```c
#define UNICODE
#include<windows.h>
#include"AutomationServerWithRegFile.h"
// global function declarations
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
// WinMain
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,int nCmdShow)
{
    // variable declarations
    WNDCLASSEX wndclass;
    HWND hwnd;
    MSG msg;
    TCHAR AppName[]=TEXT("Client");
    // code
    wndclass.cbSize=sizeof(wndclass);
    wndclass.style=CS_HREDRAW|CS_VREDRAW;
    wndclass.cbClsExtra=0;
    wndclass.cbWndExtra=0;
    wndclass.lpfnWndProc=WndProc;
    wndclass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.hInstance=hInstance;
    wndclass.lpszClassName=AppName;
    wndclass.lpszMenuName=NULL;
    wndclass.hIconSm=LoadIcon(NULL,IDI_APPLICATION);
    // register window class
    RegisterClassEx(&wndclass);
    // create window
    hwnd=CreateWindow(AppName,
                      TEXT("Client Of Exe Server"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      hInstance,
                      NULL);
    ShowWindow(hwnd,nCmdShow);
    UpdateWindow(hwnd);
    // message loop
    while(GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return(msg.wParam);
}
// Window Procedure
```

```
53  LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,LPARAM lParam)
54  {
55      // variable declarations
56      ISum *pIDispatch=NULL;
57      HRESULT hr;
58      DISPID dispid;
59      OLECHAR *szFunctionName=L"SumOfTwoIntegers";
60      VARIANT varg[2];
61      DISPPARAMS param={NULL,NULL,0,2};
62      int n1,n2;
63      // code
64      switch(iMsg)
65      {
66      case WM_CREATE:
67          // initialize COM library
68          hr=CoInitialize(NULL);
69          if(FAILED(hr))
70          {
71              MessageBox(hwnd,TEXT("COM library can not be initialized"),TEXT("COM
                  Error"),MB_OK);
72              DestroyWindow(hwnd);
73              exit(0);
74          }
75          // get ISum Interface
76          hr=CoCreateInstance(&CLSID_SumAutomation,
77                              NULL,
78                              CLSCTX_LOCAL_SERVER,
79                              &IID_IDispatch,
80                              (void **)&pIDispatch);
81          if(FAILED(hr))
82          {
83              MessageBox(hwnd,TEXT("Component Can Not Be Created"),TEXT("COM
                  Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
84              DestroyWindow(hwnd);
85              exit(0);
86          }
87          hr=pIDispatch->lpVtbl->GetIDsOfNames(pIDispatch,
88                                               &IID_NULL,
89                                               &szFunctionName,
90                                               1,
91                                               GetUserDefaultLCID(),
92                                               &dispid);
93          if(FAILED(hr))
94          {
95              MessageBox(NULL,TEXT("Can Not Get ID For Function"),TEXT
                  ("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
96              pIDispatch->lpVtbl->Release(pIDispatch);
97              DestroyWindow(hwnd);
98          }
99          n1=800;
100         n2=200;
101         // as DISPPARAMS rgvarg member receives parameters in reverse order
```

```
102         VariantInit(varg);
103         varg[0].vt=VT_INT;
104         varg[0].intVal=n2;
105         varg[1].vt=VT_INT;
106         varg[1].intVal=n1;
107         param.cArgs=2;
108         param.cNamedArgs=0;
109         param.rgdispidNamedArgs=NULL;
110         // reverse order of parameters
111         param.rgvarg=varg;
112         hr=pIDispatch->lpVtbl->Invoke(pIDispatch,
113                                       dispid,
114                                       &IID_NULL,
