

```

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     ITypeInfo *m_pTypeInfo;// *****
12 public:
13     // constructor method declarations
14     CSum(void);
15     // destructor method declarations
16     ~CSum(void);
17     // IUnknown specific method declarations (inherited)
18     HRESULT __stdcall QueryInterface(REFIID,void **);
19     ULONG __stdcall AddRef(void);
20     ULONG __stdcall Release(void);
21     // IDispatch specific method declarations (inherited) //
        *****
22     HRESULT __stdcall GetTypeInfoCount(UINT*);
23     HRESULT __stdcall GetTypeInfo(UINT,LCID,ITypeInfo**);
24     HRESULT __stdcall GetIDsOfNames(REFIID,LPOLESTR*,UINT,LCID,DISPID*);
25     HRESULT __stdcall Invoke
        (DISPID,REFIID,LCID,WORD,DISPPARAMS*,VARIANT*,EXCEPINFO*,UINT*);
26     // ISum specific method declarations (inherited)
27     HRESULT __stdcall SumOfTwoIntegers(int,int);
28     // custom methods
29     HRESULT InitInstance(HINSTANCE);
30 };
31 class CSumClassFactory:public IClassFactory
32 {
33 private:
34     long m_cRef;
35 public:
36     // constructor method declarations
37     CSumClassFactory(void);
38     // destructor method declarations
39     ~CSumClassFactory(void);
40     // IUnknown specific method declarations (inherited)
41     HRESULT __stdcall QueryInterface(REFIID,void **);
42     ULONG __stdcall AddRef(void);
43     ULONG __stdcall Release(void);
44     // IClassFactory specific method declarations (inherited)
45     HRESULT __stdcall CreateInstance(IUnknown *,REFIID,void **);
46     HRESULT __stdcall LockServer(BOOL);
47 };
48 // global variable declarations
49 long g1NumberOfActiveComponents=0;// number of active components
50 long g1NumberOfServerLocks=0;// number of locks on this dll

```

```

51 // 917898EA-9D21-4a85-81A6-DA523D483833
52 const GUID LIBID_AutomationServer=
    {0x917898ea,0x9d21,0x4a85,0x81,0xa6,0xda,0x52,0x3d,0x48,0x38,0x33};
53 CSum *gpCSum=NULL; // *****
54 IClassFactory *gpIClassFactory=NULL;
55 HWND ghwnd=NULL;
56 DWORD dwRegisterClassFactory; // ***** just renamed
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);
64     HRESULT StartMyClassFactories(void);
65     void StopMyClassFactories(void);
66     // variable declarations
67     WNDCLASSEX wndclass;
68     MSG msg;
69     HWND hwnd;
70     HRESULT hr;
71     int DontShowWindow=0; // 0 means show the window
72     TCHAR AppName[]=TEXT("ExeAutomationServer"); // *****
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);
80     MultiByteToWideChar(CP_ACP,0,lpCmdLine,255,lpszCmdLine,255);
81     pszTokens=wcstok(lpszCmdLine,szTokens);
82     while(pszTokens!=NULL)
83     {
84         // COM is calling me with Automation
85         if(wcsicmp(pszTokens,TEXT("Embedding"))==0)
86         {
87             DontShowWindow=1; // dont show window but message loop must
88             break;
89         }
90         else
91         {
92             MessageBox(NULL,TEXT("Bad Command Line Arguments.\nExiting The
93                 Application."),TEXT("Error"),MB_OK);
94             exit(0);
95         }
96         pszTokens=wcstok(NULL,szTokens);
97     }
98     // window code
99     wndclass.cbSize=sizeof(wndclass);
100    wndclass.style=CS_HREDRAW|CS_VREDRAW;
    wndclass.cbClsExtra=0;

