Bauhaus-Universität Weimar
Lehrstuhl für Informatik im Bauwesen
Prof. Dr. K. Beucke

COMET+

Eine C++ Bibliothek zur Server-Entwicklung unter COM

Lutz Röder, Karsten Koch Weimar, Oktober 1997

1. WAS IST COMET+?

Der Name COMET+ steht für eine Reihe von in C++ implementierten Macros und Klassen, welche die Entwicklung von Servern entscheidend vereinfacht. Durch deren Benutzung kann auch für kleinere Applikationen unter C++ die Bereitstellung von Funktionalität als Server mit vertretbarem Aufwand realisiert werden. Zur genauen Beschreibung siehe auch 6.

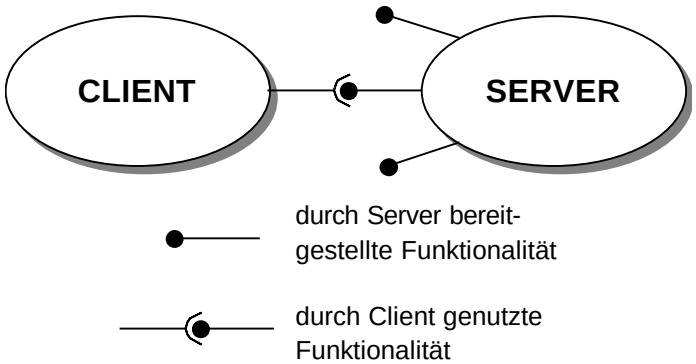
2. EINLEITUNG

Durch objektorientierte Programmierung ist es seit längerem möglich, Software in Komponenten zu unterteilen. Ständig steigende Anforderungen an Software und der damit verbundene wachsende Entwicklungsaufwand hatten die Einführung von Komponentensoftware zur Folge.

Der Begriff **Komponentensoftware** steht für Software mit einem hohen Grad an Modularität und Erweiterbarkeit. Dadurch ist eine größere Flexibilität gegeben, die individuelle Lösungen auch mit großen Systemen zuläßt. Eng mit dem Begriff der Komponentensoftware ist das **Client-Server-Prinzip** verbunden. Dieses kategorisiert jeweils zwei zusammenarbeitende Programme oder Programmteile.

Ein **Server** stellt Funktionalität bereit. Diese ist allen Programmen, die mit dem Server über eine Schnittstelle kommunizieren können, zugänglich. Programme, die Funktionalität eines Servers nutzen, heißen **Client**. Ein Programm kann zur Laufzeit auch Server und Client zugleich sein. Das hängt von seinen jeweiligen Beziehungen zu anderen Programmteilen ab.

Abbildung 1



Um Komponenten für eine Software zu entwickeln, sind also standardisierte Schnittstellen notwendig. Außerdem muß ein Mechanismus vorhanden sein, der die Zusammenarbeit zwischen Programmteilen regelt, d.h. die Schnittstellen verwaltet. Solch einen Mechanismus stellt das **COM** ("Component Object Model") der Open Systems Foundation dar.

DCOM ("Distributed COM") ist ein Standard für das Zusammenarbeiten von Programmen bzw. Programmteilen unter Windows 95/NT zur Laufzeit (auch Netzwerkfähigkeit). Funktionalität des einen Programmes kann von einem anderen Programm benutzt werden, wenn dieses von dieser Funktionalität Kenntnis hat. Damit wird redundante Implementierung von Programmteilen vermieden und gleichzeitig deren unabhängige Austauschbarkeit erreicht. COM ist also eine Technologie, die weit mehr bietet als nur das Prinzip der **Dynamic Link Libraries** (DLL's). Bereits weit verbreitete Standards wie **OLE** ("Object Linking and Embedding") und **ActiveX** machen sich diese Technologie zunutze.

Natürlich existieren zu COM Alternativen. Das Objektmodell der OMG namens **CORBA** ("Component Object Request Broker Architecture") ist ein Beispiel dafür. Für die Entwicklung unter Windows NT ist die Verwendung von COM jedoch von großem Vorteil, da die benötigten Mechanismen integrierter Bestandteil des Betriebssystems sind.

