附錄 B

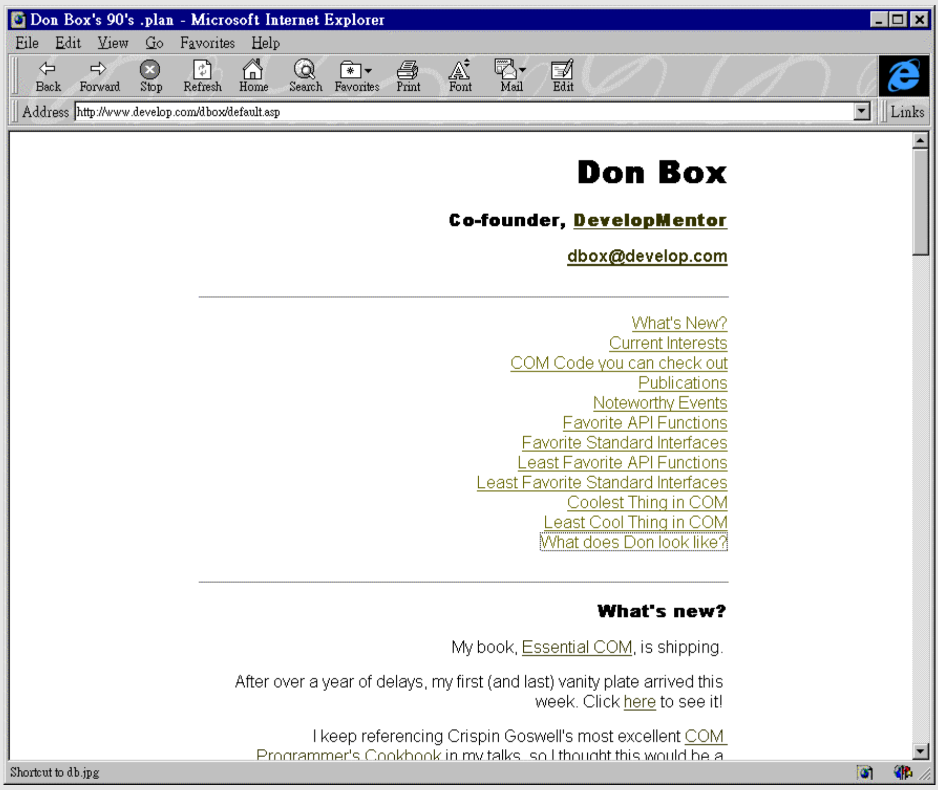
程式碼列表

本書附隨的原始碼中，除了部份是作者設計給自己使用的工具之外，還有一個完整的 COM 應用程式，名爲 COM Chat。所有原始碼都可以從網頁下載而得（http://www.develop.com/dbox/combook/sources）。爲了讓大家方便，我把 COM Chat 程式碼表列出來。

COM Chat：一個建立在 COM 之上的網路聊天程式

COM Chat 是一個完整的 COM-based 程式，用以實作出一個多主題且分散式的閒聊程式。一共有三個二進位軟體元件組成這個應用程式：comchat.exe 是 chat server，comchatps.dll 是所有 COM Chat interface 的 marshaler，而 client.exe 是個文字模式（console-based）的客戶端應用程式。這個應用程式是以單獨一個 COM class（CLSID_ChatSession）爲基礎。如**圖 B-1** 所示，這個 class object 實作出 *IClassSessionManager*，而每一個 chat session 實作出 *IChatSession*。客戶端如果希望收到 chat 通知，就必須提供一個 *IChatSessionEvents* 給 chat session object。

譯註：進入 http://www.develop.com/dbox 網頁中，可看到畫面如下：



Don Box's 90's .plan - Microsoft Internet Explorer

Address http://www.develop.com/dbox/default.asp

Don Box

Co-founder, DevelopMentor

dbox@develop.com

What's New?
Current Interests
COM Code you can check out
Publications
Noteworthy Events
Favorite API Functions
Favorite Standard Interfaces
Least Favorite API Functions
Least Favorite Standard Interfaces
Coolest Thing in COM
Least Cool Thing in COM
What does Don look like?

What's new?

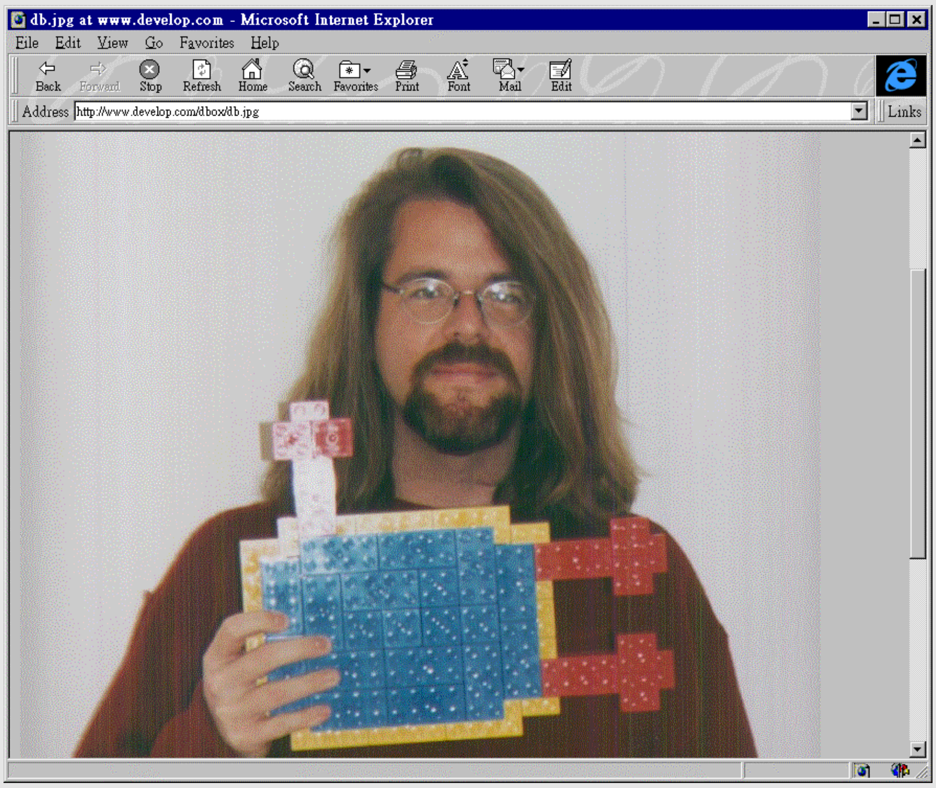
My book, Essential COM, is shipping.

After over a year of delays, my first (and last) vanity plate arrived this week. Click here to see it!

I keep referencing Crispin Goswell's most excellent COM Programmer's Cookbook in my talks, so I thought this would be a

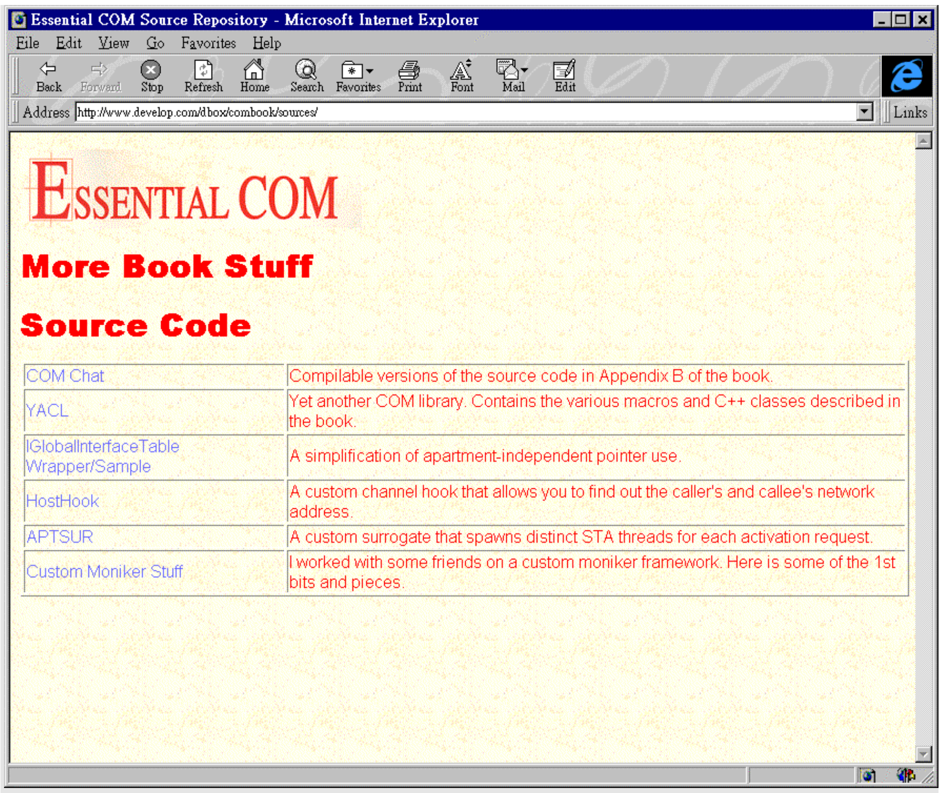
Shortcut to db.jpg

選擇最後一項 "What Does Don look like?"，可看到 Don Box 的尊容：


db.jpg at www.develop.com - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font Mail Edit
Address http://www.develop.com/dbox/db.jpg
Links