```

```

101     wndclass.cbWndExtra=0;
102     wndclass.lpfnWndProc=WndProc;
103     wndclass.hIcon=LoadIcon(hInstance,TEXT("APPICON"));// *****
104     wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
105     wndclass.hbrBackground=(HBRUSH)GetStockObject(BLACK_BRUSH);
106     wndclass.hInstance=hInstance;
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
113     hwnd=CreateWindow(AppName,
114                       TEXT("Exe Server With Reg File"),// *****
115                       WS_OVERLAPPEDWINDOW,
116                       CW_USEDEFAULT,
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;
126     if(DontShowWindow!=1)
127     {
128         // usual functions
129         ShowWindow(hwnd,SW_MAXIMIZE);// *****
130         UpdateWindow(hwnd);
131         // increament server lock
132         ++glNumberOfServerLocks;
133     }
134     if(DontShowWindow==1)// only when COM calls this program *****
135     {
136         // initialize the global instance of main object
137         gpCSum=new CSum;
138         if(gpCSum==NULL)
139         {
140             MessageBox(hwnd,TEXT("Main Component Can Not Be Created.\nMemory
141                               Problem !!!"),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
142             DestroyWindow(hwnd);
143         }
144         hr=gpCSum->InitInstance(hInstance);
145         if(FAILED(hr))
146         {
147             MessageBox(hwnd,TEXT("Main Component's Type Library Can Not Be
148                               Initialized."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
149             DestroyWindow(hwnd);
150         }
151         // start class factory
152         hr=StartMyClassFactories();

```



```

151     if(FAILED(hr))
152     {
153         if(gpCSum// *****
154         {
155             gpCSum->Release();
156             gpCSum=NULL;
157         }
158         MessageBox(hwnd,TEXT("Main Component's Class Factory Can Not Be
159             Started."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
160         DestroyWindow(hwnd);
161     }
162     // register the global object (created by InitInstance()) //
163     *****
164     hr=RegisterActiveObject(reinterpret_cast<IUnknown *>(gpCSum),
165         CLSID_SumAutomation,
166         ACTIVEOBJECT_WEAK,// ***** why ?
167         &dwRegisterActiveObject);
168     if(FAILED(hr))
169     {
170         if(gpIClassFactory)
171         {
172             gpIClassFactory->Release();
173             gpIClassFactory=NULL;
174         }
175         if(gpCSum)
176         {
177             gpCSum->Release();
178             gpCSum=NULL;
179         }
180         MessageBox(hwnd,TEXT("Main Component's Active Instance Can Not Be
181             Registered."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
182         DestroyWindow(hwnd);
183     }
184 }
185 // message loop
186 while(GetMessage(&msg,NULL,0,0))
187 {
188     TranslateMessage(&msg);
189     DispatchMessage(&msg);
190 }
191 if(DontShowWindow==1)// only when COM calls this program
192 {
193     // un-register global class factory object
194     StopMyClassFactories();
195     // un-register global main object // *****
196     if(dwRegisterActiveObject!=0)
197         RevokeActiveObject(dwRegisterActiveObject,NULL);
198 }
199 // com library un-initialization
200 CoUninitialize();
201 return((int)msg.wParam);
202

```

```

200 }
201 // Window Procedure
202 LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,LPARAM lParam)
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,
217             TEXT("This Is A COM Exe Automation Server Program. ➤
                Not For You !!!"),
218             -1,
219             &rc,
220             DT_SINGLELINE|DT_CENTER|DT_VCENTER);
221         EndPaint(hwnd,&ps);
222         break;
223     case WM_DESTROY:
224         if(g1NumberOfActiveComponents==0 && g1NumberOfServerLocks==0)
225             PostQuitMessage(0);
226         break;
227     case WM_CLOSE:
228         --g1NumberOfServerLocks;
229         ShowWindow(hwnd,SW_HIDE);
230         // fall through,hence no break
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_pITypeInfo=NULL;// *****
241     m_cRef=1;// hardcoded initialization to anticipate possible failure of ➤
242         QueryInterface()
243     InterlockedIncrement(&g1NumberOfActiveComponents);// increment global counter
244 }
245 // Implementation Of CSum's Destructor Method
246 CSum::~CSum(void)
247 {
248     // code
249     InterlockedDecrement(&g1NumberOfActiveComponents);// decrement global counter
250 }

