附錄 B

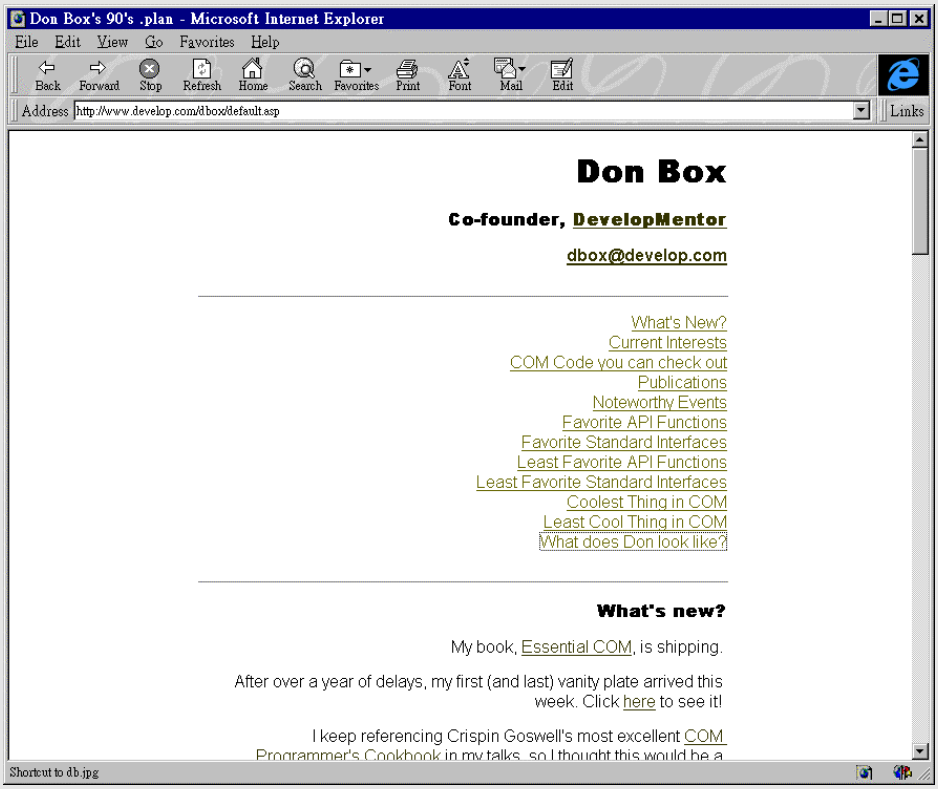
程式碼列表

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。

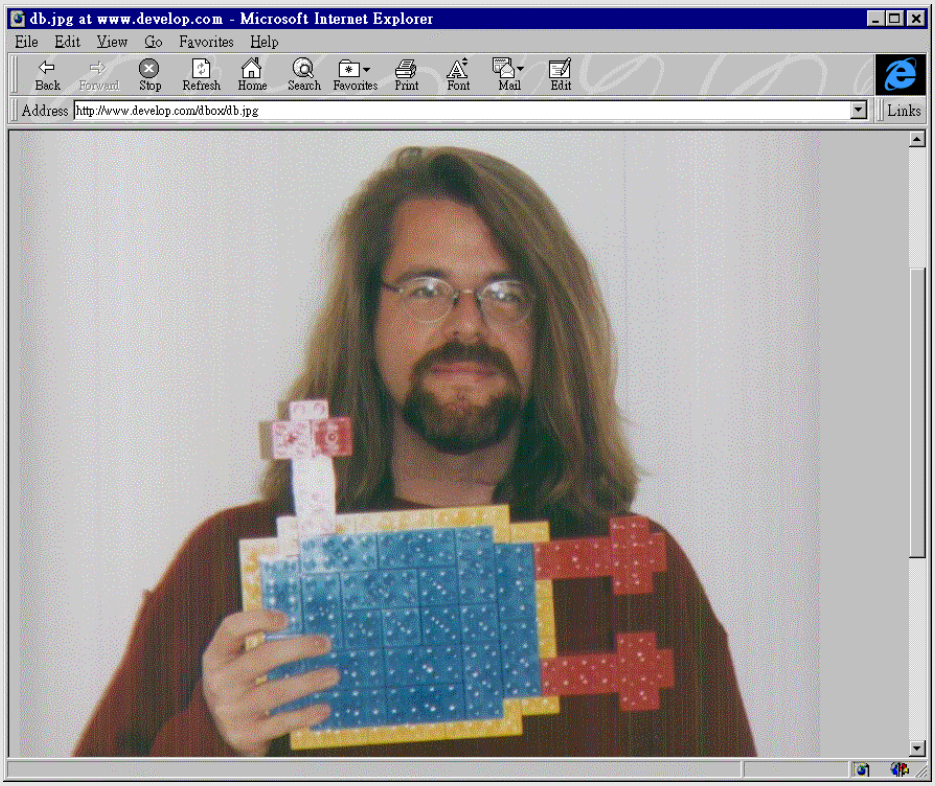
COM Chat：一個建立在 COM 之上的網路間聊程式

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：

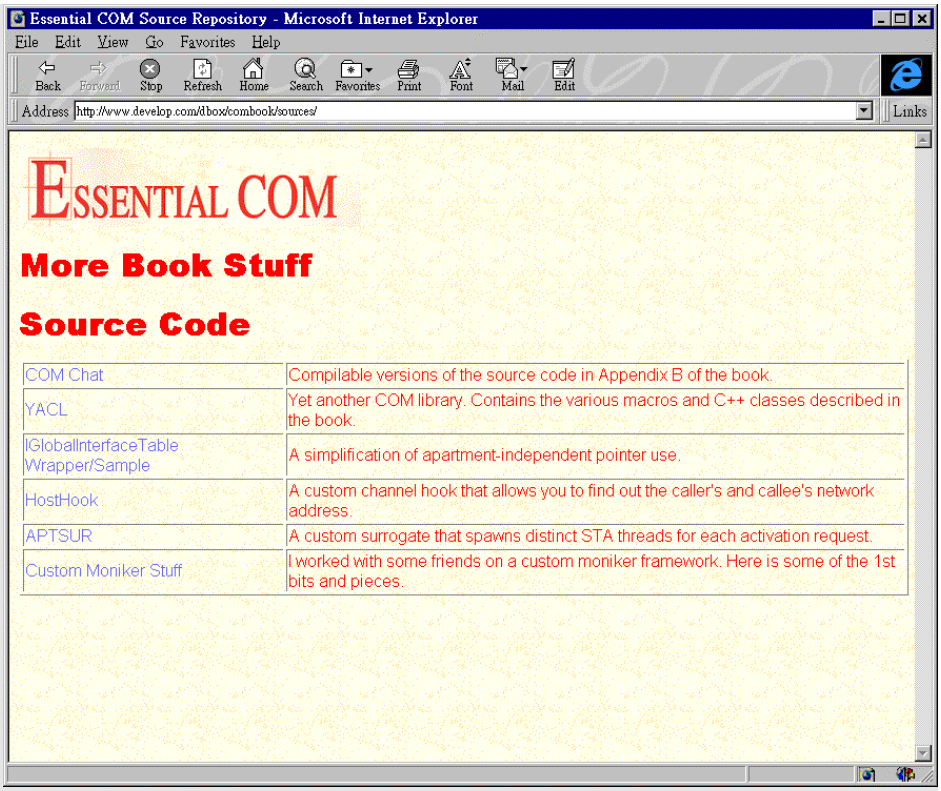


選擇最後一項 "What Does Don look like?"，可看到 Don Box 的尊容：

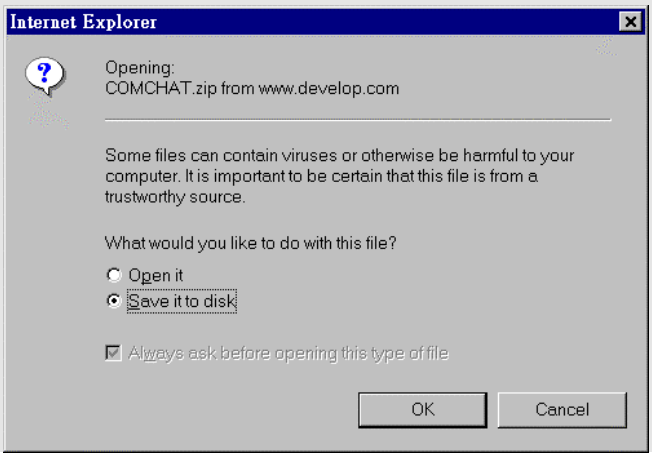


難怪 Charlie Kindel 序中劈頭就說『Don 的像片會出現在書的封底嗎？如果是，他的頭髮會有多長？』☺

進入 combook/sources 網頁，可看到畫面如下：



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：



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
```

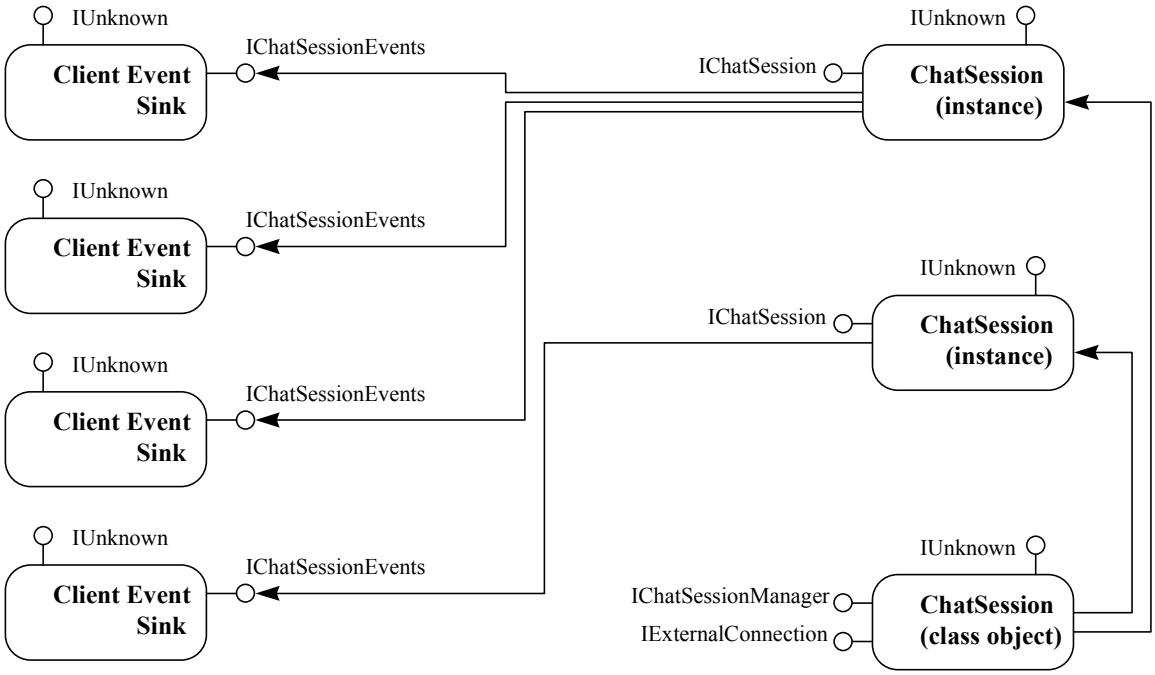


圖 B-1　COM Chat

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

```
#0001  ////////////////////////////////////////////////
#0002  //
#0003  // COMChat.idl
#0004  //
#0005  // Copyright 1997, Don Box/Addison Wesley
#0006  //
#0007  // This code accompanies the book "The Component
#0008  // Object Model" from Addison Wesley. Blah blah blah
#0009  //
#0010  //
#0011
#0012  interface IChatSessionEvents;
#0013
#0014  [
#0015   uuid(5223A050-2441-11d1-AF4F-0060976AA886),
#0016   object
#0017  ]
#0018  interface IChatSession : IUnknown
#0019  {
#0020   import "objidl.idl";
#0021
#0022   [propget] HRESULT SessionName([out, string] OLECHAR **ppwsz);
#0023   HRESULT Say([in, string] const OLECHAR *pwszStatement);
#0024   HRESULT GetStatements([out] IEnumString **ppes);
#0025
#0026   HRESULT Advise([in] IChatSessionEvents *pEventSink,
#0027                  [out] DWORD *pdwReg);
#0028   HRESULT Unadvise([in] DWORD dwReg);
#0029  }
#0030
#0031  [
#0032   uuid(5223A051-2441-11d1-AF4F-0060976AA886),
#0033   object
#0034  ]
#0035  interface IChatSessionEvents : IUnknown
#0036  {
#0037   import "objidl.idl";
#0038   HRESULT OnNewUser([in, string] const OLECHAR *pwszUser);
#0039   HRESULT OnUserLeft([in, string] const OLECHAR *pwszUser);
#0040   HRESULT OnNewStatement([in, string] const OLECHAR *pwszUser,
#0041                          [in, string] const OLECHAR *pwszStmnt);
#0042  }
#0043
```

```
#0044  [
#0045   uuid(5223A052-2441-11d1-AF4F-0060976AA886),
#0046   object
#0047  ]
#0048  interface IChatSessionManager : IUnknown
#0049  {
#0050   import "objidl.idl";
#0051   HRESULT GetSessionNames([out] IEnumString **ppes);
#0052   HRESULT FindSession([in, string] const OLECHAR *pwszName,
#0053                      [in] BOOL bDontCreate,
#0054                      [in] BOOL bAllowAnonymousAccess,
#0055                      [out] IChatSession **ppcs);
#0056   HRESULT DeleteSession([in, string] const OLECHAR *pwszName);
#0057  }
#0058
#0059  cpp_quote("DEFINE_GUID(CLSID_ChatSession,0x5223a053,0x2441,")
#0060  cpp_quote("0x11d1,0xaf,0x4f,0x0,0x60,0x97,0x6a,0xa8,0x86);")
```

client.cpp

```
#0001  ////////////////////////////////////////////////////
#0002  //
#0003  // client.cpp
#0004  //
#0005  // Copyright 1997, Don Box/Addison Wesley
#0006  //
#0007  // This code accompanies the book "The Component
#0008  // Object Model" from Addison Wesley. Blah blah blah
#0009  //
#0010  //
#0011
#0012  #define _WIN32_WINNT 0x403
#0013  #include <windows.h>
#0014  #include <stdio.h>
#0015  #include <initguid.h>
#0016  #include <wchar.h>
#0017  #include "../include/COMChat.h"
