```
1 #define UNICODE
 2 #include<windows.h>
 3 #include"AutomationServerWithRegFile.h"
 4 // global function declarations
 5 LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
 6 // class declarations
 7 class CSum:public ISum
 8 {
9 private:
10
       long m cRef;
       11
12 public:
       // constructor method declarations
13
14
       CSum(void);
       // destructor method declarations
15
       ~CSum(void):
16
       // IUnknown specific method declarations (inherited)
17
       HRESULT stdcall QueryInterface(REFIID, void **);
18
       ULONG __stdcall AddRef(void);
19
       ULONG __stdcall Release(void);
20
       // IDispatch specific method declarations (inherited) //
21
                                                                                    D
22
       HRESULT __stdcall GetTypeInfoCount(UINT*);
23
       HRESULT __stdcall GetTypeInfo(UINT,LCID,ITypeInfo**);
       HRESULT stdcall GetIDsOfNames(REFIID, LPOLESTR*, UINT, LCID, DISPID*);
24
       HRESULT stdcall Invoke
25
         (DISPID, REFIID, LCID, WORD, DISPPARAMS*, VARIANT*, EXCEPINFO*, UINT*);
26
       // ISum specific method declarations (inherited)
       HRESULT stdcall SumOfTwoIntegers(int,int);
27
       // custom methods
28
29
       HRESULT InitInstance(HINSTANCE);
31 class CSumClassFactory:public IClassFactory
32 {
33 private:
       long m cRef;
35 public:
       // constructor method declarations
36
       CSumClassFactory(void);
37
38
       // destructor method declarations
39
       ~CSumClassFactory(void);
40
       // IUnknown specific method declarations (inherited)
       HRESULT stdcall QueryInterface(REFIID, void **);
41
42
       ULONG __stdcall AddRef(void);
43
       ULONG    stdcall Release(void);
44
       // IClassFactory specific method declarations (inherited)
       HRESULT stdcall CreateInstance(IUnknown *, REFIID, void **);
45
       HRESULT __stdcall LockServer(BOOL);
46
47 };
48 // global variable declarations
49 long glNumberOfActiveComponents=0;// number of active components
50 long glNumberOfServerLocks=0;// number of locks on this dll
```

```
...tomationServerWithRegFile\AutomationServerWithRegFile.cpp
```

```
51 // 917898EA-9D21-4a85-81A6-DA523D483833
52 const GUID LIBID AutomationServer=
                                                                                D
      {0x917898ea,0x9d21,0x4a85,0x81,0xa6,0xda,0x52,0x3d,0x48,0x38,0x33};
53 CSum *gpCSum=NULL;// ******
54 IClassFactory *gpIClassFactory=NULL;
55 HWND ghwnd=NULL;
57 DWORD dwRegisterActiveObject;// *********************
58 // WinMain
59 int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
60
                     LPSTR lpCmdLine, int nCmdShow)
61 {
62
       // function declarations
63
       HRESULT InitInstance(HINSTANCE);
       HRESULT StartMyClassFactories(void);
64
65
       void StopMyClassFactories(void);
66
       // variable declarations
       WNDCLASSEX wndclass;
67
68
       MSG msg;
69
       HWND hwnd;
70
       HRESULT hr;
71
       int DontShowWindow=0;// 0 means show the window
       72
73
       TCHAR szTokens[]=TEXT("-/");
74
       TCHAR *pszTokens;
75
       TCHAR lpszCmdLine[255];
76
       // com library initialization
77
       hr=CoInitialize(NULL);
78
       if(FAILED(hr))
79
           return(0);
       MultiByteToWideChar(CP_ACP,0,lpCmdLine,255,lpszCmdLine,255);
80
       pszTokens=wcstok(lpszCmdLine,szTokens);
81
82
       while(pszTokens!=NULL)
83
       {
84
           // COM is calling me with Automation
85
           if(wcsicmp(pszTokens,TEXT("Embedding"))==0)
86
               DontShowWindow=1;// dont show window but message loop must
87
               break;
88
89
           }
90
           else
91
           {
92
