Appendix E. emitcom: An Emitter of COM Interfaces

The emitcom emitter is a program that creates a binding for a SOM class so the class can be used in the context of COM, Microsoft's component interface model. The binding exports COM-style interfaces so that a SOM class can be used from OLE 2.0 programs. The generated COM interface is aggregatable. The **emitcom** emitter generates all the files necessary to build a DLL for the binding, and **emitcom** can generate COM bindings for ancestor classes.

Contents

emitcom Syntax

The **emitcom** command is issued as follows:

emitcom ***filestem comstem***

filestem
the prefix name of a SOM IDL file (*filestem*.**idl**).

comstem
the prefix for the corresponding COM binding files.

Execution of emitcom

For the IDL file *filestem*.**idl**, **emitcom** creates a set of files that compose an interface (or usage binding) that gives an OLE 2.0 program access to the SOM class described in *filestem*.**idl**. The following files are created: *comstem*.**mak**, *comstem*.**xh**, *comstem*.**cpp**, *comstem*.**def** and *comstem*.**reg**. Once **emitcom** has run, issue the commands:

nmake -f *comstem*.**mak** to create a DLL and LIB;

regedit /s *comstem*.**reg** to register the DLL with the REG.DAT database.

The COM interface generated by **emitcom** is the SOM class's interface. That is, the interface contains the union of the methods of the SOM class and all of its ancestors.

The COM interface is generated in C++; *comstem*.**cpp** is the implementation file and *comstem*.**xh** is a header file for users of the interface. Because SOM is language neutral, it does not matter what language is used to implement the SOM class.

The *comstem*.**def** file is used by the linker to make the DLL. In generating a makefile (*comstem*.**mak**), **emitcom** makes the following decision:

- If the *filestem*.**idl** file contains a **dllname** modifier, the associated *dllname*.**lib** file is used in the link statement. If there is no **dllname** modifier in the IDL file, then the link statement is generated with *filestem***.obj**. See **Modifier Statements** on page 133 of *SOMobjects Developer Toolkit Programmer's Guide*.

- If *filestem***.obj** is not desired, one can always edit the *comstem***.mak**.

The *comstem***.reg** file contains the information for registering the COM interface to the SOM class in the registration database. The DLL file name that is used in *comstem***.reg** is *comstem***.dll**; if this DLL is to be named otherwise, you must edit the *comstem***.reg** file.

The COM interface for the SOM class named *className* is defined in *comstem***.xh**. The interface is implemented as a C++ class named *className***COMIntf**. To use the SOM class in a program, one must include the header *comstem***.xh** and create instances of the C++ class *className***COMIntf**, which creates instances of the SOM class.

Interface Identifiers

The *filestem***.idl** file must give the class identifier and the interface identifier needed for registration. This is done with two new modifiers:

CLSID *className*
IID_*className*

where *className* is the name of the SOM class for which a COM interface is being generated. The various forms of the modifiers are as follows:

CLSID_*className = guid1*;
IID_*className = guid2;*
IID_*parentClassName1 = guid3;*
IID_*parentClassName2;*
IID_*SOMObjectsToolkitClassName;*

The first two forms are mandatory, because the class and interface identifiers for the SOM class must be specified. The third form is used to specify an interface identifier for a parent class. However, if an interface identifier is specified in the IDL of the parent, the fourth form should be used. The fifth form is used for parent classes that are part of the SOMObjects Toolkit. Note that the third, fourth, and fifth forms are used only when an interface to the parent is to be aggregated into the COM binding.

Each ancestor of a SOM class provides an interface to instances of that class. Therefore, each of these may be aggregated into the COM binding. This is indispensable in the case where the instance is to be passed to code that was created for the ancestor interface (that is, code that uses the COM binding generated from the ancestor's IDL). In such cases, the caller must coerce the instance interface by calling **QueryInterface** before passing the instance into the code created for the ancestor.

User Procedure

The following diagram depicts the total process, where *filestem* is **S** and *comstem* is **C**.

