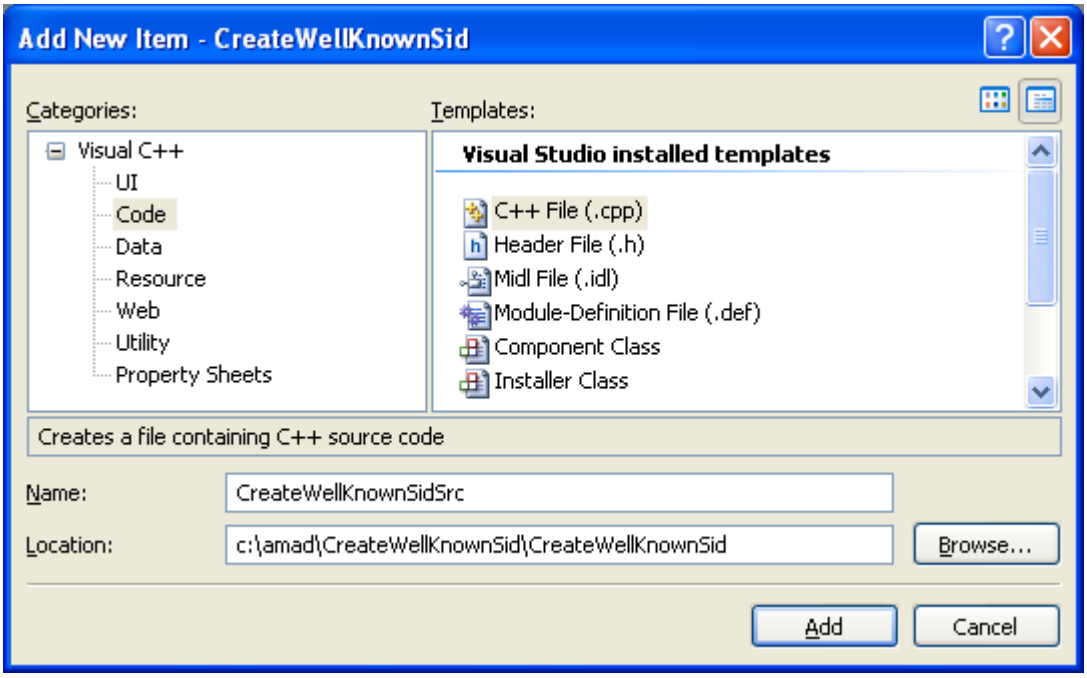


Creating A Well Known SID Program Example

The following example shows how to create a SID for the Everyone group.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.



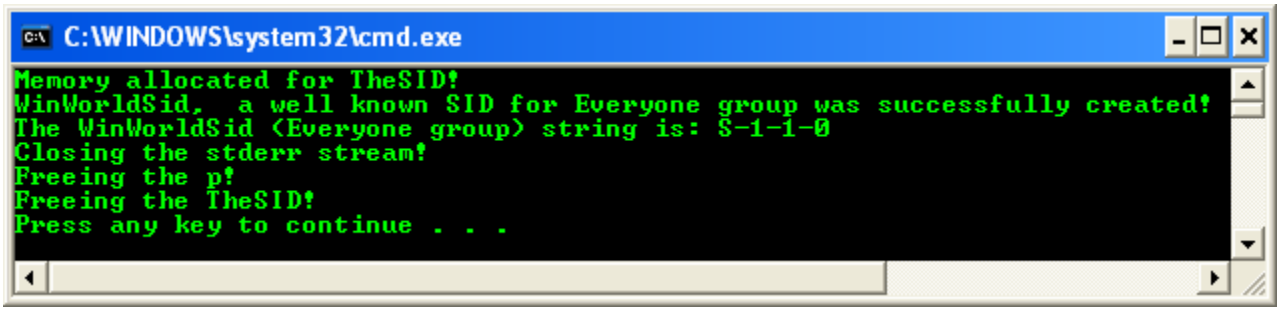
Next, add the following source code.

```
#include <windows.h>
#include <stdio.h>
#include <Sddl.h>
```

```
int wmain(int argc, WCHAR *argv[])
{
	DWORD SidSize;
	PSID TheSID;
	LPTSTR p;

	SidSize = SECURITY_MAX_SID_SIZE;

	// Allocate enough memory for the largest possible SID.
	if(!(TheSID = LocalAlloc(LMEM_FIXED, SidSize)))
	{
		fwprintf(stderr, L"Could not allocate memory for TheSID.\n");
		exit(1);
	}
	else
		wprintf(L"Memory allocated for TheSID!\n");

	// Create a SID for the Everyone group on the local computer.
	// http://msdn.microsoft.com/en-us/library/aa379649%28VS.85%29.aspx
	// http://support.microsoft.com/kb/243330
	// http://msdn.microsoft.com/en-us/library/aa379650%28VS.85%29.aspx
	if(!CreateWellKnownSid(WinWorldSid, NULL, TheSID, &SidSize))
	{
		fwprintf(stderr, L"CreateWellKnownSid() failed, error %u",
GetLastError());
	}
	else
	{
		wprintf(L"WinWorldSid,  a well known SID for Everyone group was
successfully created!\n");

		////////////////////////////////////////
		// TODO: Use the binary SID as needed.
		////////////////////////////////////////

		// Get the string version of the SID (S-1-1-0).
		if(!(ConvertSidToStringSid(TheSID, &p)))
		{
			fwprintf(stderr, L"Error during
ConvertSidToStringSid().\n");
			exit(1);
		}
		else
			wprintf(L"The WinWorldSid (Everyone group) string is:
%s\n", p);

		////////////////////////////
		// TODO: Use the string SID as needed.
		////////////////////////////////////////

		// When done, free the memory used.
		if(fclose(stderr) == 0)
			wprintf(L"Closing the stderr stream!\n");
		else
			wprintf(L"Failed to close stderr stream, error %u\n",
GetLastError());
```

```
            if(LocalFree(p) == NULL)
                  wprintf(L"Freeing the p!\n");
            else
                  wprintf(L"Failed to free p, error %u\n", GetLastError());

            if(LocalFree(TheSID) == NULL)
                  wprintf(L"Freeing the TheSID!\n");
            else
                  wprintf(L"Failed to free TheSID, error %u\n",
GetLastError());

            return 0;
      }
}
```