#0018  #include "../include/COMChat_i.c"
#0019
#0020  void Error(HRESULT hr, const char *psz)
#0021  {
#0022     printf("%s failed and returned 0x%x\n", psz, hr);
#0023  }
#0024
#0025  // utility function to print command line syntax
```

```
#0026  int Usage(void)
#0027  {
#0028     const char *psz =
#0029      "usage: client.exe <action> <user> <host>\n"
#0030      "   where:\n"
#0031      "         action = /sessions|/chat:session|/delete:session\n"
#0032      "         user = /user:domain\\user /password:pw |"
#0033                "/anonymous | <nothing>\n"
#0034      "         host = /host:hostname | <nothing>\n";
#0035     printf(psz);
#0036     return -1;
#0037  }
#0038
#0039  // utility function for printing a list of strings
#0040  void PrintAllStrings(IEnumString *pes)
#0041  {
#0042      enum { CHUNKSIZE = 64 };
#0043      OLECHAR *rgpwsz[CHUNKSIZE];
#0044      ULONG cFetched;
#0045      HRESULT hr;
#0046      do
#0047      {
#0048          hr = pes->Next(CHUNKSIZE, rgpwsz, &cFetched);
#0049          if (SUCCEEDED(hr))
#0050          {
#0051              for (ULONG i = 0; i < cFetched; i++)
#0052                  if (rgpwsz[i])
#0053                  {
#0054                      wprintf(L"%s\n", rgpwsz[i]);
#0055                      CoTaskMemFree(rgpwsz[i]);
#0056                  }
#0057          }
#0058      } while (hr == S_OK);
#0059  }
#0060
#0061  // utility function to print initial state of
#0062  // a chat session
#0063  void PrintToDate(IChatSession *pcs)
#0064  {
#0065      IEnumString *pes = 0;
#0066      HRESULT hr = pcs->GetStatements(&pes);
#0067      if (SUCCEEDED(hr))
#0068      {
#0069          PrintAllStrings(pes);
#0070          pes->Release();
#0071      }
```

```
#0072  }
#0073
#0074  // this class implements the callback interface
#0075  // that receives chat notifications. It simply
#0076  // prints the event to the console
#0077  class EventSink : public IChatSessionEvents
#0078  {
#0079  public:
#0080      STDMETHODIMP QueryInterface(REFIID riid, void**ppv)
#0081      {
#0082          if (riid == IID_IUnknown)
#0083              *ppv = static_cast<IChatSessionEvents*>(this);
#0084          else if (riid == IID_IChatSessionEvents)
#0085              *ppv = static_cast<IChatSessionEvents*>(this);
#0086          else
#0087              return (*ppv = 0), E_NOINTERFACE;
#0088          reinterpret_cast<IUnknown*>(*ppv)->AddRef();
#0089          return S_OK;
#0090      }
#0091      STDMETHODIMP_(ULONG) AddRef(void)
#0092      {
#0093          return 2;
#0094      }
#0095      STDMETHODIMP_(ULONG) Release(void)
#0096      {
#0097          return 1;
#0098      }
#0099      STDMETHODIMP OnNewStatement(const OLECHAR *pwszUser,
#0100                                  const OLECHAR *pwszStmt)
#0101      {
#0102          wprintf(L"%-14s: %s\n", pwszUser, pwszStmt);
#0103          return S_OK;
#0104      }
#0105      STDMETHODIMP OnNewUser(const OLECHAR *pwszUser)
#0106      {
#0107          wprintf(L"\n\n>>> Say Hello to %s\n\n", pwszUser);
#0108          return S_OK;
#0109      }
#0110      STDMETHODIMP OnUserLeft(const OLECHAR *pwszUser)
#0111      {
#0112          wprintf(L"\n\n>>> Say Bye to %s\n\n", pwszUser);
#0113          return S_OK;
#0114      }
#0115
#0116  };
#0117
```