```

```

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);
256     else if(riid==IID_IDispatch)// *****
257         *ppv=static_cast<ISum *>(this);
258     else if(riid==IID_ISum)
259         *ppv=static_cast<ISum *>(this);
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);
278     if(m_cRef==0)
279     {
280         delete(this);
281         if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
282             PostMessage(ghwnd,WM_QUIT,(LPARAM)0,(LPARAM)0L);
283         return(0);
284     }
285     return(m_cRef);
286 }
287 // Implementation Of ISum's Methods
288 HRESULT CSum::SumOfTwoIntegers(int num1,int num2)// *****
289 {
290     // variable declarations
291     int num3;
292     TCHAR szSum[255];
293     // code
294     num3=num1+num2;
295     wsprintf(szSum,TEXT("Automation Server Gives You Sum Of %d And %d As %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)

```

```

301 {
302     // code
303     m_cRef=1; // hardcoded initialization to anticipate possible failure of
        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
314     if(riid==IID_IUnknown)
315         *ppv=static_cast<IClassFactory*>(this);
316     else if(riid==IID_IClassFactory)
317         *ppv=static_cast<IClassFactory*>(this);
318     else
319     {
320         *ppv=NULL;
321         return(E_NOINTERFACE);
322     }
323     reinterpret_cast<IUnknown*>(*ppv)->AddRef();
324     return(S_OK);
325 }
326 ULONG CSumClassFactory::AddRef(void)
327 {
328     // code
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     {
338         delete(this);
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
349     if(pUnkOuter!=NULL)
350         return(CLASS_E_NOAGGREGATION);

```



```

351 // ***** new
352 // object is already created in WinMain(), just call QI() to get requested  ➤
    interface
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);
364 if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
365     PostMessage(ghwnd,WM_QUIT,(WPARAM)0,(LPARAM)0L);
366 return(S_OK);
367 }
368 // ***** new starts *****
369 // Implementation Of Csum's IDispatch's Methods
370 HRESULT Csum::GetTypeInfoCount(UINT *pCountTypeInfo)
371 {
372 // code
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)
381     return(DISP_E_BADINDEX);
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
394 HRESULT hr;
395 // code
396 hr=DispInvoke(this,
397               m_pITypeInfo,
398               dispIdMember,

```



```

399         wFlags,
400         pDispParams,
401         pVarResult,
402         pExcepInfo,
403         puArgErr);
404     return(hr);
405 }
406 // custom methods
407 HRESULT CSum::InitInstance(HINSTANCE hInst)
408 {
409     // variable declarations
410     HRESULT hr;
411     ITypeLib *pITypeLib=NULL;
412     TCHAR szExeFileName[_MAX_PATH],szTypeLibPath[_MAX_PATH];
413     // code
414     if(m_pTypeInfo==NULL)
415     {
416         hr=LoadRegTypeLib(LIBID_AutomationServer,
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_pTypeInfo);
432         if(FAILED(hr))
433         {
434             pITypeLib->Release();
435             return(hr);
436         }
437         pITypeLib->Release();
438     }
439     return(S_OK);
440 }
441 // ***** new ends
442 // other methods
443 HRESULT StartMyClassFactories(void)
444 {
445     // variable declarations
446     HRESULT hr;
447     // code
448     gpIClassFactory=new CSumClassFactory;
449     if(gpIClassFactory==NULL)

```

```
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,
456                             REGCLS_SINGLEUSE, // ***** why ?
457                             &dwRegisterClassFactory); // ***** just
458                                     renamed
459     if(FAILED(hr))
460     {
461         gpIClassFactory->Release();
462         return(E_FAIL);
463     }
464     return(S_OK);
465 }
466 void StopMyClassFactories(void)
467 {
468     // code
469     // un-register the class factory
470     if(dwRegisterClassFactory!=0)
471         CoRevokeClassObject(dwRegisterClassFactory);
472     if(gpIClassFactory!=NULL)
473         gpIClassFactory->Release();
474 }
```