3. SERVER-ENTWICKLUNG UNTER COM

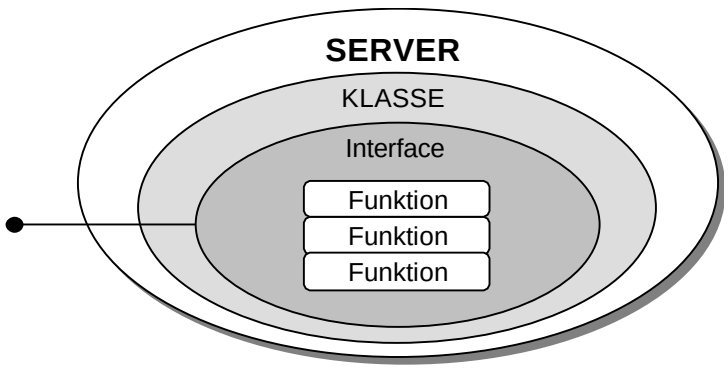
Die Implementierung und die genutzten Mechanismen hängen stark von der Art des Servers ab. Es wird zwischen zwei grundlegenden Arten unterschieden:

- Local Server
- In Process Server

Local Server liegen als Datei im EXE-Format (Execute) vor. Sie können also explizit ausgeführt werden und verfügen über genau eine Einsprungadresse. **In Process Server** hingegen liegen im DLL-Format (Dynamic Link Library) vor und verfügen über mehrere Einsprungadressen, wobei sich eine von allen anderen als Haupteinsprung unterscheidet. Diese Funktion wird auch "DllMain" genannt.

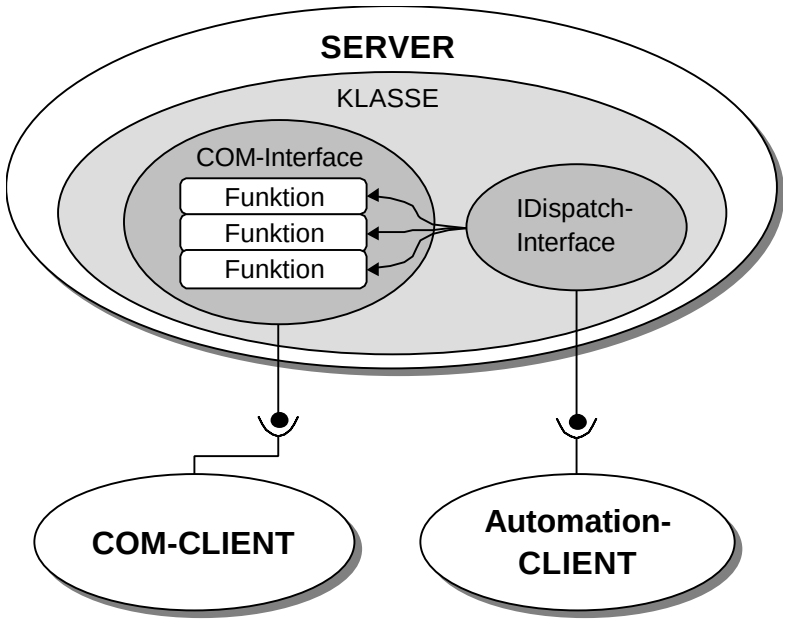
Die Implementierung eines COM-Servers wird mit Klassen realisiert, die Funktionen über sogenannte Interfaces bereitstellen. Jedes Interface ist eine Zusammenfassung von unterschiedlichen Funktionen. Es gibt mehrere Varianten der Schachtelung; Abbildung 2 zeigt eine übliche Variante.

Abbildung 2



Der OLE-Standard definiert eine Reihe von Interfaces zum Austausch und zur Einbettung von Objekten. Es gibt beispielsweise Interfaces zur Serialisierung, Visualisierung und zur Druckerausgabe von dokumentenorientierten Objekten. Außerdem gibt es unter OLE die Möglichkeit, nicht standardisierte Funktionen über ein sogenanntes **Automation-Interface** bereitzustellen. Automation-Interfaces haben den Vorteil, daß sie auch ohne großen Implementierungsaufwand genutzt werden können. Der Aufruf kann auch über Interpretersprachen wie Visual Basic erfolgen. Diese Möglichkeit im Zusammenhang mit den rein über COM bereitgestellten Funktionen wird "**Dual Interface Technology**" genannt. Die folgende Abbildung zeigt einen schematisierten Server mit Dual Interfaces.