難怪 Charlie Kindel 序中劈頭就說『Don 的像片會出現在書的封底嗎？如果是，他的頭髮會有多長？』☺

進入 combook/sources 網頁，可看到畫面如下：


Essential COM Source Repository - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font Mail Edit
Address http://www.develop.com/dbox/combook/sources/
Links

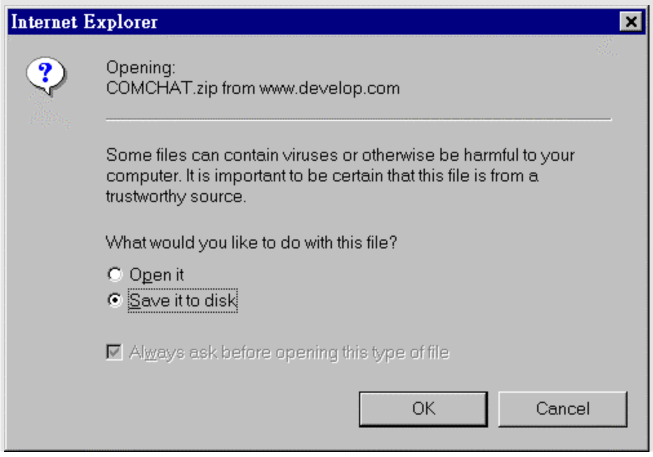
ESSENTIAL COM

More Book Stuff

Source Code

COM Chat	Compilable versions of the source code in Appendix B of the book.
YACL	Yet another COM library. Contains the various macros and C++ classes described in the book.
IGlobalInterfaceTable Wrapper/Sample	A simplification of apartment-independent pointer use.
HostHook	A custom channel hook that allows you to find out the caller's and callee's network address.
APTSUR	A custom surrogate that spawns distinct STA threads for each activation request.
Custom Moniker Stuff	I worked with some friends on a custom moniker framework. Here is some of the 1st bits and pieces.

這就是 Don Box 提供的所有程式。點選任何一個項目，會出現對話盒如下：



Opening:
COMCHAT.zip from www.develop.com

Some files can contain viruses or otherwise be harmful to your computer. It is important to be certain that this file is from a trustworthy source.

What would you like to do with this file?

Open it
[✓ Save it to disk]

[✓] Always ask before opening this type of file

按下【OK】鈕，就可以開始下載。全部下載，可得以下檔案：

```
COMCHAT   ZIP   24661   3-27-98   4:10a
YACL      ZIP   28949   3-27-98   4:12a
GIPLIP    ZIP   21322   3-27-98   4:13a
HOSTHOOK  ZIP   16352   3-27-98   4:14a
APTSUR    ZIP    9612   3-27-98   4:14a
MEOWMO~1  ZIP  111835   3-27-98   4:15a
```

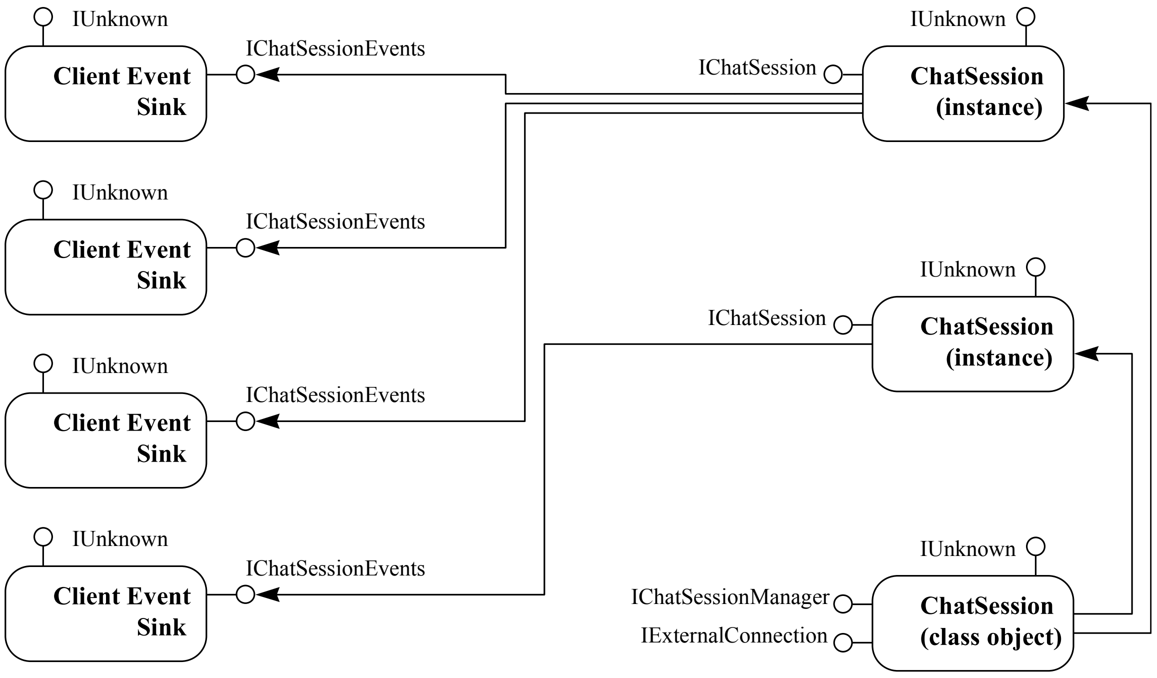


圖 B-1　COM Chat

譯註：以下程式列表是以下載自網頁的檔案爲主，與原書所列的程式碼有十分些微的差異。這些差異都不影響程式的正確性，只是寫碼手法的一些極小變化而已。

comchat.idl

```
/////////////////////////////////////////////////////
//
// COMChat.idl
//
// Copyright 1997, Don Box/Addison Wesley
//
// This code accompanies the book "The Component
// Object Model" from Addison Wesley. Blah blah blah
//
//

interface IChatSessionEvents;

[
 uuid(5223A050-2441-11d1-AF4F-0060976AA886),
 object
]
interface IChatSession : IUnknown
{
 import "objidl.idl";

 [propget] HRESULT SessionName([out, string] OLECHAR **ppwsz);
 HRESULT Say([in, string] const OLECHAR *pwszStatement);
 HRESULT GetStatements([out] IEnumString **ppes);

 HRESULT Advise([in] IChatSessionEvents *pEventSink,
                [out] DWORD *pdwReg);
 HRESULT Unadvise([in] DWORD dwReg);
}

[
 uuid(5223A051-2441-11d1-AF4F-0060976AA886),
 object
]
interface IChatSessionEvents : IUnknown
{
 import "objidl.idl";
 HRESULT OnNewUser([in, string] const OLECHAR *pwszUser);
 HRESULT OnUserLeft([in, string] const OLECHAR *pwszUser);
 HRESULT OnNewStatement([in, string] const OLECHAR *pwszUser,
                        [in, string] const OLECHAR *pwszStmnt);
}

