```cpp
#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
```

```
// 917898EA-9D21-4a85-81A6-DA523D483833
const GUID LIBID_AutomationServer=
  {0x917898ea,0x9d21,0x4a85,0x81,0xa6,0xda,0x52,0x3d,0x48,0x38,0x33};
CSum *gpCSum=NULL;// **********************
IClassFactory *gpIClassFactory=NULL;
HWND ghwnd=NULL;
DWORD dwRegisterClassFactory;// *************************** just renamed
DWORD dwRegisterActiveObject;// ***************************
// WinMain
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,int nCmdShow)
{
    // function declarations
    HRESULT InitInstance(HINSTANCE);
    HRESULT StartMyClassFactories(void);
    void StopMyClassFactories(void);
    // variable declarations
    WNDCLASSEX wndclass;
    MSG msg;
    HWND hwnd;
    HRESULT hr;
    int DontShowWindow=0;// 0 means show the window
    TCHAR AppName[]=TEXT("ExeAutomationServer");// ******************
    TCHAR szTokens[]=TEXT("-/");
    TCHAR *pszTokens;
    TCHAR lpszCmdLine[255];
    // com library initialization
    hr=CoInitialize(NULL);
    if(FAILED(hr))
        return(0);
    MultiByteToWideChar(CP_ACP,0,lpCmdLine,255,lpszCmdLine,255);
    pszTokens=wcstok(lpszCmdLine,szTokens);
    while(pszTokens!=NULL)
    {
        // COM is calling me with Automation
        if(wcsicmp(pszTokens,TEXT("Embedding"))==0)
        {
            DontShowWindow=1;// dont show window but message loop must
            break;
        }
        else
        {
            MessageBox(NULL,TEXT("Bad Command Line Arguments.\nExitting The
              Application."),TEXT("Error"),MB_OK);
            exit(0);
        }
        pszTokens=wcstok(NULL,szTokens);
    }
    // window code
    wndclass.cbSize=sizeof(wndclass);
    wndclass.style=CS_HREDRAW|CS_VREDRAW;
    wndclass.cbClsExtra=0;
    wndclass.cbWndExtra=0;
    wndclass.lpfnWndProc=WndProc;
    wndclass.hIcon=LoadIcon(hInstance,TEXT("APPICON"));// ******************
    wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground=(HBRUSH)GetStockObject(BLACK_BRUSH);
    wndclass.hInstance=hInstance;
    wndclass.lpszClassName=AppName;
    wndclass.lpszMenuName=NULL;
    wndclass.hIconSm=LoadIcon(hInstance,TEXT("APPICON"));// *********************
    // register window class
    RegisterClassEx(&wndclass);
    // create window
    hwnd=CreateWindow(AppName,
                      TEXT("Exe Server With Reg File"),// *****************
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      hInstance,
                      NULL);
    // initialize global window handle
    ghwnd=hwnd;
    if(DontShowWindow!=1)
    {
        // usual functions
        ShowWindow(hwnd,SW_MAXIMIZE);// *************
        UpdateWindow(hwnd);
        // increament server lock
        ++glNumberOfServerLocks;
    }
    if(DontShowWindow==1)// only when COM calls this program *****************
    {
        // initialize the global instance of main object
        gpCSum=new CSum;
        if(gpCSum==NULL)
        {
            MessageBox(hwnd,TEXT("Main Component Can Not Be Created.\nMemory
              Problem !!!"),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
            DestroyWindow(hwnd);
        }
        hr=gpCSum->InitInstance(hInstance);
        if(FAILED(hr))
        {
            MessageBox(hwnd,TEXT("Main Component's Type Library Can Not Be
              Initialized."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
            DestroyWindow(hwnd);
        }
        // start class factory
        hr=StartMyClassFactories();
        if(FAILED(hr))
        {
            if(gpCSum)// *******************
            {
                gpCSum->Release();
                gpCSum=NULL;
            }
            MessageBox(hwnd,TEXT("Main Component's Class Factory Can Not Be
              Started."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
            DestroyWindow(hwnd);
        }
        // register the global object (created by InitInstance()) //
          *****************
        hr=RegisterActiveObject(reinterpret_cast<IUnknown *>(gpCSum),
                                CLSID_SumAutomation,
                                ACTIVEOBJECT_WEAK,// ************** why ?