               MessageBox(NULL, TEXT("Bad Command Line Arguments.\nExitting The
                 Application."),TEXT("Error"),MB_OK);
93
               exit(0);
94
           }
95
           pszTokens=wcstok(NULL,szTokens);
96
97
       // window code
98
       wndclass.cbSize=sizeof(wndclass);
       wndclass.style=CS_HREDRAW|CS_VREDRAW;
99
       wndclass.cbClsExtra=0;
100
```

```
...tomationServerWithRegFile\AutomationServerWithRegFile.cpp
```

```
3
```

```
101
        wndclass.cbWndExtra=0;
102
        wndclass.lpfnWndProc=WndProc;
        103
104
        wndclass.hCursor=LoadCursor(NULL,IDC ARROW);
105
        wndclass.hbrBackground=(HBRUSH)GetStockObject(BLACK BRUSH);
        wndclass.hInstance=hInstance;
196
107
        wndclass.lpszClassName=AppName;
108
        wndclass.lpszMenuName=NULL;
109
        wndclass.hiconSm=Loadicon(hinstance,TEXT("APPICON"));// *************************
110
        // register window class
111
        RegisterClassEx(&wndclass);
112
        // create window
        hwnd=CreateWindow(AppName,
113
                          TEXT("Exe Server With Reg File"),// *************
114
115
                          WS OVERLAPPEDWINDOW,
                          CW USEDEFAULT,
116
117
                          CW USEDEFAULT,
118
                          CW USEDEFAULT,
119
                          CW USEDEFAULT.
120
                          NULL,
121
                          NULL,
122
                          hInstance,
123
                          NULL);
124
        // initialize global window handle
125
        ghwnd=hwnd;
        if(DontShowWindow!=1)
126
127
        {
128
            // usual functions
            ShowWindow(hwnd, SW MAXIMIZE);// *********
129
130
            UpdateWindow(hwnd);
131
            // increament server lock
            ++glNumberOfServerLocks;
132
133
        if(DontShowWindow==1)// only when COM calls this program ****************
134
135
136
            // initialize the global instance of main object
137
            gpCSum=new CSum;
            if(gpCSum==NULL)
138
139
140
                MessageBox(hwnd,TEXT("Main Component Can Not Be Created.\nMemory
                  Problem !!!"),TEXT("Error"),MB OK MB ICONERROR MB TOPMOST);
141
                DestroyWindow(hwnd);
            }
142
143
            hr=gpCSum->InitInstance(hInstance);
144
            if(FAILED(hr))
145
            {
                MessageBox(hwnd, TEXT("Main Component's Type Library Can Not Be
146
                  Initialized."), TEXT("Error"), MB_OK | MB_ICONERROR | MB_TOPMOST);
147
                DestroyWindow(hwnd);
148
            }
            // start class factory
149
150
            hr=StartMyClassFactories();
```

199

```
...tomationServerWithRegFile\AutomationServerWithRegFile.cpp
                                                                                        4
151
             if(FAILED(hr))
152
             {
                 if(gpCSum)// *************
153
154
                 {
155
                     gpCSum->Release();
                     gpCSum=NULL;
156
                 }
157
                 MessageBox(hwnd, TEXT("Main Component's Class Factory Can Not Be
158
                                                                                        P
                   Started."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
                 DestroyWindow(hwnd);
159
160
161
             // register the global object (created by InitInstance()) //
                                                                                        P
162
             hr=RegisterActiveObject(reinterpret cast<IUnknown *>(gpCSum),
                                     CLSID SumAutomation,
163
                                     ACTIVEOBJECT_WEAK, // ********* whv ?