```
#0118  // type of operations this client can perform
#0119  enum ACTION
#0120  {
#0121      ACTION_NONE,
#0122      ACTION_CHAT,
#0123      ACTION_DELETE_SESSION,
#0124      ACTION_LIST_SESSION_NAMES,
#0125  };
#0126
#0127  // run chat command
#0128  void Chat(const OLECHAR *pwszSession,
#0129           IChatSessionManager *pcsm, // manager
#0130           COAUTHIDENTITY *pcai,     // user
#0131           bool bAnonymous)          // anonymous
#0132  {
#0133  // create or get the named session
#0134      IChatSession *pcs = 0;
#0135      HRESULT hr = pcsm->FindSession(pwszSession, FALSE,
#0136                                     TRUE, &pcs);
#0137      if (SUCCEEDED(hr))
#0138      {
#0139  // adjust security blanket for session interface
#0140          if (!bAnonymous)
#0141              hr = CoSetProxyBlanket(pcs, RPC_C_AUTHN_WINNT,
#0142                                     RPC_C_AUTHZ_NONE, 0,
#0143                                     RPC_C_AUTHN_LEVEL_PKT,
#0144                                     RPC_C_IMP_LEVEL_IDENTIFY,
#0145                                     pcai, EOAC_NONE);
#0146  // catch up on past messages
#0147          PrintToDate(pcs);
#0148  // hook up event sink to receive new messages
#0149          EventSink es;
#0150          DWORD dwReg;
#0151          hr = pcs->Advise(&es, &dwReg);
#0152          if (SUCCEEDED(hr))
#0153          {
#0154  // run UI loop to get statements from console and send them
#0155              OLECHAR wszStmt[4096];
#0156              while (_getws(wszStmt))
#0157              {
#0158                  hr = pcs->Say(wszStmt);
#0159                  if (FAILED(hr))
#0160                      Error(hr, "Say");
#0161              }
#0162  // tear down connection for event sink
#0163              pcs->Unadvise(dwReg);
```

```
#0164        }
#0165        else
#0166           Error(hr, "Advise");
#0167  // release chat session
#0168        pcs->Release();
#0169     }
#0170     else
#0171        Error(hr, "FindSession");
#0172  }
#0173
#0174  // run delete command
#0175  void Delete(const OLECHAR *pwszSession,
#0176           IChatSessionManager *pcsm)
#0177  {
#0178     HRESULT hr = pcsm->DeleteSession(pwszSession);
#0179     if (FAILED(hr))
#0180        Error(hr, "DeleteSession");
#0181  }
#0182
#0183  // run list command
#0184  void List(IChatSessionManager *pcsm)
#0185  {
#0186     IEnumString *pes = 0;
#0187     HRESULT hr = pcsm->GetSessionNames(&pes);
#0188     if (SUCCEEDED(hr))
#0189     {
#0190        printf("Active Sessions:\n");
#0191        PrintAllStrings(pes);
#0192        pes->Release();
#0193     }
#0194  }
#0195
#0196  int main(int argc, char **argv)
#0197  {
#0198  // declare client control state
#0199     bool bAnonymous = false;
#0200     static OLECHAR wszSessionName[1024];
#0201     static OLECHAR wszDomainName[1024];
#0202     static OLECHAR wszUserName[1024];
#0203     static OLECHAR wszPassword[1024];
#0204     static OLECHAR wszHostName[1024];
#0205     COSERVERINFO csi = { 0, wszHostName, 0, 0 };
#0206     COSERVERINFO *pcsi = 0;
#0207     COAUTHIDENTITY cai = {
#0208        wszUserName,
#0209        0,
```

```
#0210         wszDomainName,
#0211         0,
#0212         wszPassword,
#0213         0,
#0214         SEC_WINNT_AUTH_IDENTITY_UNICODE
#0215     };
#0216     static COAUTHIDENTITY *pcai = 0;
#0217     static ACTION action = ACTION_NONE;
#0218
#0219  // parse command line
#0220     for (int i = 1; i < argc; i++)
#0221     {
#0222         if (strcmp(argv[i], "/anonymous") == 0)
#0223             bAnonymous = true;
#0224         else if (strstr(argv[i], "/delete:") == argv[i])
#0225         {
#0226             if (action != ACTION_NONE)
#0227                 return Usage();
#0228             action = ACTION_DELETE_SESSION;
#0229             mbstowcs(wszSessionName, argv[i] + 8, 1024);
#0230         }
#0231         else if (strstr(argv[i], "/chat:") == argv[i])
#0232         {
#0233             if (action != ACTION_NONE)
#0234                 return Usage();
#0235             action = ACTION_CHAT;
#0236             mbstowcs(wszSessionName, argv[i] + 6, 1024);
#0237         }
#0238         else if (strcmp(argv[i], "/sessions") == 0)
#0239         {
#0240             if (action != ACTION_NONE)
#0241                 return Usage();
#0242             action = ACTION_LIST_SESSION_NAMES;
#0243         }
#0244         else if (strstr(argv[i], "/host:") == argv[i])
#0245         {
#0246             if (pcsi != 0)
#0247                 return Usage();
#0248             mbstowcs(wszHostName, argv[i] + 6, 1024);
#0249             pcsi = &csi;
#0250         }
#0251         else if (strstr(argv[i], "/password:") == argv[i])
#0252         {
#0253             mbstowcs(wszPassword, argv[i] + 10, 1024);
#0254             cai.PasswordLength = wcslen(wszPassword);
#0255         }
```

```
#0256         else if (strstr(argv[i], "/user:") == argv[i])
#0257         {
#0258            if (pcai != 0 || bAnonymous)
#0259               return Usage();
#0260            char *pszDelim = strchr(argv[i] + 7, '\\');
#0261            if (pszDelim == 0)
#0262               return Usage();
#0263            *pszDelim = 0;
#0264            pszDelim++;
#0265            mbstowcs(wszDomainName, argv[i] + 6, 1024);
#0266            cai.DomainLength = wcslen(wszDomainName);
#0267            mbstowcs(wszUserName, pszDelim, 1024);
#0268            cai.UserLength = wcslen(wszUserName);
#0269            pcai = &cai;
#0270         }
#0271      }
#0272
#0273      if (action == ACTION_NONE)
#0274         return Usage();
#0275      HRESULT hr = CoInitializeEx(0, COINIT_MULTITHREADED);
#0276      if (FAILED(hr))
#0277         return hr;
#0278
#0279   // allow anonymous callbacks from chat server
#0280      hr = CoInitializeSecurity(0, -1, 0, 0,
#0281                            RPC_C_AUTHN_LEVEL_NONE,
#0282                            RPC_C_IMP_LEVEL_ANONYMOUS,
#0283                            0, EOAC_NONE, 0);
#0284
#0285      if (SUCCEEDED(hr))
#0286      {
#0287   // grab the requested session manager
#0288         IChatSessionManager *pcsm = 0;
#0289         hr = CoGetClassObject(CLSID_ChatSession, CLSCTX_ALL,
#0290                            pcsi, IID_IChatSessionManager,
#0291                            (void**)&pcsm);
#0292         if (SUCCEEDED(hr))
#0293         {
#0294   // apply security blanket if desired
#0295            if (!bAnonymous)
#0296               hr = CoSetProxyBlanket(pcsm, RPC_C_AUTHN_WINNT,
#0297                                   RPC_C_AUTHZ_NONE, 0,
#0298                                   RPC_C_AUTHN_LEVEL_PKT,
#0299                                   RPC_C_IMP_LEVEL_IDENTIFY,
#0300                                   pcai, EOAC_NONE);
#0301   // dispatch request
```

```
#0302            switch (action)
#0303            {
#0304            case ACTION_CHAT:
#0305               Chat(wszSessionName, pcsm, pcai, bAnonymous);
#0306               break;
#0307            case ACTION_DELETE_SESSION:
#0308               Delete(wszSessionName, pcsm);
#0309               break;
#0310            case ACTION_LIST_SESSION_NAMES:
#0311               List(pcsm);
#0312               break;
#0313            default:
#0314               Usage();
#0315            }
#0316  // release session manager
#0317            pcsm->Release();
#0318         }
#0319     }
#0320     CoUninitialize();
#0321     return hr;
#0322  }
```

chatsession.h

```
#0001  ////////////////////////////////////////////////////
#0002  //
#0003  // ChatSession.h
#0004  //
#0005  // Copyright 1997, Don Box/Addison Wesley
#0006  //
#0007  // This code accompanies the book "The Component
#0008  // Object Model" from Addison Wesley. Blah blah blah
#0009  //
#0010  //
#0011
#0012  #ifndef _CHATSESSION_H
#0013  #define _CHATSESSION_H
#0014
#0015  // this pragma shuts up the compiler warnings due to
#0016  // the pre MSC11SP1 debugger choking on long template names.
#0017  #pragma warning(disable:4786)
#0018
#0019  #define _WIN32_WINNT 0x403
#0020  #include <windows.h>
```

```
#0021  #include <map>
#0022  #include <vector>
#0023  #include <string>
#0024  using namespace std;
#0025
#0026  // bring in IDL-generated interface definitions
#0027  #include "..\include\COMChat.h"
#0028
#0029  // this class models a particular chat session
#0030  class ChatSession : public IChatSession
#0031  {
#0032      friend class StatementEnumerator;
#0033      LONG           m_cRef;
#0034      CRITICAL_SECTION    m_csStatementLock;
#0035      CRITICAL_SECTION    m_csAdviseLock;
#0036      OLECHAR            m_wszSessionName[1024];
#0037      bool           m_bIsDeleted;
#0038      bool              m_bAllowAnonymousAccess;
#0039      vector<wstring>     m_statements;
#0040      struct LISTENER
#0041      {
#0042          LISTENER          *pPrev;
#0043          LISTENER          *pNext;
#0044          OLECHAR           *pwszUser;
#0045          IChatSessionEvents *pItf;
#0046      };
#0047      LISTENER         *m_pHeadListeners;
#0048      void SLock(void);
#0049      void SUnlock(void);
#0050      void ALock(void);
#0051      void AUnlock(void);
#0052      bool CheckAccess(const OLECHAR *pwszUser);
#0053  protected:
#0054      virtual ~ChatSession(void);
#0055      void Fire_OnNewStatement(const OLECHAR *pwszUser,
#0056                               const OLECHAR *pwszStatement);
#0057      void Fire_OnNewUser(const OLECHAR *pwszUser);
#0058      void Fire_OnUserLeft(const OLECHAR *pwszUser);
#0059  public:
#0060      ChatSession(const OLECHAR *pwszSessionName,
#0061                  bool bAllowAnonymousAccess);
#0062
#0063      void Disconnect(void);
#0064  // IUnknown methods
#0065      STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
#0066      STDMETHODIMP_(ULONG) AddRef(void);
```