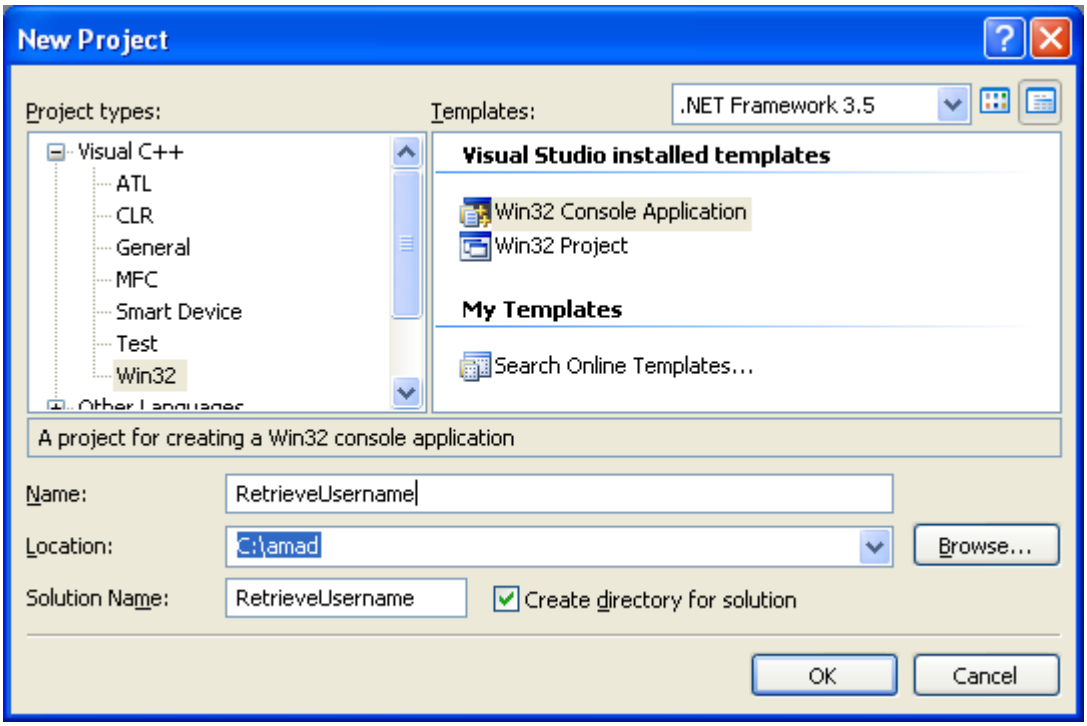
Build and run the project. The following screenshot is a sample output.