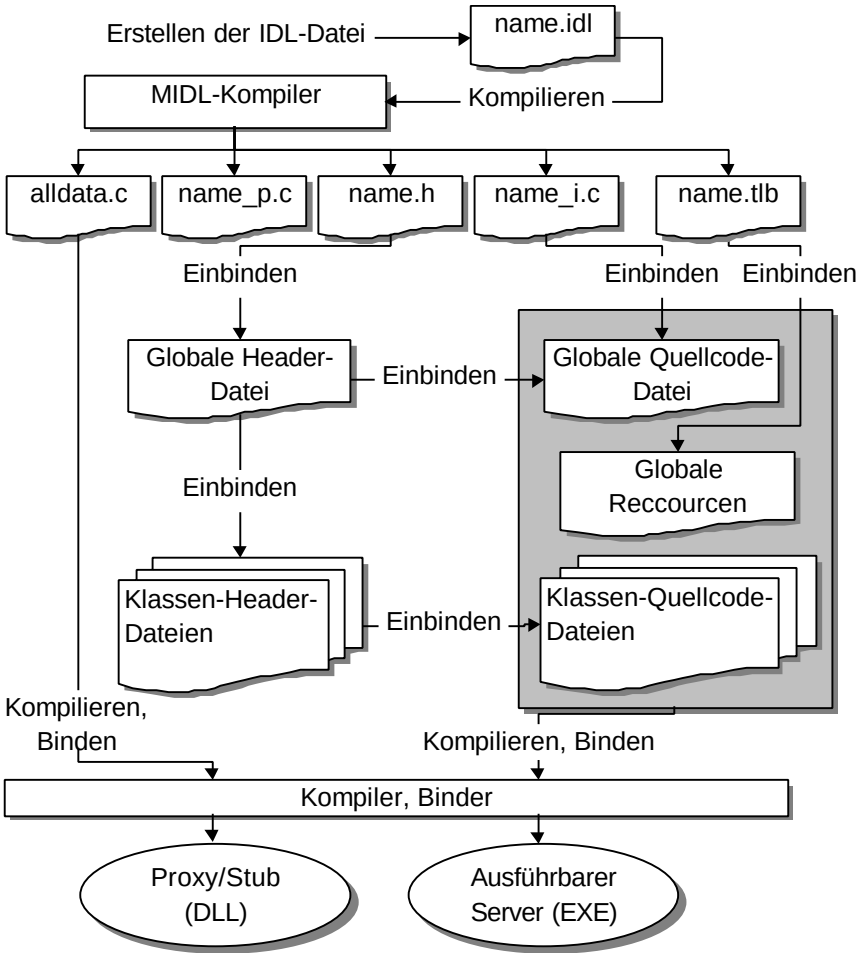
Abbildung 3



Das Interface mit dem Namen "**IDispatch**" ("I" steht für Interface) übernimmt hierbei die Funktion eines einfachen Interpreters, über den auf alle "automatisierten" Funktionen zugegriffen werden kann.

Ein Dispatch Interface kann aus einer sogenannten "TypeLibrary" (zu deutsch: Typenbibliothek) automatisch erzeugt werden. TypeLibraries wiederum werden aus Interfacequelldateien kompiliert, die im Syntax der "**Interface Definition Language**" vorliegen müssen. In die Entwicklungsumgebung von Visual C++ ist eigens dafür ein Kompiler mit dem Namen **MIDL** integriert. Dieser Kompiler erzeugt aus besagter Interfacequelldatei (Dateiendung .IDL) die TypeLibary (Dateiendung .TLB) und C++-Header- bzw. C++-Quellcodedateien, die in das bestehende Programm eingebunden und kompiliert werden. Die TypeLibrary wird als Reccource eingebunden.

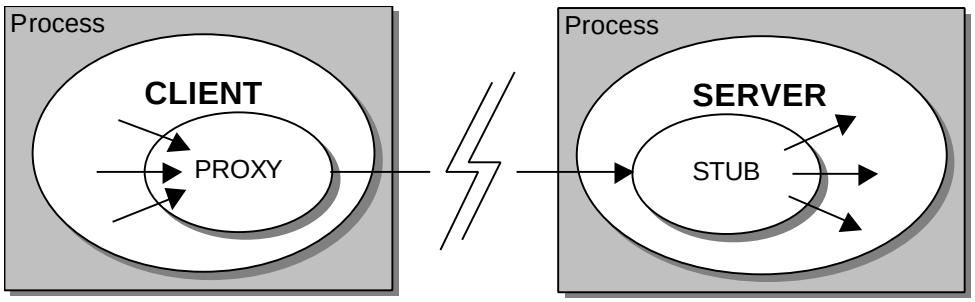
Abbildung 4



Das Kompilieren der Datei "alldata.c" ist nur dann notwendig, wenn ein Local Server entwickelt werden soll. Diese Server benötigen für die Kommunikation unter COM zusätzliche Programmteile, die **Proxy** und **Stub** genannt werden.

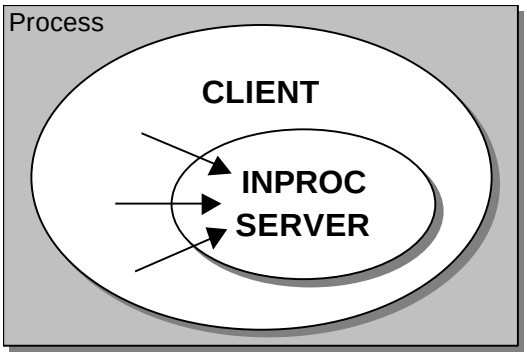
Die Aufgabe dieser Programmteile ist es, die Kommunikation zwischen Server und Client zu gewährleisten, wenn diese als unterschiedliche Prozesse ausgeführt werden. Wird beispielsweise ein Local Server von einem Clienten benötigt, so wird der Server in einem eigenen Adressraum ausgeführt und zusätzlich in beiden Programmen die Proxy/Stub-DLL geladen. Die Funktionsaufrufe werden im Client durch den Proxy verarbeitet und an den Stub übergeben, der sie wiederum verarbeitet und an den Server weiterleitet. Dies ist Notwendig, um die Funktion des COM-Mechanismus auch im Netzwerk zu garantieren.