115                                       GetUserDefaultLCID(),
116                                       DISPATCH_METHOD,
117                                       &param,
118                                       NULL,
119                                       NULL,
120                                       NULL);
121         if(FAILED(hr))
122         {
123             MessageBox(NULL,TEXT("Can Not Invoke Function"),TEXT("Error"),MB_OK|
                MB_ICONERROR|MB_TOPMOST);
124             pIDispatch->lpVtbl->Release(pIDispatch);
125             DestroyWindow(hwnd);
126         }
127         VariantClear(varg);
128         pIDispatch->lpVtbl->Release(pIDispatch);
129         DestroyWindow(hwnd);
130         break;
131     case WM_DESTROY:
132         CoUninitialize();
133         PostQuitMessage(0);
134         break;
135     }
136     return(DefWindowProc(hwnd,iMsg,wParam,lParam));
137 }
138
```

```
1  LIBRARY AutomationProxyStub.dll
2  DESCRIPTION 'MIDL Generated Proxy/Stub Dll File'
3  EXPORTS
4        DllGetClassObject   @1 PRIVATE
5        DllCanUnloadNow     @2 PRIVATE
6        GetProxyDllInfo     @3 PRIVATE
7        DllRegisterServer   @4 PRIVATE
8        DllUnregisterServer @5 PRIVATE
9
```

```
1 /h AutomationProxyStubHeader.h
2 /iid AutomationProxyStubGuids.c
3 /dlldata AutomationProxyStubDlldata.c
4 /proxy AutomationProxyStub.c
5 /out E:\Automation\AutomationServerWithRegFile\AutomationProxyStub
6 E:\Automation\AutomationServerWithRegFile\AutomationProxyStub
    \VDGAutomationServerTypeLib.idl
7
```

```
#define UNICODE
#include<windows.h>
#include"AutomationServerWithRegFile.h"
// global function declarations
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
// class declarations
class CSum:public ISum
{
private:
    long m_cRef;
    ITypeInfo *m_pITypeInfo;// **********************
public:
    // constructor method declarations
    CSum(void);
    // destructor method declarations
    ~CSum(void);
    // IUnknown specific method declarations (inherited)
    HRESULT __stdcall QueryInterface(REFIID,void **);
    ULONG __stdcall AddRef(void);
    ULONG __stdcall Release(void);
    // IDispatch specific method declarations (inherited) //
      ************************
    HRESULT __stdcall GetTypeInfoCount(UINT*);
    HRESULT __stdcall GetTypeInfo(UINT,LCID,ITypeInfo**);
    HRESULT __stdcall GetIDsOfNames(REFIID,LPOLESTR*,UINT,LCID,DISPID*);
    HRESULT __stdcall Invoke
      (DISPID,REFIID,LCID,WORD,DISPPARAMS*,VARIANT*,EXCEPINFO*,UINT*);
    // ISum specific method declarations (inherited)
    HRESULT __stdcall SumOfTwoIntegers(int,int);
    // custom methods
    HRESULT InitInstance(HINSTANCE);
};
class CSumClassFactory:public IClassFactory
{
private:
    long m_cRef;
public:
    // constructor method declarations
    CSumClassFactory(void);
    // destructor method declarations
    ~CSumClassFactory(void);
    // IUnknown specific method declarations (inherited)
    HRESULT __stdcall QueryInterface(REFIID,void **);
    ULONG __stdcall AddRef(void);
    ULONG __stdcall Release(void);
    // IClassFactory specific method declarations (inherited)
    HRESULT __stdcall CreateInstance(IUnknown *,REFIID,void **);
    HRESULT __stdcall LockServer(BOOL);
};
// global variable declarations
long glNumberOfActiveComponents=0;// number of active components
long glNumberOfServerLocks=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.\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);
99      wndclass.style=CS_HREDRAW|CS_VREDRAW;
100     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
                  Problem !!!"),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
141             DestroyWindow(hwnd);
142         }
143         hr=gpCSum->InitInstance(hInstance);
144         if(FAILED(hr))