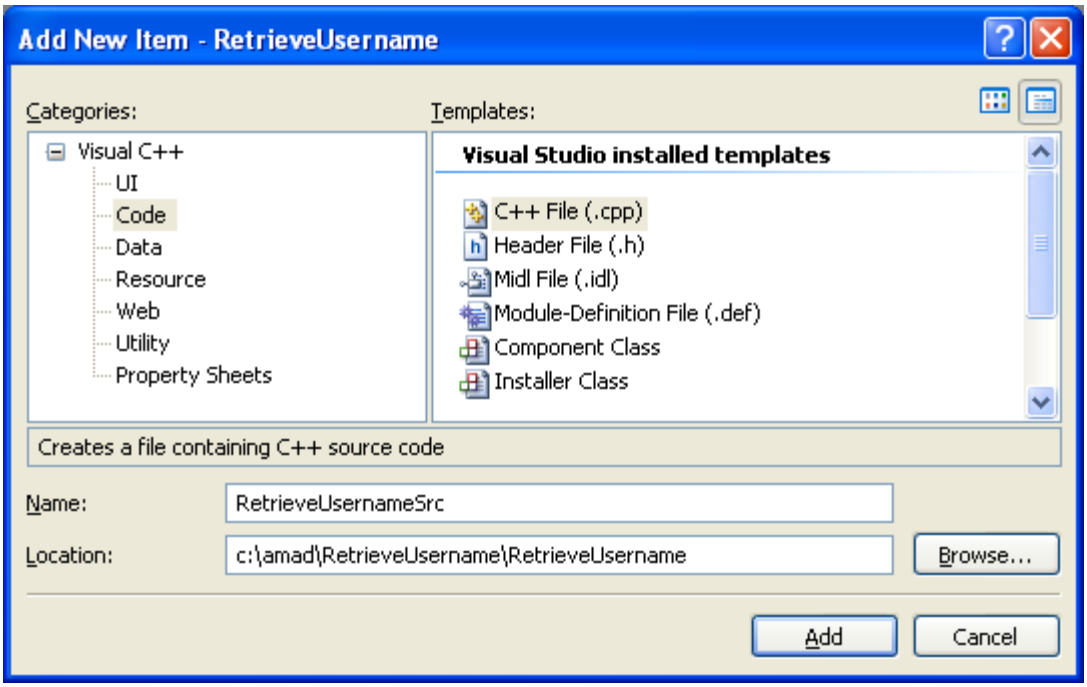
Retrieving current user and domain names on Windows NT, Windows 2000, or Windows XP Code Example

A program may sometimes need to display the user name and domain name for the current thread. Prior to Windows NT, it could be assumed that a thread was running under the account of the interactively logged on user. Windows NT, however, allows threads to run under multiple security contexts, potentially representing multiple users. For instance, in a client/server application, a thread in the server might impersonate a client through the ImpersonateNamedPipeClient() function. In this case, it runs under the user context of the client. Another example of a thread running in a different security context is a service thread, which has a domain name of NT AUTHORITY and a user name of SYSTEM, assuming that the service is running in the local system account.
If you need to obtain both the user name and the domain name for the current thread, you must first extract the user's security identifier (SID) from the thread's access token by using the GetTokenInformation() function. Then, a call to the LookupAccountSid() function can be used to retrieve the account name and domain name associated with the SID. The sample code at the end of this article demonstrates this technique.
The 32-bit functions just mentioned are not available on Microsoft Windows 95 or Microsoft

Windows 98. To retrieve the name and domain of the interactive user on Windows 95 or Windows 98, you must call a 16-bit LAN Manager function.
The following program example demonstrates how to programmatically retrieve the user name and domain name on Windows NT by using security functions within the Win32 Application Programming Interface (API).
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.