164
                                     &dwRegisterActiveObject);
165
            if(FAILED(hr))
166
167
168
                 if(gpIClassFactory)
169
                 {
170
                     gpIClassFactory->Release();
171
                     gpIClassFactory=NULL;
172
173
                 if(gpCSum)
174
175
                     gpCSum->Release();
176
                     gpCSum=NULL;
177
                 }
                 MessageBox(hwnd, TEXT("Main Component's Active Instance Can Not Be
178
                   Registered."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
                 DestroyWindow(hwnd);
179
180
             }
181
         }
         // message loop
182
183
         while(GetMessage(&msg,NULL,0,0))
184
         {
185
             TranslateMessage(&msg);
            DispatchMessage(&msg);
186
187
188
         if(DontShowWindow==1)// only when COM calls this program
189
             // un-register global class factory object
190
191
            StopMyClassFactories();
             // un-register global main object // **************
192
             if(dwRegisterActiveObject!=0)
193
194
                 RevokeActiveObject(dwRegisterActiveObject,NULL);
195
196
         // com library un-initialization
197
         CoUninitialize();
         return((int)msg.wParam);
198
```

```
200 }
201 // Window Procedure
202 LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM 1Param)
203 {
204
        // variable declarations
205
        HDC hdc;
206
        RECT rc;
207
        PAINTSTRUCT ps;
208
        // code
209
        switch(iMsg)
210
        {
211
        case WM PAINT:
212
            GetClientRect(hwnd,&rc);
213
            hdc=BeginPaint(hwnd,&ps);
214
            SetBkColor(hdc,RGB(0,0,0));
215
            SetTextColor(hdc,RGB(0,255,0));
216
            DrawText(hdc,
                     TEXT("This
217
                                 Is
                                      A
                                           COM Exe
                                                       Automation
                                                                    Server
                                                                             Program. >
                               For
                                           !!!!"),
                          Not
                                     You
218
                     -1,
219
                     &rc.
220
                     DT SINGLELINE DT CENTER DT VCENTER);
221
            EndPaint(hwnd,&ps);
222
            break:
223
        case WM DESTROY:
            if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
224
225
                PostQuitMessage(0);
226
            break;
        case WM CLOSE:
227
228
            --glNumberOfServerLocks;
229
            ShowWindow(hwnd, SW_HIDE);
            // fall through, hence no break
230
231
        default:
232
            return(DefWindowProc(hwnd,iMsg,wParam,lParam));
233
        }
234
        return(0L);
235 }
236 // Implementation Of CSum's Constructor Method
237 CSum::CSum(void)
238 {
239
        // code
240
        m cRef=1;// hardcoded initialization to anticipate possible failure of
241
          QueryInterface()
        InterlockedIncrement(&glNumberOfActiveComponents);// increment global counter
242
243 }
244 // Implementation Of CSum's Destructor Method
245 CSum::~CSum(void)
246 {
247
        // code
        InterlockedDecrement(&glNumberOfActiveComponents);// decrement global counter
248
249
```

```
250 // Implementation Of CSum's IUnknown's Methods
251 HRESULT CSum::QueryInterface(REFIID riid, void **ppv)
252 {
253
        // code
254
        if(riid==IID IUnknown)
255
            *ppv=static cast<ISum *>(this);
        else if(riid==IID_IDispatch)// *
256
257
            *ppv=static cast<ISum *>(this);
258
        else if(riid==IID_ISum)
            *ppv=static cast<ISum *>(this);
259
260
        else
261
        {
262
            *ppv=NULL;
263
            return(E NOINTERFACE);