```

```
[
 uuid(5223A052-2441-11d1-AF4F-0060976AA886),
 object
]
interface IChatSessionManager : IUnknown
{
 import "objidl.idl";
 HRESULT GetSessionNames([out] IEnumString **ppes);
 HRESULT FindSession([in, string] const OLECHAR *pwszName,
                     [in] BOOL bDontCreate,
                     [in] BOOL bAllowAnonymousAccess,
                     [out] IChatSession **ppcs);
 HRESULT DeleteSession([in, string] const OLECHAR *pwszName);
}

cpp_quote("DEFINE_GUID(CLSID_ChatSession,0x5223a053,0x2441,")
cpp_quote("0x11d1,0xaf,0x4f,0x0,0x60,0x97,0x6a,0xa8,0x86);")
```

client.cpp

```
/////////////////////////////////////////////////////
//
// client.cpp
//
// Copyright 1997, Don Box/Addison Wesley
//
// This code accompanies the book "The Component
// Object Model" from Addison Wesley. Blah blah blah
//
//

#define _WIN32_WINNT 0x403
#include <windows.h>
#include <stdio.h>
#include <initguid.h>
#include <wchar.h>
#include "../include/COMChat.h"
#include "../include/COMChat_i.c"

void Error(HRESULT hr, const char *psz)
{
    printf("%s failed and returned 0x%x\n", psz, hr);
}

// utility function to print command line syntax
```

```
int Usage(void)
{
  const char *psz =
   "usage: client.exe <action> <user> <host>\n"
   "   where:\n"
   "         action = /sessions|/chat:session|/delete:session\n"
   "         user = /user:domain\\user /password:pw |"
              "/anonymous | <nothing>\n"
   "         host = /host:hostname | <nothing>\n";
  printf(psz);
  return -1;
}

// utility function for printing a list of strings
void PrintAllStrings(IEnumString *pes)
{
    enum { CHUNKSIZE = 64 };
    OLECHAR *rgpwsz[CHUNKSIZE];
    ULONG cFetched;
    HRESULT hr;
    do
    {
        hr = pes->Next(CHUNKSIZE, rgpwsz, &cFetched);
        if (SUCCEEDED(hr))
        {
            for (ULONG i = 0; i < cFetched; i++)
                if (rgpwsz[i])
                {
                    wprintf(L"%s\n", rgpwsz[i]);
                    CoTaskMemFree(rgpwsz[i]);
                }
        }
    } while (hr == S_OK);
}

// utility function to print initial state of
// a chat session
void PrintToDate(IChatSession *pcs)
{
    IEnumString *pes = 0;
    HRESULT hr = pcs->GetStatements(&pes);
    if (SUCCEEDED(hr))
    {
        PrintAllStrings(pes);
        pes->Release();
    }
}

// this class implements the callback interface
// that receives chat notifications. It simply
// prints the event to the console
class EventSink : public IChatSessionEvents
{
public:
   STDMETHODIMP QueryInterface(REFIID riid, void**ppv)
   {
      if (riid == IID_IUnknown)
         *ppv = static_cast<IChatSessionEvents*>(this);
      else if (riid == IID_IChatSessionEvents)
         *ppv = static_cast<IChatSessionEvents*>(this);
      else
         return (*ppv = 0), E_NOINTERFACE;
      reinterpret_cast<IUnknown*>(*ppv)->AddRef();
      return S_OK;
   }
   STDMETHODIMP_(ULONG) AddRef(void)
   {
      return 2;
   }
   STDMETHODIMP_(ULONG) Release(void)
   {
      return 1;
   }
   STDMETHODIMP OnNewStatement(const OLECHAR *pwszUser,
                               const OLECHAR *pwszStmt)
   {
      wprintf(L"%-14s: %s\n", pwszUser, pwszStmt);
      return S_OK;
   }
   STDMETHODIMP OnNewUser(const OLECHAR *pwszUser)
   {
      wprintf(L"\n\n>>> Say Hello to %s\n\n", pwszUser);
      return S_OK;
   }
   STDMETHODIMP OnUserLeft(const OLECHAR *pwszUser)
   {
      wprintf(L"\n\n>>> Say Bye to %s\n\n", pwszUser);
      return S_OK;
   }

};

```

```
// type of operations this client can perform
enum ACTION
{
    ACTION_NONE,
    ACTION_CHAT,
    ACTION_DELETE_SESSION,
    ACTION_LIST_SESSION_NAMES,
};

// run chat command
void Chat(const OLECHAR *pwszSession,
          IChatSessionManager *pcsm, // manager
          COAUTHIDENTITY *pcai,      // user
          bool bAnonymous)           // anonymous
{
// create or get the named session
    IChatSession *pcs = 0;
    HRESULT hr = pcsm->FindSession(pwszSession, FALSE,
                                   TRUE, &pcs);
    if (SUCCEEDED(hr))
    {
// adjust security blanket for session interface
        if (!bAnonymous)
            hr = CoSetProxyBlanket(pcs, RPC_C_AUTHN_WINNT,
                                   RPC_C_AUTHZ_NONE, 0,
                                   RPC_C_AUTHN_LEVEL_PKT,
                                   RPC_C_IMP_LEVEL_IDENTIFY,
                                   pcai, EOAC_NONE);
// catch up on past messages
        PrintToDate(pcs);
// hook up event sink to receive new messages
        EventSink es;
        DWORD dwReg;
        hr = pcs->Advise(&es, &dwReg);
        if (SUCCEEDED(hr))
        {
// run UI loop to get statements from console and send them
            OLECHAR wszStmt[4096];
            while (_getws(wszStmt))
            {
                hr = pcs->Say(wszStmt);
                if (FAILED(hr))
                    Error(hr, "Say");
            }
// tear down connection for event sink
            pcs->Unadvise(dwReg);
        }
        else
           Error(hr, "Advise");
// release chat session
        pcs->Release();
     }
     else
        Error(hr, "FindSession");
}

// run delete command
void Delete(const OLECHAR *pwszSession,
           IChatSessionManager *pcsm)
{
    HRESULT hr = pcsm->DeleteSession(pwszSession);
    if (FAILED(hr))
       Error(hr, "DeleteSession");
}

// run list command
void List(IChatSessionManager *pcsm)
{
    IEnumString *pes = 0;
    HRESULT hr = pcsm->GetSessionNames(&pes);
    if (SUCCEEDED(hr))
    {
       printf("Active Sessions:\n");
       PrintAllStrings(pes);
       pes->Release();
    }
}

int main(int argc, char **argv)
{
// declare client control state
    bool bAnonymous = false;
    static OLECHAR wszSessionName[1024];
    static OLECHAR wszDomainName[1024];
    static OLECHAR wszUserName[1024];
    static OLECHAR wszPassword[1024];
    static OLECHAR wszHostName[1024];
    COSERVERINFO csi = { 0, wszHostName, 0, 0 };
    COSERVERINFO *pcsi = 0;
    COAUTHIDENTITY cai = {
       wszUserName,
       0,
      wszDomainName,
      0,
      wszPassword,
      0,
      SEC_WINNT_AUTH_IDENTITY_UNICODE
   };
   static COAUTHIDENTITY *pcai = 0;
   static ACTION action = ACTION_NONE;

// parse command line
   for (int i = 1; i < argc; i++)
   {
      if (strcmp(argv[i], "/anonymous") == 0)
         bAnonymous = true;
      else if (strstr(argv[i], "/delete:") == argv[i])
      {
         if (action != ACTION_NONE)
            return Usage();
         action = ACTION_DELETE_SESSION;
         mbstowcs(wszSessionName, argv[i] + 8, 1024);
      }
      else if (strstr(argv[i], "/chat:") == argv[i])
      {
         if (action != ACTION_NONE)
            return Usage();
         action = ACTION_CHAT;
         mbstowcs(wszSessionName, argv[i] + 6, 1024);
      }
      else if (strcmp(argv[i], "/sessions") == 0)
      {
         if (action != ACTION_NONE)
            return Usage();
         action = ACTION_LIST_SESSION_NAMES;
      }
      else if (strstr(argv[i], "/host:") == argv[i])
      {
         if (pcsi != 0)
            return Usage();
         mbstowcs(wszHostName, argv[i] + 6, 1024);
         pcsi = &csi;
      }
      else if (strstr(argv[i], "/password:") == argv[i])
      {
         mbstowcs(wszPassword, argv[i] + 10, 1024);
         cai.PasswordLength = wcslen(wszPassword);
      }
```

```
        else if (strstr(argv[i], "/user:") == argv[i])
        {
            if (pcai != 0 || bAnonymous)
                return Usage();
            char *pszDelim = strchr(argv[i] + 7, '\\');
            if (pszDelim == 0)
                return Usage();
            *pszDelim = 0;
            pszDelim++;
            mbstowcs(wszDomainName, argv[i] + 6, 1024);
            cai.DomainLength = wcslen(wszDomainName);
            mbstowcs(wszUserName, pszDelim, 1024);
            cai.UserLength = wcslen(wszUserName);
            pcai = &cai;
        }
    }

    if (action == ACTION_NONE)
        return Usage();
    HRESULT hr = CoInitializeEx(0, COINIT_MULTITHREADED);
    if (FAILED(hr))
        return hr;

// allow anonymous callbacks from chat server
    hr = CoInitializeSecurity(0, -1, 0, 0,
                              RPC_C_AUTHN_LEVEL_NONE,
                              RPC_C_IMP_LEVEL_ANONYMOUS,
                              0, EOAC_NONE, 0);

    if (SUCCEEDED(hr))
    {
// grab the requested session manager
        IChatSessionManager *pcsm = 0;
        hr = CoGetClassObject(CLSID_ChatSession, CLSCTX_ALL,
                              pcsi, IID_IChatSessionManager,
                              (void**)&pcsm);
        if (SUCCEEDED(hr))
        {
// apply security blanket if desired
            if (!bAnonymous)
                hr = CoSetProxyBlanket(pcsm, RPC_C_AUTHN_WINNT,
                                       RPC_C_AUTHZ_NONE, 0,
                                       RPC_C_AUTHN_LEVEL_PKT,
                                       RPC_C_IMP_LEVEL_IDENTIFY,
                                       pcai, EOAC_NONE);
// dispatch request
```

```
            switch (action)
            {
            case ACTION_CHAT:
                Chat(wszSessionName, pcsm, pcai, bAnonymous);
                break;
            case ACTION_DELETE_SESSION:
                Delete(wszSessionName, pcsm);
                break;
            case ACTION_LIST_SESSION_NAMES:
                List(pcsm);
                break;
            default:
                Usage();
            }
// release session manager
            pcsm->Release();
        }
    }
    CoUninitialize();
    return hr;
}
```