```
#0067     STDMETHODIMP_(ULONG) Release(void);
#0068
#0069  // IChatSession methods
#0070     STDMETHODIMP get_SessionName(OLECHAR **ppwsz);
#0071     STDMETHODIMP Say(const OLECHAR *pwszStatement);
#0072     STDMETHODIMP GetStatements(IEnumString **ppes);
#0073     STDMETHODIMP Advise(IChatSessionEvents *pEventSink,
#0074                         DWORD *pdwReg);
#0075     STDMETHODIMP Unadvise(DWORD dwReg);
#0076  };
#0077
#0078  // this class enumerates the statements of a session
#0079  class StatementEnumerator : public IEnumString
#0080  {
#0081     LONG                     m_cRef;
#0082     ChatSession              *m_pThis;
#0083     vector<wstring>::iterator   m_cursor;
#0084     CRITICAL_SECTION           m_csLock;
#0085  protected:
#0086     void Lock(void);
#0087     void Unlock(void);
#0088     virtual ~StatementEnumerator(void);
#0089  public:
#0090     StatementEnumerator(ChatSession *pThis);
#0091
#0092  // IUnknown methods
#0093     STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
#0094     STDMETHODIMP_(ULONG) AddRef(void);
#0095     STDMETHODIMP_(ULONG) Release(void);
#0096
#0097  // IEnumString methods
#0098     STDMETHODIMP Next(ULONG cElems, OLECHAR **rgElems,
#0099                       ULONG *pcFetched);
#0100     STDMETHODIMP Skip(ULONG cElems);
#0101     STDMETHODIMP Reset(void);
#0102     STDMETHODIMP Clone(IEnumString **ppes);
#0103  };
#0104
#0105  // this class models the management of chat sessions
#0106  // and acts as the class object for CLSID_ChatSession
#0107  class ChatSessionClass : public IChatSessionManager,
#0108                           public IExternalConnection
#0109  {
#0110     friend class SessionNamesEnumerator;
#0111     typedef map<wstring, ChatSession *> SESSIONMAP;
#0112     LONG              m_cStrongLocks;
```

```
#0113     SESSIONMAP         m_sessions;
#0114     CRITICAL_SECTION    m_csSessionLock;
#0115     void Lock(void);
#0116     void Unlock(void);
#0117     bool CheckAccess(const OLECHAR *pwszUser);
#0118  public:
#0119     virtual ~ChatSessionClass(void);
#0120     ChatSessionClass(void);
#0121
#0122     // IUnknown methods
#0123     STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
#0124     STDMETHODIMP_(ULONG) AddRef(void);
#0125     STDMETHODIMP_(ULONG) Release(void);
#0126
#0127  // IExternalConnection methods
#0128     STDMETHODIMP_(DWORD) AddConnection(DWORD extconn, DWORD);
#0129     STDMETHODIMP_(DWORD) ReleaseConnection(DWORD extconn, DWORD,
#0130                                  BOOL bLastReleaseKillsStub);
#0131  // IChatSessionManager methods
#0132     STDMETHODIMP GetSessionNames(IEnumString **ppes);
#0133     STDMETHODIMP FindSession(const OLECHAR *pwszSessionName,
#0134                           BOOL bDontCreate,
#0135                           BOOL bAllowAnonymousAccess,
#0136                           IChatSession **ppcs);
#0137     STDMETHODIMP DeleteSession(const OLECHAR *pwszSessionName);
#0138  };
#0139
#0140  // this class enumerates the session names of a server
#0141  class SessionNamesEnumerator : public IEnumString
#0142  {
#0143     LONG                     m_cRef;
#0144     vector<wstring>           *m_pStrings;
#0145     SessionNamesEnumerator     *m_pCloneSource;
#0146     vector<wstring>::iterator   m_cursor;
#0147     CRITICAL_SECTION           m_csLock;
#0148  protected:
#0149     vector<wstring>& Strings(void);
#0150     void Lock(void);
#0151     void Unlock(void);
#0152     virtual ~SessionNamesEnumerator(void);
#0153  public:
#0154     SessionNamesEnumerator(ChatSessionClass *pSessionClass);
#0155     SessionNamesEnumerator(SessionNamesEnumerator *pCloneSource);
#0156
#0157  // IUnknown methods
#0158     STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
```

```
#0159     STDMETHODIMP_(ULONG) AddRef(void);
#0160     STDMETHODIMP_(ULONG) Release(void);
#0161
#0162  // IEnumString methods
#0163     STDMETHODIMP Next(ULONG cElems, OLECHAR **rgElems,
#0164                       ULONG *pcFetched);
#0165     STDMETHODIMP Skip(ULONG cElems);
#0166     STDMETHODIMP Reset(void);
#0167     STDMETHODIMP Clone(IEnumString **ppes);
#0168  };
#0169
#0170  #endif
```

chatsession.cpp

```
#0001  ////////////////////////////////////////////////////
#0002  //
#0003  // ChatSession.cpp
#0004  //
#0005  // Copyright 1997, Don Box/Addison Wesley
#0006  //
#0007  // This code accompanies the book "The Component
#0008  // Object Model" from Addison Wesley. Blah blah blah
#0009  //
#0010  //
#0011
#0012  #include "ChatSession.h"
#0013  #include <iaccess.h>
#0014
#0015  // these routines are defined in svc.cpp to
#0016  // control server lifetime
#0017  extern void ModuleLock(void);
#0018  extern void ModuleUnlock(void);
#0019
#0020  // these access control objects are created
#0021  // in svc.cpp to control various privileged
#0022  // operations. Most operations in this class
#0023  // are non-privileged, so anyone can get in.
#0024  extern IAccessControl *g_pacUsers;
#0025  extern IAccessControl *g_pacAdmins;
#0026
#0027  // utility functions ////////////////////////
#0028
#0029  // duplicate an OLECHAR * using CoTaskMemAlloc
```

```
#0030  OLECHAR *OLESTRDUP(const OLECHAR *pwsz)
#0031  {
#0032      DWORD cb = sizeof(OLECHAR)*(wcslen(pwsz) + 1);
#0033      OLECHAR *pwszResult = (OLECHAR*)CoTaskMemAlloc(cb);
#0034      if (pwszResult)
#0035          wcscpy(pwszResult, pwsz);
#0036      return pwszResult;
#0037  }
#0038
#0039  // get the caller's username (or "anonymous" if
#0040  // no authentication was specified by the caller).
#0041  OLECHAR *GetCaller(void)
#0042  {
#0043      OLECHAR *pwsz = 0;
#0044      HRESULT hr = CoQueryClientBlanket(0,0,0,0,0,(void**)&pwsz,0);
#0045      if (SUCCEEDED(hr))
#0046          return OLESTRDUP(pwsz);
#0047      else
#0048          return OLESTRDUP(OLESTR("anonymous"));
#0049  }
#0050
#0051  // class ChatSession ////////////////////////////////
#0052
#0053  ChatSession::ChatSession(const OLECHAR *pwszSessionName,
#0054                           bool bAllowAnonymousAccess)
#0055  : m_cRef(0),
#0056    m_bAllowAnonymousAccess(bAllowAnonymousAccess),
#0057    m_pHeadListeners(0)
#0058  {
#0059      wcscpy(m_wszSessionName, pwszSessionName);
#0060      InitializeCriticalSection(&m_csStatementLock);
#0061      InitializeCriticalSection(&m_csAdviseLock);
#0062  }
#0063
#0064  ChatSession::~ChatSession(void)
#0065  {
#0066      DeleteCriticalSection(&m_csStatementLock);
#0067      DeleteCriticalSection(&m_csAdviseLock);
#0068  // tear down connected listeners
#0069      while (m_pHeadListeners)
#0070      {
#0071          LISTENER *pThisNode = m_pHeadListeners;
#0072          if (pThisNode->pItf)
#0073              pThisNode->pItf->Release();
#0074          if (pThisNode->pwszUser)
#0075              CoTaskMemFree(pThisNode->pwszUser);
```

```
#0076         m_pHeadListeners = pThisNode->pNext;
#0077         delete pThisNode;
#0078     }
#0079 }
#0080
#0081 // helper methods ////////////
#0082
#0083 void ChatSession::Disconnect(void)
#0084 {
#0085     CoDisconnectObject(this, 0);
#0086 // tear down connected listeners
#0087     ALock();
#0088     while (m_pHeadListeners)
#0089     {
#0090         LISTENER *pThisNode = m_pHeadListeners;
#0091         if (pThisNode->pItf)
#0092             pThisNode->pItf->Release();
#0093         if (pThisNode->pwszUser)
#0094             CoTaskMemFree(pThisNode->pwszUser);
#0095         m_pHeadListeners = pThisNode->pNext;
#0096         delete pThisNode;
#0097     }
#0098     AUnlock();
#0099 }
#0100
#0101 // send the OnNewStatement event to all listeners
#0102 void
#0103 ChatSession::Fire_OnNewStatement(const OLECHAR *pwszUser,
#0104                                 const OLECHAR *pwszStatement)
#0105 {
#0106     ALock();
#0107     for (LISTENER *pNode = m_pHeadListeners;
#0108          pNode != 0; pNode = pNode->pNext)
#0109     {
#0110         if (pNode->pItf)
#0111             pNode->pItf->OnNewStatement(pwszUser, pwszStatement);
#0112     }
#0113     AUnlock();
#0114 }
#0115
#0116 // send the OnNewUser event to all listeners
#0117 void
#0118 ChatSession::Fire_OnNewUser(const OLECHAR *pwszUser)
#0119 {
#0120     ALock();
#0121     for (LISTENER *pNode = m_pHeadListeners;
```