264
        }
265
        reinterpret cast<IUnknown *>(*ppv)->AddRef();
266
        return(S OK);
267 }
268 ULONG CSum::AddRef(void)
269 {
270
        // code
271
        InterlockedIncrement(&m cRef);
272
        return(m_cRef);
273 }
274 ULONG CSum::Release(void)
275 {
276
        // code
277
        InterlockedDecrement(&m cRef);
        if(m_cRef==0)
278
279
        {
280
            delete(this);
            if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
281
282
                PostMessage(ghwnd, WM_QUIT, (WPARAM)0, (LPARAM)0L);
283
            return(0);
284
        }
285
        return(m cRef);
286 }
287 // Implementation Of ISum's Methods
289 {
290
        // variable declarations
291
        int num3;
292
        TCHAR szSum[255];
293
        // code
294
        num3=num1+num2;
        wsprintf(szSum,TEXT("Automation Server Gives You Sum Of %d And %d As %
295
                                                                                    P
          d"),num1,num2,num3);
296
        MessageBox(NULL,szSum,TEXT("Automation Server"),MB_OK);
297
        return(S_OK);
298 }
299 // Implementation Of CSumClassFactory's Constructor Method
300 CSumClassFactory::CSumClassFactory(void)
```

```
...tomationServerWithRegFile\AutomationServerWithRegFile.cpp
```

```
301 {
302
         // code
         m cRef=1;// hardcoded initialization to anticipate possible failure of
303
                                                                                        P
          QueryInterface()
304 }
305 // Implementation Of CSumClassFactory's Destructor Method
306 CSumClassFactory::~CSumClassFactory(void)
307 {
308
         // code
309 }
310 // Implementation Of CSumClassFactory's IClassFactory's IUnknown's Methods
311 HRESULT CSumClassFactory::QueryInterface(REFIID riid, void **ppv)
312 {
313
         // code
        if(riid==IID IUnknown)
314
315
             *ppv=static cast<IClassFactory *>(this);
         else if(riid==IID IClassFactory)
316
             *ppv=static cast<IClassFactory *>(this);
317
318
         else
319
         {
             *ppv=NULL;
320
321
             return(E NOINTERFACE);
322
         reinterpret_cast<IUnknown *>(*ppv)->AddRef();
323
324
         return(S OK);
325 }
326 ULONG CSumClassFactory::AddRef(void)
327 {
         // code
328
329
         InterlockedIncrement(&m cRef);
330
         return(m_cRef);
331
332 ULONG CSumClassFactory::Release(void)
333 {
334
         // code
335
         InterlockedDecrement(&m cRef);
336
        if(m cRef==0)
337
         {
             delete(this);
338
339
             return(0);
340
341
         return(m_cRef);
342 }
343 // Implementation Of CSumClassFactory's IClassFactory's Methods
344 HRESULT CSumClassFactory::CreateInstance(IUnknown *pUnkOuter, REFIID riid, void
       **ppv)
345 {
346
         // variable declarations
347
        HRESULT hr:
348
        // code
        if(pUnkOuter!=NULL)
349
             return(CLASS E NOAGGREGATION);
350
```

```
...tomationServerWithRegFile\AutomationServerWithRegFile.cpp
```

```
8
```

```
// ********** new
351
352
        // object is already created in WinMain(), just call QI() to get requested
                                                                                  D
353
        hr=gpCSum->QueryInterface(riid,ppv);
354
        gpCSum->Release();// anticipate possible failure of QueryInterface()
355
        return(hr);
356 }
357 HRESULT CSumClassFactory::LockServer(BOOL fLock)
358 {
359
        // code
360
        if(fLock)
361
            InterlockedIncrement(&glNumberOfServerLocks);
362
        else
363
            InterlockedDecrement(&glNumberOfServerLocks);
        if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
364
365
            PostMessage(ghwnd, WM QUIT, (WPARAM)0, (LPARAM)0L);
366
        return(S OK);
367 }
369 // Implementation Of CSum's IDispatch's Methods
370 HRESULT CSum::GetTypeInfoCount(UINT *pCountTypeInfo)
371 {
372
373
        *pCountTypeInfo=1;// as we have only one method SumOfTwoIntegers()