chatsession.h

```
/////////////////////////////////////////////////////
//
// ChatSession.h
//
// Copyright 1997, Don Box/Addison Wesley
//
// This code accompanies the book "The Component
// Object Model" from Addison Wesley. Blah blah blah
//
//

#ifndef _CHATSESSION_H
#define _CHATSESSION_H

// this pragma shuts up the compiler warnings due to
// the pre MSC11SP1 debugger choking on long template names.
#pragma warning(disable:4786)

#define _WIN32_WINNT 0x403
#include <windows.h>
```

```
#include <map>
#include <vector>
#include <string>
using namespace std;

// bring in IDL-generated interface definitions
#include "..\include\COMChat.h"

// this class models a particular chat session
class ChatSession : public IChatSession
{
   friend class StatementEnumerator;
   LONG           m_cRef;
   CRITICAL_SECTION   m_csStatementLock;
   CRITICAL_SECTION   m_csAdviseLock;
   OLECHAR            m_wszSessionName[1024];
   bool           m_bIsDeleted;
   bool              m_bAllowAnonymousAccess;
   vector<wstring>    m_statements;
   struct LISTENER
   {
      LISTENER          *pPrev;
      LISTENER          *pNext;
      OLECHAR           *pwszUser;
      IChatSessionEvents *pItf;
   };
   LISTENER          *m_pHeadListeners;
   void SLock(void);
   void SUnlock(void);
   void ALock(void);
   void AUnlock(void);
   bool CheckAccess(const OLECHAR *pwszUser);
protected:
   virtual ~ChatSession(void);
   void Fire_OnNewStatement(const OLECHAR *pwszUser,
                            const OLECHAR *pwszStatement);
   void Fire_OnNewUser(const OLECHAR *pwszUser);
   void Fire_OnUserLeft(const OLECHAR *pwszUser);
public:
   ChatSession(const OLECHAR *pwszSessionName,
             bool bAllowAnonymousAccess);

   void Disconnect(void);
// IUnknown methods
   STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
   STDMETHODIMP_(ULONG) AddRef(void);
```

```
    STDMETHODIMP_(ULONG) Release(void);

// IChatSession methods
    STDMETHODIMP get_SessionName(OLECHAR **ppwsz);
    STDMETHODIMP Say(const OLECHAR *pwszStatement);
    STDMETHODIMP GetStatements(IEnumString **ppes);
    STDMETHODIMP Advise(IChatSessionEvents *pEventSink,
                    DWORD *pdwReg);
    STDMETHODIMP Unadvise(DWORD dwReg);
};

// this class enumerates the statements of a session
class StatementEnumerator : public IEnumString
{
    LONG                        m_cRef;
    ChatSession                 *m_pThis;
    vector<wstring>::iterator  m_cursor;
    CRITICAL_SECTION          m_csLock;
protected:
    void Lock(void);
    void Unlock(void);
    virtual ~StatementEnumerator(void);
public:
    StatementEnumerator(ChatSession *pThis);

// IUnknown methods
    STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
    STDMETHODIMP_(ULONG) AddRef(void);
    STDMETHODIMP_(ULONG) Release(void);

// IEnumString methods
    STDMETHODIMP Next(ULONG cElems, OLECHAR **rgElems,
                   ULONG *pcFetched);
    STDMETHODIMP Skip(ULONG cElems);
    STDMETHODIMP Reset(void);
    STDMETHODIMP Clone(IEnumString **ppes);
};

// this class models the management of chat sessions
// and acts as the class object for CLSID_ChatSession
class ChatSessionClass : public IChatSessionManager,
                     public IExternalConnection
{
    friend class SessionNamesEnumerator;
    typedef map<wstring, ChatSession *> SESSIONMAP;
    LONG                 m_cStrongLocks;
    SESSIONMAP        m_sessions;
    CRITICAL_SECTION   m_csSessionLock;
    void Lock(void);
    void Unlock(void);
    bool CheckAccess(const OLECHAR *pwszUser);
public:
    virtual ~ChatSessionClass(void);
    ChatSessionClass(void);

    // IUnknown methods
    STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
    STDMETHODIMP_(ULONG) AddRef(void);
    STDMETHODIMP_(ULONG) Release(void);

// IExternalConnection methods
    STDMETHODIMP_(DWORD) AddConnection(DWORD extconn, DWORD);
    STDMETHODIMP_(DWORD) ReleaseConnection(DWORD extconn, DWORD,
                                BOOL bLastReleaseKillsStub);
// IChatSessionManager methods
    STDMETHODIMP GetSessionNames(IEnumString **ppes);
    STDMETHODIMP FindSession(const OLECHAR *pwszSessionName,
                         BOOL bDontCreate,
                         BOOL bAllowAnonymousAccess,
                         IChatSession **ppcs);
    STDMETHODIMP DeleteSession(const OLECHAR *pwszSessionName);
};

// this class enumerates the session names of a server
class SessionNamesEnumerator : public IEnumString
{
    LONG                    m_cRef;
    vector<wstring>          *m_pStrings;
    SessionNamesEnumerator    *m_pCloneSource;
    vector<wstring>::iterator  m_cursor;
    CRITICAL_SECTION          m_csLock;
protected:
    vector<wstring>& Strings(void);
    void Lock(void);
    void Unlock(void);
    virtual ~SessionNamesEnumerator(void);
public:
    SessionNamesEnumerator(ChatSessionClass *pSessionClass);
    SessionNamesEnumerator(SessionNamesEnumerator *pCloneSource);

// IUnknown methods
    STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
    STDMETHODIMP_(ULONG) AddRef(void);
    STDMETHODIMP_(ULONG) Release(void);

// IEnumString methods
    STDMETHODIMP Next(ULONG cElems, OLECHAR **rgElems,
                      ULONG *pcFetched);
    STDMETHODIMP Skip(ULONG cElems);
    STDMETHODIMP Reset(void);
    STDMETHODIMP Clone(IEnumString **ppes);
};

#endif
```