145         {
146             MessageBox(hwnd,TEXT("Main Component's Type Library Can Not Be
                  Initialized."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
147             DestroyWindow(hwnd);
148         }
149         // start class factory
150         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
                  Started."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
159             DestroyWindow(hwnd);
160         }
161         // register the global object (created by InitInstance()) //
              *****************
162         hr=RegisterActiveObject(reinterpret_cast<IUnknown *>(gpCSum),
163                                 CLSID_SumAutomation,
164                                 ACTIVEOBJECT_WEAK,// ************** why ?
165                                 &dwRegisterActiveObject);
166         if(FAILED(hr))
167         {
168             if(gpIClassFactory)
169             {
170                 gpIClassFactory->Release();
171                 gpIClassFactory=NULL;
172             }
173             if(gpCSum)
174             {
175                 gpCSum->Release();
176                 gpCSum=NULL;
177             }
178             MessageBox(hwnd,TEXT("Main Component's  Active Instance Can Not Be
                  Registered."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
179             DestroyWindow(hwnd);
180         }
181     }
182     // message loop
183     while(GetMessage(&msg,NULL,0,0))
184     {
185         TranslateMessage(&msg);
186         DispatchMessage(&msg);
187     }
188     if(DontShowWindow==1)// only when COM calls this program
189     {
190         // un-register global class factory object
191         StopMyClassFactories();
192         // un-register global main object // ******************
193         if(dwRegisterActiveObject!=0)
194             RevokeActiveObject(dwRegisterActiveObject,NULL);
195     }
196     // com library un-initialization
197     CoUninitialize();
198     return((int)msg.wParam);
199
```

```
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(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
225             PostQuitMessage(0);
226         break;
227     case WM_CLOSE:
228         --glNumberOfServerLocks;
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       ➥
          QueryInterface()
242     InterlockedIncrement(&glNumberOfActiveComponents);// increment global counter
243 }
244 // Implementation Of CSum's Destructor Method
245 CSum::~CSum(void)
246 {
247     // code
248     InterlockedDecrement(&glNumberOfActiveComponents);// decrement global counter
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);
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,(WPARAM)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)
```

```
{
    // code
    m_cRef=1;// hardcoded initialization to anticipate possible failure of QueryInterface()
}
// Implementation Of CSumClassFactory's Destructor Method
CSumClassFactory::~CSumClassFactory(void)
{
    // code
}
// Implementation Of CSumClassFactory's IClassFactory's IUnknown's Methods
HRESULT CSumClassFactory::QueryInterface(REFIID riid,void **ppv)
{
    // code
    if(riid==IID_IUnknown)
        *ppv=static_cast<IClassFactory *>(this);
    else if(riid==IID_IClassFactory)
        *ppv=static_cast<IClassFactory *>(this);
    else
    {
        *ppv=NULL;
        return(E_NOINTERFACE);
    }
    reinterpret_cast<IUnknown *>(*ppv)->AddRef();
    return(S_OK);
}
ULONG CSumClassFactory::AddRef(void)
{
    // code
    InterlockedIncrement(&m_cRef);
    return(m_cRef);
}
ULONG CSumClassFactory::Release(void)
{
    // code
    InterlockedDecrement(&m_cRef);
    if(m_cRef==0)
    {
        delete(this);
        return(0);
    }
    return(m_cRef);
}
// Implementation Of CSumClassFactory's IClassFactory's Methods
HRESULT CSumClassFactory::CreateInstance(IUnknown *pUnkOuter,REFIID riid,void **ppv)
{
    // variable declarations
    HRESULT hr;
    // code
    if(pUnkOuter!=NULL)
        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_pITypeInfo==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_pITypeInfo);
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 declaraions
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
                         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
```

```
1  class ISum:public IDispatch// ************************
2  {
3  public:
4      // ISum specific method declarations
5      virtual HRESULT __stdcall SumOfTwoIntegers(int,int)=0;// pure virtual
6  };
7  // CLSID of SumAutomation Component {175E20CF-F4D3-48e2-A8E5-1AFD73797502}