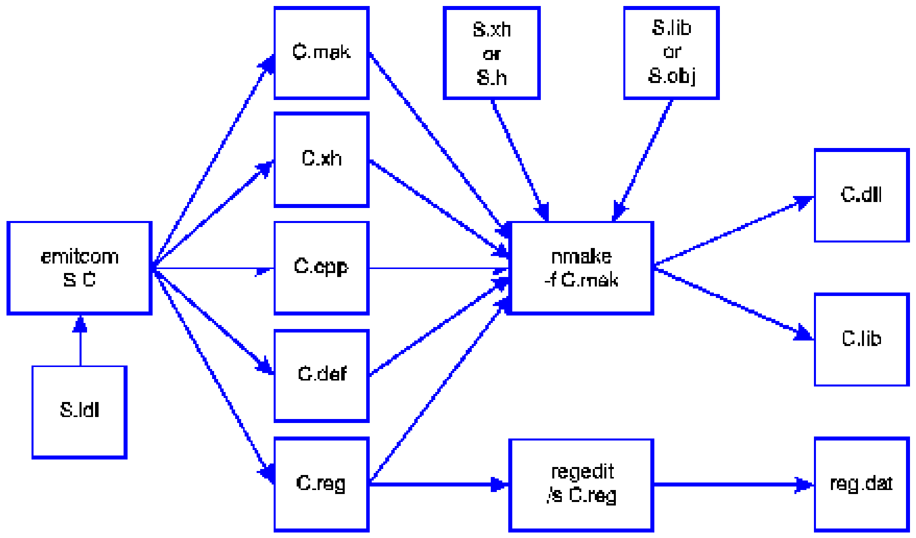

Figure 37. emitcom Total Process

In summary, you will perform the following steps:

1. Add CLSID and IID to *filestem*.**idl** with the modifiers:

   CLSID_*className*
   IID_*className*

   and to produce the files:

   comstem.**mak**, *comstem*.**xh**, *comstem*.**cpp**, *comstem*.**def**, *comstem*.**reg**

   comstem.**mak** is generated with the value of the **dllname** SOM IDL modifier or *filestem*.**obj** in the **LINK** command.

2. Run **“nmake -f** *comstem*.**mak”** to produce the files:

   comstem.**lib**, *comstem*.**dll**

3. Run **“regedit /s** *comstem*.**reg”** to update **reg.dat** (`\windows\reg.dat`). Remember to update the DLL location in *comstem*.**reg** if necessary.

4. Install the header (*comstem*.**xh**) and library (*comstem*.**dll** and *comstem*.**lib**) in the required directory.

The Generated Interface

Suppose the SOM class in *filestem*.**idl** is named *X*. The COM interface generated by **emitcom** in the *comstem*.**xh** file then appears as follows.

```
#include “<filestem>.xh”

DEFINE_GUID (CLSID_X,  <class identifier>);
DEFINE_GUID (IID_X, <interface identifier>);
class XCOMIntf : IUnknown
{
 public:
    XCOMIntf( LPUNKNOWN );          // constructor
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);
    // SOM methods
    < all methods supported by X >

};
class XCOMFactory : public IClassFactory
{
  public:
    XCOMFactory();
    STDMETHOD(QueryInterface)(REFIID riid,void FAR* FAR* ppv);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);
    STDMETHOD(CreateInstance)(IUnknown FAR* punkOuter,
                              REFIID riid,
                              void FAR* FAR* ppv);
    STDMETHOD(LockServer)(BOOL fLock);
};
```

There is a C++ class named **XCOMIntf** that contains the three **IUnknown** methods and all of the methods that the SOM class *X* supports. Any method defined in the *X* SOM class or any of its ancestor classes).

There is one constructor for **XCOMIntf** which takes an LPUNKNOWN parameter that is the **pUnkOuter** of the controlling interface in the case that **XCOMIntf** is part of an aggregate. If the interface is not part of an aggregate, the constructor should be called with a NULL value.

Customizing the *comstem*.makCOM

The *comstem***.mak** file is used to create a DLL that implements COM interface. The file is designed to be invoked from a makefile. There are two macro parameters in *comstem***.mak** that can be set: OBJS and LIBS. The first is used to indicate any other object files that are to be linked into the DLL. The second is used to specify any other libraries on which the DLL depends.