```
#0122          pNode != 0; pNode = pNode->pNext)
#0123     {
#0124        if (pNode->pItf)
#0125           pNode->pItf->OnNewUser(pwszUser);
#0126     }
#0127     AUnlock();
#0128  }
#0129
#0130  // send the OnUserLeft event to all listeners
#0131  void
#0132  ChatSession::Fire_OnUserLeft(const OLECHAR *pwszUser)
#0133  {
#0134     ALock();
#0135     for (LISTENER *pNode = m_pHeadListeners;
#0136          pNode != 0; pNode = pNode->pNext)
#0137     {
#0138        if (pNode->pItf)
#0139           pNode->pItf->OnUserLeft(pwszUser);
#0140     }
#0141     AUnlock();
#0142  }
#0143
#0144  // lock wrappers
#0145  void ChatSession::SLock(void)
#0146  {
#0147     EnterCriticalSection(&m_csStatementLock);
#0148  }
#0149
#0150  void ChatSession::SUnlock(void)
#0151  {
#0152     LeaveCriticalSection(&m_csStatementLock);
#0153  }
#0154
#0155  void ChatSession::ALock(void)
#0156  {
#0157     EnterCriticalSection(&m_csAdviseLock);
#0158  }
#0159
#0160  void ChatSession::AUnlock(void)
#0161  {
#0162     LeaveCriticalSection(&m_csAdviseLock);
#0163  }
#0164
#0165  // helper method to check access to Say method
#0166  bool
#0167  ChatSession::CheckAccess(const OLECHAR *pwszUser)
```

```
#0168  {
#0169      if (wcscmp(pwszUser, L"anonymous") == 0)
#0170          return m_bAllowAnonymousAccess;
#0171  // form trustee from caller and use Access Control
#0172  // object hardwired to COMChat Users group
#0173      TRUSTEEW trustee = {
#0174          0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
#0175          TRUSTEE_IS_USER,
#0176          const_cast<OLECHAR*>(pwszUser)
#0177      };
#0178      BOOL bIsAllowed;
#0179      HRESULT hr = g_pacUsers->IsAccessAllowed(&trustee,0,
#0180                                               COM_RIGHTS_EXECUTE,
#0181                                               &bIsAllowed);
#0182      return SUCCEEDED(hr) && bIsAllowed != FALSE;
#0183  }
#0184
#0185  // IUnknown methods
#0186  STDMETHODIMP
#0187  ChatSession::QueryInterface(REFIID riid, void **ppv)
#0188  {
#0189      if (riid == IID_IUnknown)
#0190          *ppv = static_cast<IChatSession*>(this);
#0191      else if (riid == IID_IChatSession)
#0192          *ppv = static_cast<IChatSession*>(this);
#0193      else
#0194          return (*ppv = 0), E_NOINTERFACE;
#0195      reinterpret_cast<IUnknown*>(*ppv)->AddRef();
#0196      return S_OK;
#0197
#0198  }
#0199
#0200  STDMETHODIMP_(ULONG)
#0201  ChatSession::AddRef(void)
#0202  {
#0203      ModuleLock();
#0204      return InterlockedIncrement(&m_cRef);
#0205  }
#0206
#0207  STDMETHODIMP_(ULONG)
#0208  ChatSession::Release(void)
#0209  {
#0210      LONG res = InterlockedDecrement(&m_cRef);
#0211      if (res == 0)
#0212          delete this;
#0213      ModuleUnlock();
```

```
#0214     return res;
#0215  }
#0216
#0217  // IChatSession methods
#0218  STDMETHODIMP
#0219  ChatSession::get_SessionName(OLECHAR **ppwsz)
#0220  {
#0221     if (!ppwsz)
#0222        return E_INVALIDARG;
#0223     else if ((*ppwsz = OLESTRDUP(m_wszSessionName)) == 0)
#0224        return E_OUTOFMEMORY;
#0225     return S_OK;
#0226  }
#0227
#0228  STDMETHODIMP
#0229  ChatSession::Say(const OLECHAR *pwszStatement)
#0230  {
#0231     HRESULT hr = S_OK;
#0232  // protect access to method
#0233     OLECHAR *pwszUser = GetCaller();
#0234     if (pwszUser && CheckAccess(pwszUser))
#0235     {
#0236        SLock();
#0237        try
#0238        {
#0239           wstring s = pwszUser;
#0240           s += L":";
#0241           s += pwszStatement;
#0242           m_statements.push_back(s);
#0243        }
#0244        catch(...)
#0245        {
#0246           hr = E_OUTOFMEMORY;
#0247        }
#0248        SUnlock();
#0249        if (SUCCEEDED(hr))
#0250           Fire_OnNewStatement(pwszUser, pwszStatement);
#0251     }
#0252     else
#0253        hr = E_ACCESSDENIED;
#0254     CoTaskMemFree(pwszUser);
#0255     return hr;
#0256  }
#0257
#0258  STDMETHODIMP
#0259  ChatSession::GetStatements(IEnumString **ppes)
```

```
#0260 {
#0261     if (ppes == 0)
#0262         return E_INVALIDARG;
#0263     *ppes = new StatementEnumerator(this);
#0264     if (*ppes == 0)
#0265         return E_OUTOFMEMORY;
#0266     (*ppes)->AddRef();
#0267     return S_OK;
#0268 }
#0269
#0270 STDMETHODIMP
#0271 ChatSession::Advise(IChatSessionEvents *pEventSink,
#0272                  DWORD *pdwReg)
#0273 {
#0274     HRESULT hr = S_OK;
#0275     if (pEventSink == 0 || pdwReg == 0)
#0276         return E_INVALIDARG;
#0277     LISTENER *pNew = new LISTENER;
#0278     if (pNew == 0)
#0279         return E_OUTOFMEMORY;
#0280     OLECHAR *pwszUser = GetCaller();
#0281     if (pwszUser)
#0282     {
#0283         Fire_OnNewUser(pwszUser);
#0284         ALock();
#0285         pNew->pwszUser = pwszUser;
#0286         if (pNew->pItf = pEventSink)
#0287             pEventSink->AddRef();
#0288         pNew->pNext = m_pHeadListeners;
#0289         if (m_pHeadListeners)
#0290             m_pHeadListeners->pPrev = pNew;
#0291         pNew->pPrev = 0;
#0292         m_pHeadListeners = pNew;
#0293         AUnlock();
#0294     }
#0295     else
#0296     {
#0297         delete pNew;
#0298         return E_OUTOFMEMORY;
#0299     }
#0300     *pdwReg = reinterpret_cast<DWORD>(pNew);
#0301     return hr;
#0302 }
#0303
#0304 STDMETHODIMP
#0305 ChatSession::Unadvise(DWORD dwReg)
```

```
#0306 {
#0307     if (dwReg == 0)
#0308         return E_INVALIDARG;
#0309     HRESULT hr = S_OK;
#0310     LISTENER *pThisNode = reinterpret_cast<LISTENER *>(dwReg);
#0311     ALock();
#0312     if (pThisNode->pPrev)
#0313         pThisNode->pPrev->pNext = pThisNode->pNext;
#0314     else
#0315         m_pHeadListeners = pThisNode->pNext;
#0316     if (pThisNode->pNext)
#0317         pThisNode->pNext->pPrev = pThisNode->pPrev;
#0318     if (pThisNode->pItf)
#0319         pThisNode->pItf->Release();
#0320     OLECHAR *pwszUser = pThisNode->pwszUser;
#0321     delete pThisNode;
#0322     AUnlock();
#0323     Fire_OnUserLeft(pwszUser);
#0324     CoTaskMemFree(pwszUser);
#0325     return hr;
#0326 }
#0327
#0328 // class StatementEnumerator ////////////////////
#0329
#0330 StatementEnumerator::StatementEnumerator(ChatSession *pThis)
#0331 : m_cRef(0),
#0332   m_pThis(pThis),
#0333   m_cursor(pThis->m_statements.begin())
#0334 {
#0335     m_pThis->AddRef();
#0336     InitializeCriticalSection(&m_csLock);
#0337 }
#0338
#0339 StatementEnumerator::~StatementEnumerator(void)
#0340 {
#0341     m_pThis->Release();
#0342     DeleteCriticalSection(&m_csLock);
#0343 }
#0344
#0345 // lock helpers (note that ChatSession is locked
#0346 // simultaneously)
#0347 void
#0348 StatementEnumerator::Lock(void)
#0349 {
#0350     EnterCriticalSection(&m_csLock);
#0351     m_pThis->SLock();
```

```
#0352  }
#0353
#0354  void
#0355  StatementEnumerator::Unlock(void)
#0356  {
#0357      LeaveCriticalSection(&m_csLock);
#0358      m_pThis->SUnlock();
#0359  }
#0360
#0361  // IUnknown methods
#0362  STDMETHODIMP
#0363  StatementEnumerator::QueryInterface(REFIID riid, void **ppv)
#0364  {
#0365      if (riid == IID_IUnknown)
#0366          *ppv = static_cast<IEnumString*>(this);
#0367      else if (riid == IID_IEnumString)
#0368          *ppv = static_cast<IEnumString*>(this);
#0369      else
#0370          return (*ppv = 0), E_NOINTERFACE;
#0371      reinterpret_cast<IUnknown*>(*ppv)->AddRef();
#0372      return S_OK;
#0373
#0374  }
#0375
#0376  STDMETHODIMP_(ULONG)
#0377  StatementEnumerator::AddRef(void)
#0378  {
#0379      return InterlockedIncrement(&m_cRef);
#0380  }
#0381
#0382  STDMETHODIMP_(ULONG)
#0383  StatementEnumerator::Release(void)
#0384  {
#0385      LONG res = InterlockedDecrement(&m_cRef);
#0386      if (res == 0)
#0387          delete this;
#0388      return res;
#0389  }
#0390
#0391  // IEnumString methods
#0392  STDMETHODIMP
#0393  StatementEnumerator::Next(ULONG cElems, OLECHAR **rgElems,
#0394                            ULONG *pcFetched)
#0395  {
#0396      if (pcFetched == 0 && cElems > 1)
#0397          return E_INVALIDARG;
```