8  const CLSID CLSID_SumAutomation=                                          ↵
     {0x175e20cf,0xf4d3,0x48e2,0xa8,0xe5,0x1a,0xfd,0x73,0x79,0x75,0x2};
9  // IID of ISum Interface {57140047-6957-4538-BF77-3208D32BAF63}
10 const IID IID_ISum=                                                       ↵
     {0x57140047,0x6957,0x4538,0xbf,0x77,0x32,0x8,0xd3,0x2b,0xaf,0x63};
11
```

```
1 #include<windows.h>
2 // icon
3 APPICON ICON MyIcon.ico
4 1 TYPELIB "VDGAutomationServerTypeLib.tlb"
5
```

```cpp
#define UNICODE
#include<windows.h>
#include<process.h>
#include"AutomationServerWithRegFile.h"
// global function declarations
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
// global variable declarations
ISum *pISum=NULL;
// WinMain
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,int nCmdShow)
{
    // variable declarations
    WNDCLASSEX wndclass;
    HWND hwnd;
    MSG msg;
    TCHAR AppName[]=TEXT("ComClient");
    HRESULT hr;
    // code
    // COM Initialization
    hr=CoInitialize(NULL);
    if(FAILED(hr))
    {
        MessageBox(NULL,TEXT("COM Library Can Not Be Initialized.\nProgram Will Now Exit."),TEXT("Program Error"),MB_OK);
        exit(0);
    }
    // WNDCLASSEX initialization
    wndclass.cbSize=sizeof(wndclass);
    wndclass.style=CS_HREDRAW|CS_VREDRAW;
    wndclass.cbClsExtra=0;
    wndclass.cbWndExtra=0;
    wndclass.lpfnWndProc=WndProc;
    wndclass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.hInstance=hInstance;
    wndclass.lpszClassName=AppName;
    wndclass.lpszMenuName=NULL;
    wndclass.hIconSm=LoadIcon(NULL,IDI_APPLICATION);
    // register window class
    RegisterClassEx(&wndclass);
    // create window
    hwnd=CreateWindow(AppName,
                      TEXT("Client Of COM Dll Server"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
```

```
52                          hInstance,
53                          NULL);
54      ShowWindow(hwnd,nCmdShow);
55      UpdateWindow(hwnd);
56      // message loop
57      while(GetMessage(&msg,NULL,0,0))
58      {
59          TranslateMessage(&msg);
60          DispatchMessage(&msg);
61      }
62      // COM Un-initialization
63      CoUninitialize();
64      return((int)msg.wParam);
65  }
66  // Window Procedure
67  LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,LPARAM lParam)
68  {
69      // function declarations
70      void SafeInterfaceRelease(void);
71      // variable declarations
72      HRESULT hr;
73      int iNum1,iNum2;
74      // code
75      switch(iMsg)
76      {
77      case WM_CREATE:
78          hr=CoCreateInstance(CLSID_SumAutomation,NULL,CLSCTX_LOCAL_SERVER,
79                              IID_ISum,(void **)&pISum);
80          if(FAILED(hr))
81          {
82              MessageBox(hwnd,TEXT("ISum Interface Can Not Be Obtained"),TEXT
                  ("Error"),MB_OK|MB_TOPMOST);
83              DestroyWindow(hwnd);
84          }
85          // initialize arguments hardcoded
86          iNum1=155;
87          iNum2=145;
88          // call SumOfTwoIntegers() of ISum to get the sum
89          pISum->SumOfTwoIntegers(iNum1,iNum2);
90          pISum->Release();
91          pISum=NULL;// make relesed interface NULL
92          // exit the application
93          DestroyWindow(hwnd);
94          break;
95      case WM_DESTROY:
96          SafeInterfaceRelease();
97          PostQuitMessage(0);
98          break;
99      }
100     return(DefWindowProc(hwnd,iMsg,wParam,lParam));
101 }
102 void SafeInterfaceRelease(void)
```

```
103 {
104     // code
105     if(pISum)
106     {
107         pISum->Release();
108         pISum=NULL;
109     }
110 }
111
```

```
#define UNICODE
#include<windows.h>
#include"AutomationServerWithRegFile.h"
// global function declarations
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
// WinMain
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,int nCmdShow)
{
    // variable declarations
    WNDCLASSEX wndclass;
    HWND hwnd;
    MSG msg;
    TCHAR AppName[]=TEXT("Client");
    // code
    wndclass.cbSize=sizeof(wndclass);
    wndclass.style=CS_HREDRAW|CS_VREDRAW;
    wndclass.cbClsExtra=0;
    wndclass.cbWndExtra=0;
    wndclass.lpfnWndProc=WndProc;
    wndclass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.hInstance=hInstance;
    wndclass.lpszClassName=AppName;
    wndclass.lpszMenuName=NULL;
    wndclass.hIconSm=LoadIcon(NULL,IDI_APPLICATION);
    // register window class
    RegisterClassEx(&wndclass);
    // create window
    hwnd=CreateWindow(AppName,
                      TEXT("Client Of Exe Server"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      hInstance,