                                &dwRegisterActiveObject);
        if(FAILED(hr))
        {
            if(gpIClassFactory)
            {
                gpIClassFactory->Release();
                gpIClassFactory=NULL;
            }
            if(gpCSum)
            {
                gpCSum->Release();
                gpCSum=NULL;
            }
            MessageBox(hwnd,TEXT("Main Component's  Active Instance Can Not Be
              Registered."),TEXT("Error"),MB_OK|MB_ICONERROR|MB_TOPMOST);
            DestroyWindow(hwnd);
        }
    }
    // message loop
    while(GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    if(DontShowWindow==1)// only when COM calls this program
    {
        // un-register global class factory object
        StopMyClassFactories();
        // un-register global main object // *******************
        if(dwRegisterActiveObject!=0)
            RevokeActiveObject(dwRegisterActiveObject,NULL);
    }
    // com library un-initialization
    CoUninitialize();
    return((int)msg.wParam);

```

```
}
// Window Procedure
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,LPARAM lParam)
{
    // variable declarations
    HDC hdc;
    RECT rc;
    PAINTSTRUCT ps;
    // code
    switch(iMsg)
    {
    case WM_PAINT:
        GetClientRect(hwnd,&rc);
        hdc=BeginPaint(hwnd,&ps);
        SetBkColor(hdc,RGB(0,0,0));
        SetTextColor(hdc,RGB(0,255,0));
        DrawText(hdc,
                 TEXT("This   Is   A   COM   Exe   Automation   Server   Program.
                      Not   For   You   !!!"),
                 -1,
                 &rc,
                 DT_SINGLELINE|DT_CENTER|DT_VCENTER);
        EndPaint(hwnd,&ps);
        break;
    case WM_DESTROY:
        if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
            PostQuitMessage(0);
        break;
    case WM_CLOSE:
        --glNumberOfServerLocks;
        ShowWindow(hwnd,SW_HIDE);
        // fall through,hence no break
    default:
        return(DefWindowProc(hwnd,iMsg,wParam,lParam));
    }
    return(0L);
}
// Implementation Of CSum's Constructor Method
CSum::CSum(void)
{
    // code
    m_pITypeInfo=NULL;// *****************
    m_cRef=1;// hardcoded initialization to anticipate possible failure of
      QueryInterface()
    InterlockedIncrement(&glNumberOfActiveComponents);// increment global counter
}
// Implementation Of CSum's Destructor Method
CSum::~CSum(void)
{
    // code
    InterlockedDecrement(&glNumberOfActiveComponents);// decrement global counter
}
```

```
// Implementation Of CSum's IUnknown's Methods
HRESULT CSum::QueryInterface(REFIID riid,void **ppv)
{
    // code
    if(riid==IID_IUnknown)
        *ppv=static_cast<ISum *>(this);
    else if(riid==IID_IDispatch)// *************
        *ppv=static_cast<ISum *>(this);
    else if(riid==IID_ISum)
        *ppv=static_cast<ISum *>(this);
    else
    {
        *ppv=NULL;
        return(E_NOINTERFACE);
    }
    reinterpret_cast<IUnknown *>(*ppv)->AddRef();
    return(S_OK);
}
ULONG CSum::AddRef(void)
{
    // code
    InterlockedIncrement(&m_cRef);
    return(m_cRef);
}
ULONG CSum::Release(void)
{
    // code
    InterlockedDecrement(&m_cRef);
    if(m_cRef==0)
    {
        delete(this);
        if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
            PostMessage(ghwnd,WM_QUIT,(WPARAM)0,(LPARAM)0L);
        return(0);
    }
    return(m_cRef);
}
// Implementation Of ISum's Methods
HRESULT CSum::SumOfTwoIntegers(int num1,int num2)// ********************
{
    // variable declarations
    int num3;
    TCHAR szSum[255];
    // code
    num3=num1+num2;
    wsprintf(szSum,TEXT("Automation Server Gives You Sum Of %d And %d As %
      d"),num1,num2,num3);
    MessageBox(NULL,szSum,TEXT("Automation Server"),MB_OK);
    return(S_OK);
}
// Implementation Of CSumClassFactory's Constructor Method
CSumClassFactory::CSumClassFactory(void)
```

```cpp
{
    // 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);