chatsession.cpp

```
/////////////////////////////////////////////////////
//
// ChatSession.cpp
//
// Copyright 1997, Don Box/Addison Wesley
//
// This code accompanies the book "The Component
// Object Model" from Addison Wesley. Blah blah blah
//
//

#include "ChatSession.h"
#include <iaccess.h>

// these routines are defined in svc.cpp to
// control server lifetime
extern void ModuleLock(void);
extern void ModuleUnlock(void);

// these access control objects are created
// in svc.cpp to control various privileged
// operations. Most operations in this class
// are non-privileged, so anyone can get in.
extern IAccessControl *g_pacUsers;
extern IAccessControl *g_pacAdmins;

// utility functions ////////////////////////

// duplicate an OLECHAR * using CoTaskMemAlloc
```

```
OLECHAR *OLESTRDUP(const OLECHAR *pwsz)
{
   DWORD cb = sizeof(OLECHAR)*(wcslen(pwsz) + 1);
   OLECHAR *pwszResult = (OLECHAR*)CoTaskMemAlloc(cb);
   if (pwszResult)
      wcscpy(pwszResult, pwsz);
   return pwszResult;
}

// get the caller's username (or "anonymous" if
// no authentication was specified by the caller).
OLECHAR *GetCaller(void)
{
   OLECHAR *pwsz = 0;
   HRESULT hr = CoQueryClientBlanket(0,0,0,0,0,(void**)&pwsz,0);
   if (SUCCEEDED(hr))
      return OLESTRDUP(pwsz);
   else
      return OLESTRDUP(OLESTR("anonymous"));
}

// class ChatSession ///////////////////////////////

ChatSession::ChatSession(const OLECHAR *pwszSessionName,
                     bool bAllowAnonymousAccess)
: m_cRef(0),
  m_bAllowAnonymousAccess(bAllowAnonymousAccess),
  m_pHeadListeners(0)
{
   wcscpy(m_wszSessionName, pwszSessionName);
   InitializeCriticalSection(&m_csStatementLock);
   InitializeCriticalSection(&m_csAdviseLock);
}

ChatSession::~ChatSession(void)
{
   DeleteCriticalSection(&m_csStatementLock);
   DeleteCriticalSection(&m_csAdviseLock);
// tear down connected listeners
   while (m_pHeadListeners)
   {
      LISTENER *pThisNode = m_pHeadListeners;
      if (pThisNode->pItf)
         pThisNode->pItf->Release();
      if (pThisNode->pwszUser)
         CoTaskMemFree(pThisNode->pwszUser);
      m_pHeadListeners = pThisNode->pNext;
      delete pThisNode;
   }
}

// helper methods //////////

void ChatSession::Disconnect(void)
{
   CoDisconnectObject(this, 0);
// tear down connected listeners
   ALock();
   while (m_pHeadListeners)
   {
      LISTENER *pThisNode = m_pHeadListeners;
      if (pThisNode->pItf)
         pThisNode->pItf->Release();
      if (pThisNode->pwszUser)
         CoTaskMemFree(pThisNode->pwszUser);
      m_pHeadListeners = pThisNode->pNext;
      delete pThisNode;
   }
   AUnlock();
}

// send the OnNewStatement event to all listeners
void
ChatSession::Fire_OnNewStatement(const OLECHAR *pwszUser,
                            const OLECHAR *pwszStatement)
{
   ALock();
   for (LISTENER *pNode = m_pHeadListeners;
        pNode != 0; pNode = pNode->pNext)
   {
      if (pNode->pItf)
         pNode->pItf->OnNewStatement(pwszUser, pwszStatement);
   }
   AUnlock();
}

// send the OnNewUser event to all listeners
void
ChatSession::Fire_OnNewUser(const OLECHAR *pwszUser)
{
   ALock();
   for (LISTENER *pNode = m_pHeadListeners;
        pNode != 0; pNode = pNode->pNext)
    {
        if (pNode->pItf)
            pNode->pItf->OnNewUser(pwszUser);
    }
    AUnlock();
}

// send the OnUserLeft event to all listeners
void
ChatSession::Fire_OnUserLeft(const OLECHAR *pwszUser)
{
    ALock();
    for (LISTENER *pNode = m_pHeadListeners;
         pNode != 0; pNode = pNode->pNext)
    {
        if (pNode->pItf)
            pNode->pItf->OnUserLeft(pwszUser);
    }
    AUnlock();
}

// lock wrappers
void ChatSession::SLock(void)
{
    EnterCriticalSection(&m_csStatementLock);
}

void ChatSession::SUnlock(void)
{
    LeaveCriticalSection(&m_csStatementLock);
}

void ChatSession::ALock(void)
{
    EnterCriticalSection(&m_csAdviseLock);
}

void ChatSession::AUnlock(void)
{
    LeaveCriticalSection(&m_csAdviseLock);
}

// helper method to check access to Say method
bool
ChatSession::CheckAccess(const OLECHAR *pwszUser)
```

```
{
    if (wcscmp(pwszUser, L"anonymous") == 0)
        return m_bAllowAnonymousAccess;
// form trustee from caller and use Access Control
// object hardwired to COMChat Users group
    TRUSTEEW trustee = {
        0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
        TRUSTEE_IS_USER,
        const_cast<OLECHAR*>(pwszUser)
    };
    BOOL bIsAllowed;
    HRESULT hr = g_pacUsers->IsAccessAllowed(&trustee,0,
                                          COM_RIGHTS_EXECUTE,
                                          &bIsAllowed);
    return SUCCEEDED(hr) && bIsAllowed != FALSE;
}

// IUnknown methods
STDMETHODIMP
ChatSession::QueryInterface(REFIID riid, void **ppv)
{
    if (riid == IID_IUnknown)
        *ppv = static_cast<IChatSession*>(this);
    else if (riid == IID_IChatSession)
        *ppv = static_cast<IChatSession*>(this);
    else
        return (*ppv = 0), E_NOINTERFACE;
    reinterpret_cast<IUnknown*>(*ppv)->AddRef();
    return S_OK;

}

STDMETHODIMP_(ULONG)
ChatSession::AddRef(void)
{
    ModuleLock();
    return InterlockedIncrement(&m_cRef);
}

STDMETHODIMP_(ULONG)
ChatSession::Release(void)
{
    LONG res = InterlockedDecrement(&m_cRef);
    if (res == 0)
        delete this;
    ModuleUnlock();
```

```
   return res;
}

// IChatSession methods
STDMETHODIMP
ChatSession::get_SessionName(OLECHAR **ppwsz)
{
   if (!ppwsz)
      return E_INVALIDARG;
   else if ((*ppwsz = OLESTRDUP(m_wszSessionName)) == 0)
      return E_OUTOFMEMORY;
   return S_OK;
}

STDMETHODIMP
ChatSession::Say(const OLECHAR *pwszStatement)
{
   HRESULT hr = S_OK;
// protect access to method
   OLECHAR *pwszUser = GetCaller();
   if (pwszUser && CheckAccess(pwszUser))
   {
      SLock();
      try
      {
         wstring s = pwszUser;
         s += L":";
         s += pwszStatement;
         m_statements.push_back(s);
      }
      catch(...)
      {
         hr = E_OUTOFMEMORY;
      }
      SUnlock();
      if (SUCCEEDED(hr))
         Fire_OnNewStatement(pwszUser, pwszStatement);
   }
   else
      hr = E_ACCESSDENIED;
   CoTaskMemFree(pwszUser);
   return hr;
}

STDMETHODIMP
ChatSession::GetStatements(IEnumString **ppes)
```

```
{
   if (ppes == 0)
      return E_INVALIDARG;
   *ppes = new StatementEnumerator(this);
   if (*ppes == 0)
      return E_OUTOFMEMORY;
   (*ppes)->AddRef();
   return S_OK;
}

STDMETHODIMP
ChatSession::Advise(IChatSessionEvents *pEventSink,
                 DWORD *pdwReg)
{
   HRESULT hr = S_OK;
   if (pEventSink == 0 || pdwReg == 0)
      return E_INVALIDARG;
   LISTENER *pNew = new LISTENER;
   if (pNew == 0)
      return E_OUTOFMEMORY;
   OLECHAR *pwszUser = GetCaller();
   if (pwszUser)
   {
      Fire_OnNewUser(pwszUser);
      ALock();
      pNew->pwszUser = pwszUser;
      if (pNew->pItf = pEventSink)
         pEventSink->AddRef();
      pNew->pNext = m_pHeadListeners;
      if (m_pHeadListeners)
         m_pHeadListeners->pPrev = pNew;
      pNew->pPrev = 0;
      m_pHeadListeners = pNew;
      AUnlock();
   }
   else
   {
      delete pNew;
      return E_OUTOFMEMORY;
   }
   *pdwReg = reinterpret_cast<DWORD>(pNew);
   return hr;
}

STDMETHODIMP
ChatSession::Unadvise(DWORD dwReg)
```

```
{
   if (dwReg == 0)
      return E_INVALIDARG;
   HRESULT hr = S_OK;
   LISTENER *pThisNode = reinterpret_cast<LISTENER *>(dwReg);
   ALock();
   if (pThisNode->pPrev)
      pThisNode->pPrev->pNext = pThisNode->pNext;
   else
      m_pHeadListeners = pThisNode->pNext;
   if (pThisNode->pNext)
      pThisNode->pNext->pPrev = pThisNode->pPrev;
   if (pThisNode->pItf)
      pThisNode->pItf->Release();
   OLECHAR *pwszUser = pThisNode->pwszUser;
   delete pThisNode;
   AUnlock();
   Fire_OnUserLeft(pwszUser);
   CoTaskMemFree(pwszUser);
   return hr;
}

// class StatementEnumerator ///////////////////

StatementEnumerator::StatementEnumerator(ChatSession *pThis)
: m_cRef(0),
  m_pThis(pThis),
  m_cursor(pThis->m_statements.begin())
{
   m_pThis->AddRef();
   InitializeCriticalSection(&m_csLock);
}

StatementEnumerator::~StatementEnumerator(void)
{
   m_pThis->Release();
   DeleteCriticalSection(&m_csLock);
}

// lock helpers (note that ChatSession is locked
// simultaneously)
void
StatementEnumerator::Lock(void)
{
   EnterCriticalSection(&m_csLock);
   m_pThis->SLock();
```

```
}

void
StatementEnumerator::Unlock(void)
{
   LeaveCriticalSection(&m_csLock);
   m_pThis->SUnlock();
}

// IUnknown methods
STDMETHODIMP
StatementEnumerator::QueryInterface(REFIID riid, void **ppv)
{
   if (riid == IID_IUnknown)
      *ppv = static_cast<IEnumString*>(this);
   else if (riid == IID_IEnumString)
      *ppv = static_cast<IEnumString*>(this);
   else
      return (*ppv = 0), E_NOINTERFACE;
   reinterpret_cast<IUnknown*>(*ppv)->AddRef();
   return S_OK;

}

STDMETHODIMP_(ULONG)
StatementEnumerator::AddRef(void)
{
   return InterlockedIncrement(&m_cRef);
}

STDMETHODIMP_(ULONG)
StatementEnumerator::Release(void)
{
   LONG res = InterlockedDecrement(&m_cRef);
   if (res == 0)
      delete this;
   return res;
}

// IEnumString methods
STDMETHODIMP
StatementEnumerator::Next(ULONG cElems, OLECHAR **rgElems,
                          ULONG *pcFetched)
{
   if (pcFetched == 0 && cElems > 1)
      return E_INVALIDARG;
```

```
    ZeroMemory(rgElems, sizeof(OLECHAR*) * cElems);
    Lock();
    ULONG cActual = 0;
    while (cActual < cElems
          && m_cursor != m_pThis->m_statements.end())
    {
       if (rgElems[cActual] = OLESTRDUP((*m_cursor).c_str()))
       {
          m_cursor++;
          cActual++;
       }
       else // allocation error, unwind
       {
          while (cActual > 0)
          {
             cActual--;
             CoTaskMemFree(rgElems[cActual]);
             rgElems[cActual] = 0;
          }
          break;
       }
    }
    Unlock();
    if (pcFetched)
       *pcFetched = cActual;
    return cElems == cActual ? S_OK : S_FALSE;
}

STDMETHODIMP
StatementEnumerator::Skip(ULONG cElems)
{
    Lock();
    ULONG cActual = 0;
    while (cActual < cElems
          && m_cursor != m_pThis->m_statements.end())
    {
       m_cursor++;
       cActual++;
    }
    Unlock();
    return cElems == cActual ? S_OK : S_FALSE;
}

STDMETHODIMP
StatementEnumerator::Reset(void)
{
    Lock();
    m_cursor = m_pThis->m_statements.begin();
    Unlock();
    return S_OK;
}

STDMETHODIMP
StatementEnumerator::Clone(IEnumString **ppes)
{
    if (ppes == 0)
       return E_INVALIDARG;
    if (*ppes = new StatementEnumerator(m_pThis))
       return S_OK;
    return E_OUTOFMEMORY;
}

// class ChatSessionClass ////////////////////

ChatSessionClass::ChatSessionClass(void)
: m_cStrongLocks(0)
{
    InitializeCriticalSection(&m_csSessionLock);
}

ChatSessionClass::~ChatSessionClass(void)
{
    DeleteCriticalSection(&m_csSessionLock);
}

void
ChatSessionClass::Lock(void)
{
    EnterCriticalSection(&m_csSessionLock);
}

void
ChatSessionClass::Unlock(void)
{
    LeaveCriticalSection(&m_csSessionLock);
}
// helper method to protect access to DeleteSession
// to only allow COMChat Admins to delete groups
bool
ChatSessionClass::CheckAccess(const OLECHAR *pwszUser)
{
    if (wcscmp(pwszUser, L"anonymous") == 0)
```

```
            return false;

        TRUSTEEW trustee = {
            0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
            TRUSTEE_IS_USER, const_cast<OLECHAR*>(pwszUser)
        };
        BOOL bIsAllowed;
        HRESULT hr = g_pacAdmins->IsAccessAllowed(&trustee,0,
                                            COM_RIGHTS_EXECUTE,
                                            &bIsAllowed);
        if (FAILED(hr))
            bIsAllowed = false;
        return SUCCEEDED(hr) && bIsAllowed != FALSE;
    }


    // IUnknown methods
    STDMETHODIMP
    ChatSessionClass::QueryInterface(REFIID riid, void **ppv)
    {
        if (riid == IID_IUnknown)
            *ppv = static_cast<IChatSessionManager*>(this);
        else if (riid == IID_IChatSessionManager)
            *ppv = static_cast<IChatSessionManager*>(this);
        else if (riid == IID_IExternalConnection)
            *ppv = static_cast<IExternalConnection*>(this);
        else
            return (*ppv = 0), E_NOINTERFACE;
        reinterpret_cast<IUnknown*>(*ppv)->AddRef();
        return S_OK;
    }

    STDMETHODIMP_(ULONG)
    ChatSessionClass::AddRef(void)
    {
        return 2;
    }

    STDMETHODIMP_(ULONG)
    ChatSessionClass::Release(void)
    {
        return 1;
    }

    // IExternalConnection methods
    STDMETHODIMP_(DWORD)
```

```
ChatSessionClass::AddConnection(DWORD extconn, DWORD)
{
   if (extconn & EXTCONN_STRONG)
   {
      ModuleLock();
      return InterlockedIncrement(&m_cStrongLocks);
   }
   return 0;
}

STDMETHODIMP_(DWORD)
ChatSessionClass::ReleaseConnection(DWORD extconn, DWORD,
                               BOOL bLastReleaseKillsStub)
{
   if (extconn & EXTCONN_STRONG)
   {
      LONG res = InterlockedDecrement(&m_cStrongLocks);
      if (res == 0 && bLastReleaseKillsStub)
         CoDisconnectObject(
            static_cast<IExternalConnection*>(this), 0);
      ModuleUnlock();
      return res;
   }
   return 0;
}

// IChatSessionManager methods
STDMETHODIMP
ChatSessionClass::GetSessionNames(IEnumString **ppes)
{
   if (ppes == 0)
      return E_INVALIDARG;
   if (*ppes = new SessionNamesEnumerator(this))
   {
      (*ppes)->AddRef();
      return S_OK;
   }
   else
      return E_OUTOFMEMORY;
}

STDMETHODIMP
ChatSessionClass::FindSession(const OLECHAR *pwszSessionName,
                          BOOL bDontCreate,
                          BOOL bAllowAnonymousAccess,
                          IChatSession **ppcs)
```

```
{
   if (ppcs == 0)
      return E_INVALIDARG;
   HRESULT hr = E_FAIL;
   *ppcs = 0;
   OLECHAR *pwszUser = GetCaller();
   Lock();
   SESSIONMAP::iterator it = m_sessions.find(pwszSessionName);
   if (it == m_sessions.end())
   {
      if (bDontCreate)
         hr = E_FAIL;
      else if (!bAllowAnonymousAccess
             && wcscmp(pwszUser, L"anonymous") == 0)
         hr = E_ACCESSDENIED;
      else
      {
         ChatSession *pNew =
            new ChatSession(pwszSessionName,
                        bAllowAnonymousAccess != FALSE);
         if (pNew)
         {
            pNew->AddRef();
            m_sessions.insert(
               pair<wstring,
                   ChatSession*>(pwszSessionName,
                             pNew));
            (*ppcs = pNew)->AddRef();
            hr = S_OK;
         }
         else
            hr = E_OUTOFMEMORY;
      }
   }
   else
   {
      (*ppcs = (*it).second)->AddRef();
      hr = S_OK;
   }
   Unlock();
   CoTaskMemFree(pwszUser);
   return hr;
}

STDMETHODIMP
ChatSessionClass::DeleteSession(const OLECHAR *pwszSessionName)
```

```
{
   if (pwszSessionName == 0)
      return E_INVALIDARG;
   HRESULT hr = E_FAIL;
   OLECHAR *pwszUser = GetCaller();
   if (CheckAccess(pwszUser))
   {
      Lock();
      SESSIONMAP::iterator it
                     = m_sessions.find(pwszSessionName);
      if (it == m_sessions.end())
      {
         hr = E_FAIL;
      }
      else
      {
         (*it).second->Disconnect();
         (*it).second->Release();
         m_sessions.erase(it);
         hr = S_OK;
      }
      Unlock();
   }
   else
      hr = E_ACCESSDENIED;
   CoTaskMemFree(pwszUser);
   return hr;
}

// class SessionNamesEnumerator

vector<wstring>&
SessionNamesEnumerator::Strings(void)
{
   if (m_pStrings)
      return *m_pStrings;
   else
      return *(m_pCloneSource->m_pStrings);
}

void
SessionNamesEnumerator::Lock(void)
{
   EnterCriticalSection(&m_csLock);
}