```
#0398     ZeroMemory(rgElems, sizeof(OLECHAR*) * cElems);
#0399     Lock();
#0400     ULONG cActual = 0;
#0401     while (cActual < cElems
#0402           && m_cursor != m_pThis->m_statements.end())
#0403     {
#0404        if (rgElems[cActual] = OLESTRDUP((*m_cursor).c_str()))
#0405        {
#0406           m_cursor++;
#0407           cActual++;
#0408        }
#0409        else // allocation error, unwind
#0410        {
#0411           while (cActual > 0)
#0412           {
#0413              cActual--;
#0414              CoTaskMemFree(rgElems[cActual]);
#0415              rgElems[cActual] = 0;
#0416           }
#0417           break;
#0418        }
#0419     }
#0420     Unlock();
#0421     if (pcFetched)
#0422        *pcFetched = cActual;
#0423     return cElems == cActual ? S_OK : S_FALSE;
#0424  }
#0425
#0426  STDMETHODIMP
#0427  StatementEnumerator::Skip(ULONG cElems)
#0428  {
#0429     Lock();
#0430     ULONG cActual = 0;
#0431     while (cActual < cElems
#0432           && m_cursor != m_pThis->m_statements.end())
#0433     {
#0434        m_cursor++;
#0435        cActual++;
#0436     }
#0437     Unlock();
#0438     return cElems == cActual ? S_OK : S_FALSE;
#0439  }
#0440
#0441  STDMETHODIMP
#0442  StatementEnumerator::Reset(void)
#0443  {
```

```
#0444     Lock();
#0445     m_cursor = m_pThis->m_statements.begin();
#0446     Unlock();
#0447     return S_OK;
#0448 }
#0449
#0450 STDMETHODIMP
#0451 StatementEnumerator::Clone(IEnumString **ppes)
#0452 {
#0453     if (ppes == 0)
#0454         return E_INVALIDARG;
#0455     if (*ppes = new StatementEnumerator(m_pThis))
#0456         return S_OK;
#0457     return E_OUTOFMEMORY;
#0458 }
#0459
#0460 // class ChatSessionClass ////////////////////
#0461
#0462 ChatSessionClass::ChatSessionClass(void)
#0463 : m_cStrongLocks(0)
#0464 {
#0465     InitializeCriticalSection(&m_csSessionLock);
#0466 }
#0467
#0468 ChatSessionClass::~ChatSessionClass(void)
#0469 {
#0470     DeleteCriticalSection(&m_csSessionLock);
#0471 }
#0472
#0473 void
#0474 ChatSessionClass::Lock(void)
#0475 {
#0476     EnterCriticalSection(&m_csSessionLock);
#0477 }
#0478
#0479 void
#0480 ChatSessionClass::Unlock(void)
#0481 {
#0482     LeaveCriticalSection(&m_csSessionLock);
#0483 }
#0484 // helper method to protect access to DeleteSession
#0485 // to only allow COMChat Admins to delete groups
#0486 bool
#0487 ChatSessionClass::CheckAccess(const OLECHAR *pwszUser)
#0488 {
#0489     if (wcscmp(pwszUser, L"anonymous") == 0)
```

```
#0490         return false;
#0491
#0492     TRUSTEEW trustee = {
#0493         0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
#0494         TRUSTEE_IS_USER, const_cast<OLECHAR*>(pwszUser)
#0495     };
#0496     BOOL bIsAllowed;
#0497     HRESULT hr = g_pacAdmins->IsAccessAllowed(&trustee,0,
#0498                                         COM_RIGHTS_EXECUTE,
#0499                                         &bIsAllowed);
#0500     if (FAILED(hr))
#0501         bIsAllowed = false;
#0502     return SUCCEEDED(hr) && bIsAllowed != FALSE;
#0503 }
#0504
#0505
#0506 // IUnknown methods
#0507 STDMETHODIMP
#0508 ChatSessionClass::QueryInterface(REFIID riid, void **ppv)
#0509 {
#0510     if (riid == IID_IUnknown)
#0511         *ppv = static_cast<IChatSessionManager*>(this);
#0512     else if (riid == IID_IChatSessionManager)
#0513         *ppv = static_cast<IChatSessionManager*>(this);
#0514     else if (riid == IID_IExternalConnection)
#0515         *ppv = static_cast<IExternalConnection*>(this);
#0516     else
#0517         return (*ppv = 0), E_NOINTERFACE;
#0518     reinterpret_cast<IUnknown*>(*ppv)->AddRef();
#0519     return S_OK;
#0520 }
#0521
#0522 STDMETHODIMP_(ULONG)
#0523 ChatSessionClass::AddRef(void)
#0524 {
#0525     return 2;
#0526 }
#0527
#0528 STDMETHODIMP_(ULONG)
#0529 ChatSessionClass::Release(void)
#0530 {
#0531     return 1;
#0532 }
#0533
#0534 // IExternalConnection methods
#0535 STDMETHODIMP_(DWORD)
```

```
#0536  ChatSessionClass::AddConnection(DWORD extconn, DWORD)
#0537  {
#0538      if (extconn & EXTCONN_STRONG)
#0539      {
#0540          ModuleLock();
#0541          return InterlockedIncrement(&m_cStrongLocks);
#0542      }
#0543      return 0;
#0544  }
#0545
#0546  STDMETHODIMP_(DWORD)
#0547  ChatSessionClass::ReleaseConnection(DWORD extconn, DWORD,
#0548                                      BOOL bLastReleaseKillsStub)
#0549  {
#0550      if (extconn & EXTCONN_STRONG)
#0551      {
#0552          LONG res = InterlockedDecrement(&m_cStrongLocks);
#0553          if (res == 0 && bLastReleaseKillsStub)
#0554              CoDisconnectObject(
#0555                  static_cast<IExternalConnection*>(this), 0);
#0556          ModuleUnlock();
#0557          return res;
#0558      }
#0559      return 0;
#0560  }
#0561
#0562  // IChatSessionManager methods
#0563  STDMETHODIMP
#0564  ChatSessionClass::GetSessionNames(IEnumString **ppes)
#0565  {
#0566      if (ppes == 0)
#0567          return E_INVALIDARG;
#0568      if (*ppes = new SessionNamesEnumerator(this))
#0569      {
#0570          (*ppes)->AddRef();
#0571          return S_OK;
#0572      }
#0573      else
#0574          return E_OUTOFMEMORY;
#0575  }
#0576
#0577  STDMETHODIMP
#0578  ChatSessionClass::FindSession(const OLECHAR *pwszSessionName,
#0579                              BOOL bDontCreate,
#0580                              BOOL bAllowAnonymousAccess,
#0581                              IChatSession **ppcs)
```

```
#0582  {
#0583      if (ppcs == 0)
#0584          return E_INVALIDARG;
#0585      HRESULT hr = E_FAIL;
#0586      *ppcs = 0;
#0587      OLECHAR *pwszUser = GetCaller();
#0588      Lock();
#0589      SESSIONMAP::iterator it = m_sessions.find(pwszSessionName);
#0590      if (it == m_sessions.end())
#0591      {
#0592          if (bDontCreate)
#0593              hr = E_FAIL;
#0594          else if (!bAllowAnonymousAccess
#0595                  && wcscmp(pwszUser, L"anonymous") == 0)
#0596              hr = E_ACCESSDENIED;
#0597          else
#0598          {
#0599              ChatSession *pNew =
#0600                  new ChatSession(pwszSessionName,
#0601                                  bAllowAnonymousAccess != FALSE);
#0602              if (pNew)
#0603              {
#0604                  pNew->AddRef();
#0605                  m_sessions.insert(
#0606                      pair<wstring,
#0607                           ChatSession*>(pwszSessionName,
#0608                                         pNew));
#0609                  (*ppcs = pNew)->AddRef();
#0610                  hr = S_OK;
#0611              }
#0612              else
#0613                  hr = E_OUTOFMEMORY;
#0614          }
#0615      }
#0616      else
#0617      {
#0618          (*ppcs = (*it).second)->AddRef();
#0619          hr = S_OK;
#0620      }
#0621      Unlock();
#0622      CoTaskMemFree(pwszUser);
#0623      return hr;
#0624  }
#0625
#0626  STDMETHODIMP
#0627  ChatSessionClass::DeleteSession(const OLECHAR *pwszSessionName)
```