    // ********************************************** new
    // object is already created in WinMain(), just call QI() to get requested interface
    hr=gpCSum->QueryInterface(riid,ppv);
    gpCSum->Release();// anticipate possible failure of QueryInterface()
    return(hr);
}
HRESULT CSumClassFactory::LockServer(BOOL fLock)
{
    // code
    if(fLock)
        InterlockedIncrement(&glNumberOfServerLocks);
    else
        InterlockedDecrement(&glNumberOfServerLocks);
    if(glNumberOfActiveComponents==0 && glNumberOfServerLocks==0)
        PostMessage(ghwnd,WM_QUIT,(WPARAM)0,(LPARAM)0L);
    return(S_OK);
}
// ******************************************** new starts **********************
// Implementation Of CSum's IDispatch's Methods
HRESULT CSum::GetTypeInfoCount(UINT *pCountTypeInfo)
{
    // code
    *pCountTypeInfo=1;// as we have only one method SumOfTwoIntegers()
    return(S_OK);
}
HRESULT CSum::GetTypeInfo(UINT iTypeInfo,LCID lcid,ITypeInfo **ppITypeInfo)
{
    // code
    *ppITypeInfo=NULL;
    if(iTypeInfo!=0)
        return(DISP_E_BADINDEX);
    m_pITypeInfo->AddRef();
    *ppITypeInfo=m_pITypeInfo;
    return(S_OK);
}
HRESULT CSum::GetIDsOfNames(REFIID riid,LPOLESTR *rgszNames,UINT cNames,LCID lcid,DISPID *rgDispId)
{
    // code
    return(DispGetIDsOfNames(m_pITypeInfo,rgszNames,cNames,rgDispId));
}
HRESULT CSum::Invoke(DISPID dispIdMember,REFIID riid,LCID lcid,WORD wFlags,DISPPARAMS *pDispParams,VARIANT *pVarResult,EXCEPINFO *pExcepInfo,UINT *puArgErr)
{
    // variable declarations
    HRESULT hr;
    // code
    hr=DispInvoke(this,
                  m_pITypeInfo,
                  dispIdMember,
                    wFlags,
                    pDispParams,
                    pVarResult,
                    pExcepInfo,
                    puArgErr);
    return(hr);
}
// custom methods
HRESULT CSum::InitInstance(HINSTANCE hInst)
{
    // variable declarations
    HRESULT hr;
    ITypeLib *pITypeLib=NULL;
    TCHAR szExeFileName[_MAX_PATH],szTypeLibPath[_MAX_PATH];
    // code
    if(m_pITypeInfo==NULL)
    {
        hr=LoadRegTypeLib(LIBID_AutomationServer,
                          1,0,// major/minor version numbers
                          0x00,
                          &pITypeLib);
        if(FAILED(hr))
        {
            GetModuleFileName(hInst,szExeFileName,_MAX_PATH);
            wsprintf(szTypeLibPath,TEXT("%s\\1"),szExeFileName);
            hr=LoadTypeLib(szTypeLibPath,&pITypeLib);
            if(FAILED(hr))
                return(hr);
            hr=RegisterTypeLib(pITypeLib,szTypeLibPath,NULL);
            if(FAILED(hr))
                return(hr);
        }
        hr=pITypeLib->GetTypeInfoOfGuid(IID_ISum,&m_pITypeInfo);
        if(FAILED(hr))
        {
            pITypeLib->Release();
            return(hr);
        }
        pITypeLib->Release();
    }
    return(S_OK);
}
// ************************************* new ends
  ***********************************
// other methods
HRESULT StartMyClassFactories(void)
{
    // variable declaraions
    HRESULT hr;
    // code
    gpIClassFactory=new CSumClassFactory;
    if(gpIClassFactory==NULL)
```

```cpp
        return(E_OUTOFMEMORY);
    gpIClassFactory->AddRef();
    // register the class factory
    hr=CoRegisterClassObject(CLSID_SumAutomation,
                             static_cast<IUnknown *>(gpIClassFactory),
                             CLSCTX_LOCAL_SERVER,
                             REGCLS_SINGLEUSE,// ******************** why ?
                             &dwRegisterClassFactory);// ************* just renamed
    if(FAILED(hr))
    {
        gpIClassFactory->Release();
        return(E_FAIL);
    }
    return(S_OK);
}
void StopMyClassFactories(void)
{
    // code
    // un-register the class factory
    if(dwRegisterClassFactory!=0)
        CoRevokeClassObject(dwRegisterClassFactory);
    if(gpIClassFactory!=NULL)
        gpIClassFactory->Release();
}
```