```

```
void
SessionNamesEnumerator::Unlock(void)
{
    LeaveCriticalSection(&m_csLock);
}

SessionNamesEnumerator::SessionNamesEnumerator(
                             ChatSessionClass *pSessionClass)
: m_cRef(0),
  m_pStrings(0),
  m_pCloneSource(0)
{
    typedef ChatSessionClass::SESSIONMAP::iterator iterator;
    ChatSessionClass::SESSIONMAP &sessions
        = pSessionClass->m_sessions;
    m_pStrings = new vector<wstring>;
    pSessionClass->Lock();
    for (iterator it = sessions.begin();
         it != sessions.end();
         it++)
    {
        m_pStrings->push_back((*it).first);
    }
    pSessionClass->Unlock();
    m_cursor = Strings().begin();
    InitializeCriticalSection(&m_csLock);
}

SessionNamesEnumerator::SessionNamesEnumerator(
                     SessionNamesEnumerator *pCloneSource)
: m_cRef(0),
  m_pStrings(0),
  m_pCloneSource(pCloneSource)
{
    m_pCloneSource->AddRef();
    m_cursor = Strings().begin();
    InitializeCriticalSection(&m_csLock);
}

SessionNamesEnumerator::~SessionNamesEnumerator(void)
{
    if (m_pCloneSource)
        m_pCloneSource->Release();
    else if (m_pStrings)
        delete m_pStrings;
    DeleteCriticalSection(&m_csLock);
```

```
}

// IUnknown methods

STDMETHODIMP
SessionNamesEnumerator::QueryInterface(REFIID riid, void **ppv)
{
   if (riid == IID_IUnknown)
      *ppv = static_cast<IEnumString*>(this);
   else if (riid == IID_IEnumString)
      *ppv = static_cast<IEnumString*>(this);
   else
      return (*ppv = 0), E_NOINTERFACE;
   reinterpret_cast<IUnknown*>(*ppv)->AddRef();
   return S_OK;
}

STDMETHODIMP_(ULONG)
SessionNamesEnumerator::AddRef(void)
{
   ModuleLock();
   return InterlockedIncrement(&m_cRef);
}

STDMETHODIMP_(ULONG)
SessionNamesEnumerator::Release(void)
{
   LONG res = InterlockedDecrement(&m_cRef);
   if (res == 0)
      delete this;
   ModuleUnlock();
   return res;
}

// IEnumString methods
STDMETHODIMP
SessionNamesEnumerator::Next(ULONG cElems, OLECHAR **rgElems,
                         ULONG *pcFetched)
{
   if (cElems > 1 && pcFetched == 0)
      return E_INVALIDARG;
   ULONG cActual = 0;
   vector<wstring> &rstrings = Strings();
   Lock();
   while (cActual < cElems
         && m_cursor != rstrings.end())
```

```
    {
        if (rgElems[cActual] = OLESTRDUP((*m_cursor).c_str()))
        {
            m_cursor++;
            cActual++;
        }
        else // allocation error, unwind
        {
            while (cActual > 0)
            {
                cActual--;
                CoTaskMemFree(rgElems[cActual]);
                rgElems[cActual] = 0;
            }
            break;
        }
    }
    Unlock();
    if (cActual)
        *pcFetched = cActual;
    return cActual == cElems ? S_OK : S_FALSE;
}

STDMETHODIMP
SessionNamesEnumerator::Skip(ULONG cElems)
{
    ULONG cActual = 0;
    vector<wstring> &rstrings = Strings();
    Lock();
    while (cActual < cElems
           && m_cursor != rstrings.end())
    {
        m_cursor++;
        cActual++;
    }
    Unlock();
    return cActual == cElems ? S_OK : S_FALSE;
}

STDMETHODIMP
SessionNamesEnumerator::Reset(void)
{
    Lock();
    m_cursor = Strings().begin();
    Unlock();
    return S_OK;
```

```
}

STDMETHODIMP
SessionNamesEnumerator::Clone(IEnumString **ppes)
{
    if (ppes == 0)
       return E_INVALIDARG;
    SessionNamesEnumerator *pCloneSource = m_pCloneSource;
    if (pCloneSource == 0) // we are the source
       m_pCloneSource = this;
    *ppes = new SessionNamesEnumerator(pCloneSource);
    if (*ppes)
    {
       (*ppes)->AddRef();
       return S_OK;
    }
    return E_OUTOFMEMORY;
}
```