In addition, when the environment variable COMDEBUG is set to 1, the *comstem***.dll** is compiled with the debugger options.

emitcom Example

As an example, the standard SOM `Hello` sample has been modified to generate a COM binding for the `Hello` class. The full text of this modified example is among the SOM samples. The following is a modified IDL file for the `Hello` SOM sample program that can be used to generate a COM interface. The `Hello` sample SOM class is implemented in C (not C++), yet the COM binding is implemented in C++.

```
#include <somobj.idl>
interface Hello : SOMObject
/* this is a simple class that demonstrates how to define
 * the interface to a new class of objects in SOM IDL.
 */
{
        string sayHello();
        // This method returns the string “Hello, World!”.
#ifdef __SOMIDL__
implementation
 {
 releaseorder: sayHello;
 CLSID_Hello = “12345678-abcd-1234-1234-123456789012";
 IID_Hello =   “01234567-0123-cdef-0123-012345678901";
 };
#endif
};
```

Next is a fragment of a main program that uses the COM interface generated by **emitcom**. Although this looks like using a SOM class with the C++ bindings, it actually is an example of using a COM interface. That is, `HelloCOMClass` is an implementation of a COM interface that supports both the **IUnknown** methods and all the methods of the `Hello` SOM class.

```
HelloCOMIntf *pintf;
HRESULT hr;
LPCLASSFACTORY  pHelloFactory;
switch (message){
case WM_CREATE:
             hr = CoGetClassObject(CLSID_Hello,
                              CLSCTX_INPROC_SERVER,
                              NULL,
                              IID_IClassFactory,
                            (void FAR* FAR*)& pHelloFactory);
if ( SUCCEEDED(hr) ) {
pHelloFactory->CreateInstance(NULL,
                                      IID_Hello,
                                 (void FAR* FAR*)&pintf );
pHelloFactory->Release();
}
else {
PostQuitMessage(2);
}
return 0;
case WM_PAINT:
hdc = BeginPaint (hwnd, &ps) ;
GetClientRect (hwnd, &rect) ;
strcpy(sBuf,
                pintf->sayHello(somGetGlobalEnvironment()));
DrawText (hdc, sBuf, -1, &rect,
   DT_SINGLELINE |
   DT_CENTER | DT_VCENTER);
EndPaint (hwnd, &ps) ;
return 0 ;
case WM_DESTROY:
PostQuitMessage (0) ;
return 0 ;
}
```

Following is an example of the main procedure for the preceding message loop.

```
#include <comhello.xh>
long FAR PASCAL _export WndProc (HWND, UINT, UINT, LONG) ;
int PASCAL WinMain (HANDLE hInstance,
                    HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
static char szAppName[] = “Hello” ;
HWND        hwnd ;
MSG         msg ;
WNDCLASS    wndclass ;
HRESULT hr;
hr = CoInitialize( NULL );
if ( !SUCCEEDED(hr) ) {
exit(1) }
if (!hPrevInstance){
  wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc   = WndProc ;
  wndclass.cbClsExtra    = 0 ;
  wndclass.cbWndExtra    = 0 ;
  wndclass.hInstance     = hInstance ;
wndclass.hIcon    = LoadIcon (NULL, IDI_APPLICATION) ;
  wndclass.hCursor  = LoadCursor (NULL, IDC_ARROW);
  wndclass.hbrBackground = GetStockObject (LTGRAY_BRUSH);
  wndclass.lpszMenuName  = NULL ;
  wndclass.lpszClassName = szAppName ;
  RegisterClass (&wndclass) ; }
hwnd = CreateWindow (
   szAppName,               // window class name
     “Hello Program”,         // window caption
     WS_OVERLAPPEDWINDOW,     // window style
   CW_USEDEFAULT,     // initial x position
   CW_USEDEFAULT,     // initial y position
   CW_USEDEFAULT,     // initial x size
   CW_USEDEFAULT,     // initial y size
   NULL,              // parent window handle
   NULL,              // window menu handle
   hInstance,         // program instance handle
   NULL) ;            // creation parameters
ShowWindow (hwnd, nCmdShow) ;
UpdateWindow (hwnd) ;
while (GetMessage (&msg, NULL, 0, 0)) {
TranslateMessage (&msg) ;
DispatchMessage (&msg) ;}
CoUninitialize( );
return msg.wParam ;
}
```

emitcom Limitations

The following are known limitations at the current time:

- *filestem*.**idl** cannot contain more than one interface nor can it contain IDL modules.
- **emitcom** creates the following temporary files: *filestem*.**cmm**, *filestem*.**cmh**, *filestem*.**cmc**, *filestem*.**cmd**, and *filestem*.**reg**. The **emitcom** emitter should not be run in a directory where you have files with these names (when **emitcom** runs, it overwrites these files). *filestem* is the first parameter to **emitcom**.
- *comstem*.**mak** is for Microsoft’s **nmake**; *comstem*.**mak** expects the C++ compiler to be named **cl**. This makefile uses the temporary file *comstem*.**lrf**.