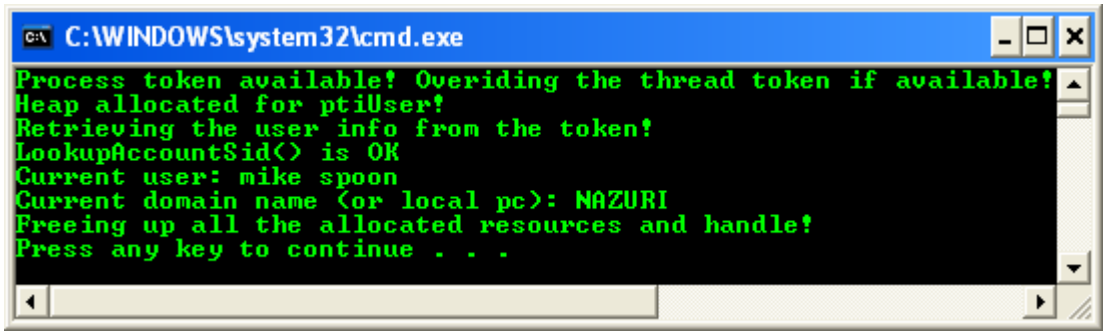
Next, add the following source code.

```
#include <windows.h>
#include <stdio.h>

//*************************************************************
//  FUNCTION:     GetCurrentUserAndDomain - This function looks up
//                the user name and domain name for the user account
//                associated with the calling thread.
//  PARAMETERS:   szUser - a buffer that receives the user name
//                pcchUser - the size, in characters, of szUser
//                szDomain - a buffer that receives the domain name
//                pcchDomain - the size, in characters, of szDomain
//  RETURN VALUE: TRUE if the function succeeds. Otherwise, FALSE and
//                GetLastError() will return the failure reason.
//
//                If either of the supplied buffers are too small,
//                GetLastError() will return ERROR_INSUFFICIENT_BUFFER
//                and pcchUser and pcchDomain will be adjusted to
//                reflect the required buffer sizes.
//*************************************************************
#define MAX_NAME 256

int wmain(int argc, WCHAR **argv)
{
      WCHAR szUser[MAX_NAME];
      WCHAR szDomain[MAX_NAME];
      BOOL          fSuccess = FALSE;
      HANDLE        hToken   = NULL;
      PTOKEN_USER   ptiUser  = NULL;
      DWORD         cbti     = 0;
      SID_NAME_USE snu;
```

```
	__try {
		// Get the calling thread's access token.
		if (!OpenThreadToken(GetCurrentThread(), TOKEN_QUERY, TRUE,
&hToken))
		{
			if (GetLastError() != ERROR_NO_TOKEN)
			{
				wprintf(L"No thread token available!\n");
				__leave;
			}

			// Retry against process token if no thread token exists.
			if (!OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY,
&hToken))
			{
				wprintf(L"No process token available! error %u\n",
GetLastError());
				__leave;
			}
			else
				wprintf(L"Process token available! Overiding the
thread token if available!\n");
		}
		else
			wprintf(L"Thread token available!\n");


		// Obtain the size of the user information in the token.
		if (GetTokenInformation(hToken, TokenUser, NULL, 0, &cbti))
		{
			// Call should have failed due to zero-length buffer.
			wprintf(L"Getting the size of the user info in the
token!\n");
			__leave;
		}
		else
		{
			// Call should have failed due to zero-length buffer.
			if (GetLastError() != ERROR_INSUFFICIENT_BUFFER)
			{
				wprintf(L"Insufficient buffer! Re-allocating...\n");
				__leave;
			}
		}

		// Allocate buffer for user information in the token.
		ptiUser = (PTOKEN_USER) HeapAlloc(GetProcessHeap(), 0, cbti);
		if (!ptiUser)
		{
			wprintf(L"Heap allocation for ptiUser failed, error %u\n",
GetLastError());
			__leave;
		}
		else
			wprintf(L"Heap allocated for ptiUser!\n");
```

```
            // Retrieve the user information from the token.
            if (!GetTokenInformation(hToken, TokenUser, ptiUser, cbti,
&cbti))
            {
                  wprintf(L"GetTokenInformation() failed, error %u\n",
GetLastError());
                  __leave;
            }
            else
                  wprintf(L"Retrieving the user info from the token!\n");

            // Retrieve user name and domain name based on user's SID.
            // The 1st NULL means search the local pc and then domain
controller...
            if (!LookupAccountSid(NULL, ptiUser->User.Sid, szUser, &cbti,
szDomain, &cbti, &snu))
            {
                  wprintf(L"LookupAccountSid() failed, error %u\n",
GetLastError());
                  __leave;
            }
            else
            {
                  wprintf(L"LookupAccountSid() is OK\n");
                  wprintf(L"Current user: %s\n", szUser);
                  wprintf(L"Current domain name (or local pc): %s\n",
szDomain);
            }
            // All should be OK
            fSuccess = TRUE;
      }
      __finally
      {
            // Free resources.
            wprintf(L"Freeing up all the allocated resources and handle!\n");
            if (hToken)
                  CloseHandle(hToken);
            if (ptiUser)
                  HeapFree(GetProcessHeap(), 0, ptiUser);
      }

   return fSuccess;
}
```

Build and run the project. The following screenshot is a sample output.


C:\WINDOWS\system32\cmd.exe
Process token available! Overiding the thread token if available!
Heap allocated for ptiUser!
Retrieving the user info from the token!
LookupAccountSid() is OK
Current user: mike spoon
Current domain name (or local pc): NAZURI
Freeing up all the allocated resources and handle!
Press any key to continue . . .