svc.cpp

```
/////////////////////////////////////////////////////
//
// svc.cpp
//
// Copyright 1997, Don Box/Addison Wesley
//
// This code accompanies the book "The Component
// Object Model" from Addison Wesley. Blah blah blah
//
//

#define _WIN32_WINNT 0x403
#include <windows.h>
#include <olectl.h>
#include <initguid.h>
#include <iaccess.h>

#include "ChatSession.h"
#include "../include/COMChat_i.c"


#if !defined(HAVE_IID_IACCESSCONTROL)
// there is a common bug is the SDK headers and libs
// that causes IID_IAccessControl to be undefined.
// We define it here to give the GUID linkage.
DEFINE_GUID(IID_IAccessControl,0xEEDD23E0, 0x8410, 0x11CE,
            0xA1, 0xC3, 0x08, 0x00, 0x2B, 0x2B, 0x8D, 0x8F);
#endif

// standard MTA lifetime management helpers
HANDLE g_heventDone = CreateEvent(0, TRUE, FALSE, 0);

void ModuleLock(void)
{
    CoAddRefServerProcess();
}

void ModuleUnlock(void)
{
    if (CoReleaseServerProcess() == 0)
        SetEvent(g_heventDone);
}

// standard self-registration table
const char *g_RegTable[][3] = {
{ "CLSID\\{5223A053-2441-11d1-AF4F-0060976AA886}",
  0, "ChatSession" },
{ "CLSID\\{5223A053-2441-11d1-AF4F-0060976AA886}",
  "AppId", "{5223A054-2441-11d1-AF4F-0060976AA886}"
},
{ "CLSID\\{5223A053-2441-11d1-AF4F-0060976AA886}\\LocalServer32",
  0, (const char*)-1 // rogue value indicating file name
},
{ "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
  0, "ChatSession Server" },
{ "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
  "RunAs", "Domain\\ReplaceMe"
},
{ "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
  "Chat Admins Group", "Domain\\ReplaceMe"
},
{ "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
  "Chat Users Group", "Domain\\ReplaceMe"
},
{ "AppID\\COMChat.exe",
  "AppId", "{5223A054-2441-11d1-AF4F-0060976AA886}"
},
};