374
        return(S OK);
375 }
376 HRESULT CSum::GetTypeInfo(UINT iTypeInfo,LCID lcid,ITypeInfo **ppITypeInfo)
377 {
378
        // code
379
        *ppITypeInfo=NULL;
380
        if(iTypeInfo!=0)
            return(DISP E BADINDEX);
381
382
        m pITypeInfo->AddRef();
383
        *ppITypeInfo=m pITypeInfo;
384
        return(S_OK);
385 }
386 HRESULT CSum::GetIDsOfNames(REFIID riid, LPOLESTR *rgszNames, UINT cNames, LCID
      lcid,DISPID *rgDispId)
387 {
388
        // code
389
        return(DispGetIDsOfNames(m pITypeInfo,rgszNames,cNames,rgDispId));
390 }
391 HRESULT CSum::Invoke(DISPID dispIdMember, REFIID riid, LCID lcid, WORD
      wFlags_DISPPARAMS *pDispParams_VARIANT *pVarResult_EXCEPINFO *pExcepInfo_UINT
      *puArgErr)
392 {
393
        // variable declarations
        HRESULT hr;
394
395
        // code
396
        hr=DispInvoke(this,
397
                     m_pITypeInfo,
398
                     dispIdMember,
```

```
399
                      wFlags,
400
                      pDispParams,
401
                      pVarResult,
492
                      pExcepInfo,
403
                      puArgErr);
404
        return(hr);
405 }
406 // custom methods
407 HRESULT CSum::InitInstance(HINSTANCE hInst)
408 {
        // variable declarations
409
410
        HRESULT hr;
411
        ITypeLib *pITypeLib=NULL;
412
        TCHAR szExeFileName[ MAX PATH],szTypeLibPath[ MAX PATH];
413
        // code
414
        if(m_pITypeInfo==NULL)
415
        {
            hr=LoadRegTypeLib(LIBID_AutomationServer,
416
417
                              1,0,// major/minor version numbers
418
                              0x00,
419
                              &pITypeLib);
420
            if(FAILED(hr))
421
422
                GetModuleFileName(hInst,szExeFileName,_MAX_PATH);
423
                wsprintf(szTypeLibPath,TEXT("%s\\1"),szExeFileName);
424
                hr=LoadTypeLib(szTypeLibPath,&pITypeLib);
425
                if(FAILED(hr))
426
                    return(hr);
427
                hr=RegisterTypeLib(pITypeLib,szTypeLibPath,NULL);
428
                if(FAILED(hr))
429
                    return(hr);
430
431
            hr=pITypeLib->GetTypeInfoOfGuid(IID_ISum,&m_pITypeInfo);
432
            if(FAILED(hr))
433
            {
434
                pITypeLib->Release();
435
                return(hr);
436
437
            pITypeLib->Release();
438
        }
439
        return(S OK);
440
    // ******* new ends
441
                                                                                     P
      ***********
442 // other methods
443 HRESULT StartMyClassFactories(void)
444 {
445
        // variable declaraions
446
        HRESULT hr:
447
        // code
        gpIClassFactory=new CSumClassFactory;
448
449
        if(gpIClassFactory==NULL)
```

```
... to mation Server With Reg File \setminus Automation Server With Reg File. cpp
```

```
10
```

```
450
            return(E OUTOFMEMORY);
451
        gpIClassFactory->AddRef();
452
        // register the class factory
453
        hr=CoRegisterClassObject(CLSID_SumAutomation,
454
                                  static_cast<IUnknown *>(gpIClassFactory),
455
                                  CLSCTX LOCAL SERVER,
                                  REGCLS_SINGLEUSE,// ************** whv ?
456
                                  &dwRegisterClassFactory);// ********* just
457
                                                                                       P
                         renamed
458
        if(FAILED(hr))
459
        {
460
             gpIClassFactory->Release();
461
            return(E_FAIL);
462
        }
463
        return(S_OK);
464 }
465 void StopMyClassFactories(void)
466 {
467
        // code
468
        // un-register the class factory
469
        if(dwRegisterClassFactory!=0)
470
            CoRevokeClassObject(dwRegisterClassFactory);
471
        if(gpIClassFactory!=NULL)
472
            gpIClassFactory->Release();
473 }
474
```