                      NULL);
    ShowWindow(hwnd,nCmdShow);
    UpdateWindow(hwnd);
    // message loop
    while(GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return(msg.wParam);
}
// Window Procedure
```

```
53  LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,LPARAM lParam)
54  {
55      // variable declarations
56      IDispatch *pIDispatch=NULL;
57      HRESULT hr;
58      DISPID dispid;
59      OLECHAR *szFunctionName=L"SumOfTwoIntegers";
60      VARIANT varg[2];
61      DISPPARAMS param={varg,0,2,NULL};
62      int n1,n2;
63      // code
64      switch(iMsg)
65      {
66      case WM_CREATE:
67          // initialize COM library
68          hr=CoInitialize(NULL);
69          if(FAILED(hr))
70          {
71              MessageBox(hwnd,TEXT("COM library can not be initialized"),TEXT("COM
                  Error"),MB_OK);
72              DestroyWindow(hwnd);
73              exit(0);
74          }
75          // get ISum Interface
76          hr=CoCreateInstance(CLSID_SumAutomation,
77                              NULL,
78                              CLSCTX_LOCAL_SERVER,
79                              IID_IDispatch,
80                              (void **)&pIDispatch);
81          if(FAILED(hr))
82          {
83              MessageBox(hwnd,TEXT("Component Can Not Be Created"),TEXT("COM
                  Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
84              DestroyWindow(hwnd);
85              exit(0);
86          }
87          hr=pIDispatch->GetIDsOfNames(IID_NULL,
88                                       &szFunctionName,
89                                       1,
90                                       GetUserDefaultLCID(),
91                                       &dispid);
92          if(FAILED(hr))
93          {
94              MessageBox(NULL,TEXT("Can Not Get ID For Function"),TEXT
                  ("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
95              pIDispatch->Release();
96              DestroyWindow(hwnd);
97          }
98          n1=75;
99          n2=25;
100         // as DISPPARAMS rgvarg member receives parameters in reverse order
101         VariantInit(varg);
```

```
102         varg[0].vt=VT_INT;
103         varg[0].intVal=n2;
104         varg[1].vt=VT_INT;
105         varg[1].intVal=n1;
106         param.cArgs=2;
107         param.cNamedArgs=0;
108         param.rgdispidNamedArgs=NULL;
109         // reverse order of parameters
110         param.rgvarg=varg;
111         hr=pIDispatch->Invoke(dispid,
112                               IID_NULL,
113                               GetUserDefaultLCID(),
114                               DISPATCH_METHOD,
115                               &param,
116                               NULL,
117                               NULL,
118                               NULL);
119         if(FAILED(hr))
120         {
121             MessageBox(NULL,TEXT("Can Not Invoke Function"),TEXT("Error"),MB_OK|
                MB_ICONERROR|MB_TOPMOST);
122             pIDispatch->Release();
123             DestroyWindow(hwnd);
124         }
125         VariantClear(varg);
126         pIDispatch->Release();
127         DestroyWindow(hwnd);
128         break;
129     case WM_DESTROY:
130         CoUninitialize();
131         PostQuitMessage(0);
132         break;
133     }
134     return(DefWindowProc(hwnd,iMsg,wParam,lParam));
135 }
136
```

```
REGEDIT4
[HKEY_CLASSES_ROOT\CLSID\{175E20CF-F4D3-48e2-A8E5-1AFD73797502}]
@="MyComAutomationExe"
[HKEY_CLASSES_ROOT\CLSID\{175E20CF-F4D3-48e2-
A8E5-1AFD73797502}\LocalServer32]
@="E:\\WINNT\\system32\\AutomationServerWithRegFile.exe"
```

```
1  import "unknwn.idl" ;
2  // ISum Interface
3  [
4      object,
5      uuid(57140047-6957-4538-BF77-3208D32BAF63),// IID Of ISum
6      helpstring("ISum Interface"),
7      pointer_default(unique),
8      dual,
9      oleautomation
10 ]
11 interface ISum : IDispatch
12 {
13     import "oaidl.idl";
14     HRESULT SumOfTwoIntegers([in]int,[in]int);
15 };
16 // The Actual TypeLib Related Code
17 [
18     uuid(917898EA-9D21-4a85-81A6-DA523D483833),// LIBID Of Type Library
19     version(1.0),// major version number.minor version number
20     helpstring("SumAutomation Component's Type Library")
21 ]
22 library VDGAutomationServerTypeLib
23 {
24     importlib("stdole32.tlb");
25     // component code
26     [
27         uuid(175E20CF-F4D3-48e2-A8E5-1AFD73797502),// CLSID Of SumSubtract
            Component
28         helpstring("SumAutomation Component Class")
29     ]
30     coclass CSum
31     {
32         [default]interface ISum;
33     };
34 };
35
```