```

```
// self-unregistration routine
STDAPI UnregisterServer(void) {
  HRESULT hr = S_OK;
  int nEntries = sizeof(g_RegTable)/sizeof(*g_RegTable);
  for (int i = nEntries - 1; i >= 0; i--){
    const char *pszKeyName = g_RegTable[i][0];

    long err = RegDeleteKeyA(HKEY_CLASSES_ROOT, pszKeyName);
    if (err != ERROR_SUCCESS)
      hr = S_FALSE;
  }
  return hr;
}

// self-registration routine
STDAPI RegisterServer(HINSTANCE hInstance = 0) {
  HRESULT hr = S_OK;
// look up server's file name
  char szFileName[MAX_PATH];
  GetModuleFileNameA(hInstance, szFileName, MAX_PATH);
// register entries from table
  int nEntries = sizeof(g_RegTable)/sizeof(*g_RegTable);
  for (int i = 0; SUCCEEDED(hr) && i < nEntries; i++) {
    const char *pszKeyName   = g_RegTable[i][0];
    const char *pszValueName = g_RegTable[i][1];
    const char *pszValue     = g_RegTable[i][2];
// map rogue value to module file name
    if (pszValue == (const char*)-1)
      pszValue = szFileName;
    HKEY hkey;
// create the key
    long err = RegCreateKeyA(HKEY_CLASSES_ROOT,
                        pszKeyName, &hkey);
    if (err == ERROR_SUCCESS) {
// set the value
      err = RegSetValueExA(hkey, pszValueName, 0,
                       REG_SZ, (const BYTE*)pszValue,
                       (strlen(pszValue) + 1));
      RegCloseKey(hkey);
    }
    if (err != ERROR_SUCCESS) {
  // if cannot add key or value, back out and fail
      UnregisterServer();
      hr = SELFREG_E_CLASS;
    }
  }
  return hr;
}

// these point to standard access control objects
// used to protect particular methods
IAccessControl *g_pacUsers = 0;
IAccessControl *g_pacAdmins = 0;

// this routine is called at process init time
// to build access control objects and to allow
// anonymous access to server by default
HRESULT InitializeApplicationSecurity(void)
{
// load groupnames from registry
    static OLECHAR wszAdminsGroup[1024];
    static OLECHAR wszUsersGroup[1024];
    HKEY hkey;
    long err = RegOpenKeyEx(HKEY_CLASSES_ROOT,
        __TEXT("AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}"),
                        0, KEY_QUERY_VALUE,
                        &hkey);
    if (err == ERROR_SUCCESS)
    {
       DWORD cb = sizeof(wszAdminsGroup);
       err = RegQueryValueExW(hkey, L"Chat Admins Group",
                          0, 0, (BYTE*)wszAdminsGroup,
                          &cb);
       cb = sizeof(wszAdminsGroup);
       if (err == ERROR_SUCCESS)
          err = RegQueryValueExW(hkey,
                              L"Chat Users Group",
                              0, 0, (BYTE*)wszUsersGroup,
                              &cb);
       RegCloseKey(hkey);
    }
    if (err != ERROR_SUCCESS)
       return MAKE_HRESULT(SEVERITY_ERROR, FACILITY_WIN32,
                        GetLastError());

// declare vectors of user/groups for 2 access
// control objects

    ACTRL_ACCESS_ENTRYW rgaaeUsers[] = {
      { {0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
       TRUSTEE_IS_GROUP, wszUsersGroup },
       ACTRL_ACCESS_ALLOWED, COM_RIGHTS_EXECUTE, 0,
   NO_INHERITANCE, 0 },
};
ACTRL_ACCESS_ENTRY_LISTW aaelUsers = {
   sizeof(rgaaeUsers)/sizeof(*rgaaeUsers),
   rgaaeUsers
};
ACTRL_PROPERTY_ENTRYW apeUsers = { 0, &aaelUsers, 0 };
ACTRL_ACCESSW aaUsers = { 1, &apeUsers };

ACTRL_ACCESS_ENTRYW rgaaeAdmins[] = {
  { {0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
   TRUSTEE_IS_GROUP, wszAdminsGroup },
   ACTRL_ACCESS_ALLOWED, COM_RIGHTS_EXECUTE, 0,
   NO_INHERITANCE, 0 },
};
ACTRL_ACCESS_ENTRY_LISTW aaelAdmins = {
   sizeof(rgaaeAdmins)/sizeof(*rgaaeAdmins),
   rgaaeAdmins
};
ACTRL_PROPERTY_ENTRYW apeAdmins = { 0, &aaelAdmins, 0 };
ACTRL_ACCESSW aaAdmins = { 1, &apeAdmins };

HRESULT hr = CoInitializeSecurity(0, -1, 0, 0,
                      RPC_C_AUTHN_LEVEL_NONE,
                      RPC_C_IMP_LEVEL_ANONYMOUS,
                      0,
                      EOAC_NONE,
                      0);
if (SUCCEEDED(hr))
{
   hr = CoCreateInstance(CLSID_DCOMAccessControl,
                      0, CLSCTX_ALL, IID_IAccessControl,
                      (void**)&g_pacUsers);
   if (SUCCEEDED(hr))
      hr = g_pacUsers->SetAccessRights(&aaUsers);
   if (SUCCEEDED(hr))
   {
      hr = CoCreateInstance(CLSID_DCOMAccessControl,
                         0, CLSCTX_ALL,
                         IID_IAccessControl,
                         (void**)&g_pacAdmins);
      if (SUCCEEDED(hr))
         hr = g_pacAdmins->SetAccessRights(&aaAdmins);
   }
   if (FAILED(hr))
   {
```

```
            if (g_pacAdmins)
            {
                g_pacAdmins->Release();
                g_pacAdmins = 0;
            }
            if (g_pacUsers)
            {
                g_pacUsers->Release();
                g_pacUsers = 0;
            }
        }
    }
    return hr;
}

// the main thread routine that simply registers the class
// object and waits to die
int WINAPI WinMain(HINSTANCE, HINSTANCE,
                LPSTR szCmdParam, int)
{
    const TCHAR *pszPrompt =
        __TEXT("Ensure that you have properly ")
        __TEXT("configured the application to ")
        __TEXT("run as a particular user and that ")
        __TEXT("you have manually changed the ")
        __TEXT("Users and Admins Group registry ")
        __TEXT("settings under this server's AppID.");

    HRESULT hr = CoInitializeEx(0, COINIT_MULTITHREADED);
    if (FAILED(hr))
        return hr;

// look for self-registration flags
    if (strstr(szCmdParam, "/UnregServer") != 0
        || strstr(szCmdParam, "-UnregServer") != 0)
    {
        hr = UnregisterServer();
        CoUninitialize();
        return hr;
    }
    else if (strstr(szCmdParam, "/RegServer") != 0
            || strstr(szCmdParam, "-RegServer") != 0)
    {
        hr = RegisterServer();
        MessageBox(0, pszPrompt, __TEXT("COMChat"),
                MB_SETFOREGROUND);
        CoUninitialize();
        return hr;
    }

// set up process security
    hr = InitializeApplicationSecurity();
    if (SUCCEEDED(hr))
    {
// register class object and wait to die
        DWORD  dwReg;
        static ChatSessionClass cmc;
        hr = CoRegisterClassObject(CLSID_ChatSession,
              static_cast<IExternalConnection*>(&cmc),
                                   CLSCTX_LOCAL_SERVER
                  REGCLS_SUSPENDED|REGCLS_MULTIPLEUSE,
                                   &dwReg);
        if (SUCCEEDED(hr))
        {
            hr = CoResumeClassObjects();
            if (SUCCEEDED(hr))
                WaitForSingleObject(g_heventDone, INFINITE);
            CoRevokeClassObject(dwReg);
        }
        g_pacUsers->Release();
        g_pacAdmins->Release();
    }
    if (FAILED(hr))
        MessageBox(0, pszPrompt, __TEXT("Error"),
                   MB_SETFOREGROUND);

    CoUninitialize();
    return 0;
}
```