```
#0628 {
#0629     if (pwszSessionName == 0)
#0630         return E_INVALIDARG;
#0631     HRESULT hr = E_FAIL;
#0632     OLECHAR *pwszUser = GetCaller();
#0633     if (CheckAccess(pwszUser))
#0634     {
#0635         Lock();
#0636         SESSIONMAP::iterator it
#0637                       = m_sessions.find(pwszSessionName);
#0638         if (it == m_sessions.end())
#0639         {
#0640             hr = E_FAIL;
#0641         }
#0642         else
#0643         {
#0644             (*it).second->Disconnect();
#0645             (*it).second->Release();
#0646             m_sessions.erase(it);
#0647             hr = S_OK;
#0648         }
#0649         Unlock();
#0650     }
#0651     else
#0652         hr = E_ACCESSDENIED;
#0653     CoTaskMemFree(pwszUser);
#0654     return hr;
#0655 }
#0656
#0657 // class SessionNamesEnumerator
#0658
#0659 vector<wstring>&
#0660 SessionNamesEnumerator::Strings(void)
#0661 {
#0662     if (m_pStrings)
#0663         return *m_pStrings;
#0664     else
#0665         return *(m_pCloneSource->m_pStrings);
#0666 }
#0667
#0668 void
#0669 SessionNamesEnumerator::Lock(void)
#0670 {
#0671     EnterCriticalSection(&m_csLock);
#0672 }
#0673
```

```
#0674  void
#0675  SessionNamesEnumerator::Unlock(void)
#0676  {
#0677      LeaveCriticalSection(&m_csLock);
#0678  }
#0679
#0680  SessionNamesEnumerator::SessionNamesEnumerator(
#0681                               ChatSessionClass *pSessionClass)
#0682  : m_cRef(0),
#0683    m_pStrings(0),
#0684    m_pCloneSource(0)
#0685  {
#0686      typedef ChatSessionClass::SESSIONMAP::iterator iterator;
#0687      ChatSessionClass::SESSIONMAP &sessions
#0688          = pSessionClass->m_sessions;
#0689      m_pStrings = new vector<wstring>;
#0690      pSessionClass->Lock();
#0691      for (iterator it = sessions.begin();
#0692           it != sessions.end();
#0693           it++)
#0694      {
#0695          m_pStrings->push_back((*it).first);
#0696      }
#0697      pSessionClass->Unlock();
#0698      m_cursor = Strings().begin();
#0699      InitializeCriticalSection(&m_csLock);
#0700  }
#0701
#0702  SessionNamesEnumerator::SessionNamesEnumerator(
#0703                       SessionNamesEnumerator *pCloneSource)
#0704  : m_cRef(0),
#0705    m_pStrings(0),
#0706    m_pCloneSource(pCloneSource)
#0707  {
#0708      m_pCloneSource->AddRef();
#0709      m_cursor = Strings().begin();
#0710      InitializeCriticalSection(&m_csLock);
#0711  }
#0712
#0713  SessionNamesEnumerator::~SessionNamesEnumerator(void)
#0714  {
#0715      if (m_pCloneSource)
#0716          m_pCloneSource->Release();
#0717      else if (m_pStrings)
#0718          delete m_pStrings;
#0719      DeleteCriticalSection(&m_csLock);
```

```
#0720  }
#0721
#0722  // IUnknown methods
#0723
#0724  STDMETHODIMP
#0725  SessionNamesEnumerator::QueryInterface(REFIID riid, void **ppv)
#0726  {
#0727      if (riid == IID_IUnknown)
#0728          *ppv = static_cast<IEnumString*>(this);
#0729      else if (riid == IID_IEnumString)
#0730          *ppv = static_cast<IEnumString*>(this);
#0731      else
#0732          return (*ppv = 0), E_NOINTERFACE;
#0733      reinterpret_cast<IUnknown*>(*ppv)->AddRef();
#0734      return S_OK;
#0735  }
#0736
#0737  STDMETHODIMP_(ULONG)
#0738  SessionNamesEnumerator::AddRef(void)
#0739  {
#0740      ModuleLock();
#0741      return InterlockedIncrement(&m_cRef);
#0742  }
#0743
#0744  STDMETHODIMP_(ULONG)
#0745  SessionNamesEnumerator::Release(void)
#0746  {
#0747      LONG res = InterlockedDecrement(&m_cRef);
#0748      if (res == 0)
#0749          delete this;
#0750      ModuleUnlock();
#0751      return res;
#0752  }
#0753
#0754  // IEnumString methods
#0755  STDMETHODIMP
#0756  SessionNamesEnumerator::Next(ULONG cElems, OLECHAR **rgElems,
#0757                            ULONG *pcFetched)
#0758  {
#0759      if (cElems > 1 && pcFetched == 0)
#0760          return E_INVALIDARG;
#0761      ULONG cActual = 0;
#0762      vector<wstring> &rstrings = Strings();
#0763      Lock();
#0764      while (cActual < cElems
#0765             && m_cursor != rstrings.end())
```

```
#0766     {
#0767        if (rgElems[cActual] = OLESTRDUP((*m_cursor).c_str()))
#0768        {
#0769           m_cursor++;
#0770           cActual++;
#0771        }
#0772        else // allocation error, unwind
#0773        {
#0774           while (cActual > 0)
#0775           {
#0776              cActual--;
#0777              CoTaskMemFree(rgElems[cActual]);
#0778              rgElems[cActual] = 0;
#0779           }
#0780           break;
#0781        }
#0782     }
#0783     Unlock();
#0784     if (cActual)
#0785        *pcFetched = cActual;
#0786     return cActual == cElems ? S_OK : S_FALSE;
#0787  }
#0788
#0789  STDMETHODIMP
#0790  SessionNamesEnumerator::Skip(ULONG cElems)
#0791  {
#0792     ULONG cActual = 0;
#0793     vector<wstring> &rstrings = Strings();
#0794     Lock();
#0795     while (cActual < cElems
#0796           && m_cursor != rstrings.end())
#0797     {
#0798        m_cursor++;
#0799        cActual++;
#0800     }
#0801     Unlock();
#0802     return cActual == cElems ? S_OK : S_FALSE;
#0803  }
#0804
#0805  STDMETHODIMP
#0806  SessionNamesEnumerator::Reset(void)
#0807  {
#0808     Lock();
#0809     m_cursor = Strings().begin();
#0810     Unlock();
#0811     return S_OK;
```

```
#0812  }
#0813
#0814  STDMETHODIMP
#0815  SessionNamesEnumerator::Clone(IEnumString **ppes)
#0816  {
#0817      if (ppes == 0)
#0818          return E_INVALIDARG;
#0819      SessionNamesEnumerator *pCloneSource = m_pCloneSource;
#0820      if (pCloneSource == 0) // we are the source
#0821          m_pCloneSource = this;
#0822      *ppes = new SessionNamesEnumerator(pCloneSource);
#0823      if (*ppes)
#0824      {
#0825          (*ppes)->AddRef();
#0826          return S_OK;
#0827      }
#0828      return E_OUTOFMEMORY;
#0829  }
```

svc.cpp

```
#0001  ////////////////////////////////////////////////
#0002  //
#0003  // svc.cpp
#0004  //
#0005  // Copyright 1997, Don Box/Addison Wesley
#0006  //
#0007  // This code accompanies the book "The Component
#0008  // Object Model" from Addison Wesley. Blah blah blah
#0009  //
#0010  //
#0011
#0012  #define _WIN32_WINNT 0x403
#0013  #include <windows.h>
#0014  #include <olectl.h>
#0015  #include <initguid.h>
#0016  #include <iaccess.h>
#0017
#0018  #include "ChatSession.h"
#0019  #include "../include/COMChat_i.c"
#0020
#0021
#0022  #if !defined(HAVE_IID_IACCESSCONTROL)
#0023  // there is a common bug is the SDK headers and libs
```

```
#0024  // that causes IID_IAccessControl to be undefined.
#0025  // We define it here to give the GUID linkage.
#0026  DEFINE_GUID(IID_IAccessControl,0xEEDD23E0, 0x8410, 0x11CE,
#0027            0xA1, 0xC3, 0x08, 0x00, 0x2B, 0x2B, 0x8D, 0x8F);
#0028  #endif
#0029
#0030  // standard MTA lifetime management helpers
#0031  HANDLE g_heventDone = CreateEvent(0, TRUE, FALSE, 0);
#0032
#0033  void ModuleLock(void)
#0034  {
#0035      CoAddRefServerProcess();
#0036  }
#0037
#0038  void ModuleUnlock(void)
#0039  {
#0040      if (CoReleaseServerProcess() == 0)
#0041          SetEvent(g_heventDone);
#0042  }
#0043
#0044  // standard self-registration table
#0045  const char *g_RegTable[][3] = {
#0046  { "CLSID\\{5223A053-2441-11d1-AF4F-0060976AA886}",
#0047    0, "ChatSession" },
#0048  { "CLSID\\{5223A053-2441-11d1-AF4F-0060976AA886}",
#0049    "AppId", "{5223A054-2441-11d1-AF4F-0060976AA886}"
#0050  },
#0051  { "CLSID\\{5223A053-2441-11d1-AF4F-0060976AA886}\\LocalServer32",
#0052    0, (const char*)-1 // rogue value indicating file name
#0053  },
#0054  { "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
#0055    0, "ChatSession Server" },
#0056  { "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
#0057    "RunAs", "Domain\\ReplaceMe"
#0058  },
#0059  { "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
#0060    "Chat Admins Group", "Domain\\ReplaceMe"
#0061  },
#0062  { "AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}",
#0063    "Chat Users Group", "Domain\\ReplaceMe"
#0064  },
#0065  { "AppID\\COMChat.exe",
#0066    "AppId", "{5223A054-2441-11d1-AF4F-0060976AA886}"
#0067  },
#0068  };
#0069
```

```
#0070  // self-unregistration routine
#0071  STDAPI UnregisterServer(void) {
#0072    HRESULT hr = S_OK;
#0073    int nEntries = sizeof(g_RegTable)/sizeof(*g_RegTable);
#0074    for (int i = nEntries - 1; i >= 0; i--){
#0075      const char *pszKeyName = g_RegTable[i][0];
#0076
#0077      long err = RegDeleteKeyA(HKEY_CLASSES_ROOT, pszKeyName);
#0078      if (err != ERROR_SUCCESS)
#0079        hr = S_FALSE;
#0080    }
#0081    return hr;
#0082  }
#0083
#0084  // self-registration routine
#0085  STDAPI RegisterServer(HINSTANCE hInstance = 0) {
#0086    HRESULT hr = S_OK;
#0087  // look up server's file name
#0088    char szFileName[MAX_PATH];
#0089    GetModuleFileNameA(hInstance, szFileName, MAX_PATH);
#0090  // register entries from table
#0091    int nEntries = sizeof(g_RegTable)/sizeof(*g_RegTable);
#0092    for (int i = 0; SUCCEEDED(hr) && i < nEntries; i++) {
#0093      const char *pszKeyName   = g_RegTable[i][0];
#0094      const char *pszValueName = g_RegTable[i][1];
#0095      const char *pszValue     = g_RegTable[i][2];
#0096  // map rogue value to module file name
#0097      if (pszValue == (const char*)-1)
#0098        pszValue = szFileName;
#0099      HKEY hkey;
#0100  // create the key
#0101      long err = RegCreateKeyA(HKEY_CLASSES_ROOT,
#0102                           pszKeyName, &hkey);
#0103      if (err == ERROR_SUCCESS) {
#0104  // set the value
#0105        err = RegSetValueExA(hkey, pszValueName, 0,
#0106                             REG_SZ, (const BYTE*)pszValue,
#0107                             (strlen(pszValue) + 1));
#0108        RegCloseKey(hkey);
#0109      }
#0110      if (err != ERROR_SUCCESS) {
#0111    // if cannot add key or value, back out and fail
#0112        UnregisterServer();
#0113        hr = SELFREG_E_CLASS;
#0114      }
#0115    }
```

```
#0116   return hr;
#0117  }
#0118
#0119  // these point to standard access control objects
#0120  // used to protect particular methods
#0121  IAccessControl *g_pacUsers = 0;
#0122  IAccessControl *g_pacAdmins = 0;
#0123
#0124  // this routine is called at process init time
#0125  // to build access control objects and to allow
#0126  // anonymous access to server by default
#0127  HRESULT InitializeApplicationSecurity(void)
#0128  {
#0129  // load groupnames from registry
#0130     static OLECHAR wszAdminsGroup[1024];
#0131     static OLECHAR wszUsersGroup[1024];
#0132     HKEY hkey;
#0133     long err = RegOpenKeyEx(HKEY_CLASSES_ROOT,
#0134          __TEXT("AppID\\{5223A054-2441-11d1-AF4F-0060976AA886}"),
#0135                          0, KEY_QUERY_VALUE,
#0136                          &hkey);
#0137     if (err == ERROR_SUCCESS)
#0138     {
#0139        DWORD cb = sizeof(wszAdminsGroup);
#0140        err = RegQueryValueExW(hkey, L"Chat Admins Group",
#0141                           0, 0, (BYTE*)wszAdminsGroup,
#0142                           &cb);
#0143        cb = sizeof(wszAdminsGroup);
#0144        if (err == ERROR_SUCCESS)
#0145           err = RegQueryValueExW(hkey,
#0146                                  L"Chat Users Group",
#0147                                  0, 0, (BYTE*)wszUsersGroup,
#0148                                  &cb);
#0149        RegCloseKey(hkey);
#0150     }
#0151     if (err != ERROR_SUCCESS)
#0152        return MAKE_HRESULT(SEVERITY_ERROR, FACILITY_WIN32,
#0153                          GetLastError());
#0154
#0155  // declare vectors of user/groups for 2 access
#0156  // control objects
#0157
#0158     ACTRL_ACCESS_ENTRYW rgaaeUsers[] = {
#0159       { {0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
#0160        TRUSTEE_IS_GROUP, wszUsersGroup  },
#0161        ACTRL_ACCESS_ALLOWED, COM_RIGHTS_EXECUTE, 0,
```

```
#0162         NO_INHERITANCE, 0 },
#0163     };
#0164     ACTRL_ACCESS_ENTRY_LISTW aaelUsers = {
#0165         sizeof(rgaaeUsers)/sizeof(*rgaaeUsers),
#0166         rgaaeUsers
#0167     };
#0168     ACTRL_PROPERTY_ENTRYW apeUsers = { 0, &aaelUsers, 0 };
#0169     ACTRL_ACCESSW aaUsers = { 1, &apeUsers };
#0170
#0171     ACTRL_ACCESS_ENTRYW rgaaeAdmins[] = {
#0172       { {0, NO_MULTIPLE_TRUSTEE, TRUSTEE_IS_NAME,
#0173         TRUSTEE_IS_GROUP, wszAdminsGroup },
#0174         ACTRL_ACCESS_ALLOWED, COM_RIGHTS_EXECUTE, 0,
#0175         NO_INHERITANCE, 0 },
#0176     };
#0177     ACTRL_ACCESS_ENTRY_LISTW aaelAdmins = {
#0178         sizeof(rgaaeAdmins)/sizeof(*rgaaeAdmins),
#0179         rgaaeAdmins
#0180     };
#0181     ACTRL_PROPERTY_ENTRYW apeAdmins = { 0, &aaelAdmins, 0 };
#0182     ACTRL_ACCESSW aaAdmins = { 1, &apeAdmins };
#0183
#0184     HRESULT hr = CoInitializeSecurity(0, -1, 0, 0,
#0185                           RPC_C_AUTHN_LEVEL_NONE,
#0186                           RPC_C_IMP_LEVEL_ANONYMOUS,
#0187                           0,
#0188                           EOAC_NONE,
#0189                           0);
#0190     if (SUCCEEDED(hr))
#0191     {
#0192         hr = CoCreateInstance(CLSID_DCOMAccessControl,
#0193                           0, CLSCTX_ALL, IID_IAccessControl,
#0194                           (void**)&g_pacUsers);
#0195         if (SUCCEEDED(hr))
#0196             hr = g_pacUsers->SetAccessRights(&aaUsers);
#0197         if (SUCCEEDED(hr))
#0198         {
#0199             hr = CoCreateInstance(CLSID_DCOMAccessControl,
#0200                                 0, CLSCTX_ALL,
#0201                                 IID_IAccessControl,
#0202                                 (void**)&g_pacAdmins);
#0203             if (SUCCEEDED(hr))
#0204                 hr = g_pacAdmins->SetAccessRights(&aaAdmins);
#0205         }
#0206         if (FAILED(hr))
#0207         {
```

```
#0208            if (g_pacAdmins)
#0209            {
#0210               g_pacAdmins->Release();
#0211               g_pacAdmins = 0;
#0212            }
#0213            if (g_pacUsers)
#0214            {
#0215               g_pacUsers->Release();
#0216               g_pacUsers = 0;
#0217            }
#0218         }
#0219     }
#0220     return hr;
#0221  }
#0222
#0223  // the main thread routine that simply registers the class
#0224  // object and waits to die
#0225  int WINAPI WinMain(HINSTANCE, HINSTANCE,
#0226                  LPSTR szCmdParam, int)
#0227  {
#0228     const TCHAR *pszPrompt =
#0229         __TEXT("Ensure that you have properly ")
#0230         __TEXT("configured the application to ")
#0231         __TEXT("run as a particular user and that ")
#0232         __TEXT("you have manually changed the ")
#0233         __TEXT("Users and Admins Group registry ")
#0234         __TEXT("settings under this server's AppID.");
#0235
#0236     HRESULT hr = CoInitializeEx(0, COINIT_MULTITHREADED);
#0237     if (FAILED(hr))
#0238        return hr;
#0239
#0240  // look for self-registration flags
#0241     if (strstr(szCmdParam, "/UnregServer") != 0
#0242        || strstr(szCmdParam, "-UnregServer") != 0)
#0243     {
#0244        hr = UnregisterServer();
#0245        CoUninitialize();
#0246        return hr;
#0247     }
#0248     else if (strstr(szCmdParam, "/RegServer") != 0
#0249              || strstr(szCmdParam, "-RegServer") != 0)
#0250     {
#0251        hr = RegisterServer();
#0252        MessageBox(0, pszPrompt, __TEXT("COMChat"),
#0253                  MB_SETFOREGROUND);
```

```
#0254         CoUninitialize();
#0255         return hr;
#0256     }
#0257
#0258  // set up process security
#0259     hr = InitializeApplicationSecurity();
#0260     if (SUCCEEDED(hr))
#0261     {
#0262  // register class object and wait to die
#0263         DWORD  dwReg;
#0264         static ChatSessionClass cmc;
#0265         hr = CoRegisterClassObject(CLSID_ChatSession,
#0266              static_cast<IExternalConnection*>(&cmc),
#0267                                 CLSCTX_LOCAL_SERVER
#0268                 REGCLS_SUSPENDED|REGCLS_MULTIPLEUSE,
#0269                                 &dwReg);
#0270         if (SUCCEEDED(hr))
#0271         {
#0272             hr = CoResumeClassObjects();
#0273             if (SUCCEEDED(hr))
#0274                 WaitForSingleObject(g_heventDone, INFINITE);
#0275             CoRevokeClassObject(dwReg);
#0276         }
#0277         g_pacUsers->Release();
#0278         g_pacAdmins->Release();
#0279     }
#0280     if (FAILED(hr))
#0281         MessageBox(0, pszPrompt, __TEXT("Error"),
#0282                  MB_SETFOREGROUND);
#0283
#0284     CoUninitialize();
#0285     return 0;
#0286  }
```