Abbildung 5



Wird der Server jedoch als In Process Server im Adressraum des Clienten ausgeführt, so wird die Proxy/Stub-DLL nicht benötigt, da direkte Funktionsaufrufe möglich sind.

Abbildung 6



Um nun Local oder In Process Server zu entwickeln gibt es zur Zeit mehrere Wege.

Der wohl aufwändigste ist die eigene Implementierung des COM-Standards (IUnknown, IClassfactory etc.). Vorteile sind natürlich eine maximale Ausnutzung der durch die COM-Technologie gegebenen Möglichkeiten und eine flexible Implementierung. Da diese aber nur durch unverhältnismäßig hohe Einarbeitungs- und Implementierungdauern zu erreichen sind, ist diese Variante für kleinere Projekte inakzeptabel.

Eine weitere Möglichkeit ist die Implementierung mithilfe der Macros der **MFC** ("Microsoft Foundation Classes"). Diese bieten jedoch keine Unterstützung von TypeLibraries. Dadurch wird die Implementierung des Automation-Interfaces unnötig erschwert. Ein weiteres Problem ist die Erstellung von Dual Interfaces.

Die **ATL** (Active Template Library) verspricht eine einfache Implementierung von Interfaces und der dazugehörigen Programmteile. Als eine dritte Alternative stellt dieser Weg eine einfache Möglichkeit dar, umfassende Teile der Implementierung zu Automatisieren. Jedoch ist auch hier eine Einarbeitungszeit notwendig, die für kleinere Projekte nicht vertretbar ist.

4. DIE COMET+ BIBLIOTHEK

Ziel der COMET+-Bibliothek ist die Vereinfachung der Entwicklung von COM-Servern. Einige Teile der COM-Implementierung werden sogar vollständig von Macros übernommen. Das sind zum Beispiel die Implementierung des IUnknown-Interfaces, der ClassFactory und die Registrierung. Auch Dual Interfaces können automatisch generiert werden. Zu detaillierte Informationen siehe Anlage A: COMET+ Sourcecode.

Im wesentlichen besteht die Bibliothek aus den 3 Klassen

- ComModule
- ComClassFactory
- ComClass

und 6 Macros, die auf zwei Ebenen zum Einsatz kommen.

Die Server Ebene

Im Server wird eine Instanz der Klasse ComModule angelegt, die folgende Funktionen bereitstellt:

- `Initialize`
  Diese Funktion muß beim Start des Servers aufgerufen werden.
- `Uninitialize`
  Diese Funktion muß beim Beenden des Servers aufgerufen werden.
- `UpdateRegistry`
  Diese Funktion nimmt die Registrierung vor.
- `GetClassObject`
  Diese Funktion übernimmt die Aufgaben von `DllGetClassObject`
- `CanUnloadNow`
  Diese Funktion übernimmt die Aufgaben von `DllGetUnloadNow`

Des weiteren muß eine `COM_CLASS_TABLE` deklariert werden, in der festgelegt wird, welche Klassen unter welchem Namen registriert werden. Die folgende Abbildung zeigt eine solche Table. "ClassName" steht dabei für den Klassennamen und "Application.ClassName" für den Registrierungseintrag.

Abbildung 7

server.cpp

```
COM_CLASS_TABLE_BEGIN()
  COM_CLASS_TABLE_MEMBER(ClassName,CLSID_ClassName,"Application.ClassName")
  ...
COM_CLASS_TABLE_END()
```

Die Klassenebene

Klassen, die Funktionen über Interfaces bereitstellen sollen, müssen von ComClass und von der Interfaceklasse abgeleitet sein. Sie enthalten eine COM_CLASS_INTERFACE_TABLE mit dem Namen der Interfaces und der Interface ID (IID).

Abbildung 8

class.h

```
class  ClassName: public ComClass , public IClassName , ...
{
  COM_CLASS_INTERFACE_TABLE_BEGIN()
    COM_CLASS_INTERFACE_TABLE_MEMBER(IClassName,IID_IClassName)
  COM_CLASS_INTERFACE_TABLE_END()
};
```

Wird zur Implementierung des Servers die MFC (afxdisp.h) verwendet, so soll hier vermerkt werden, daß über das Macro COM_CLASS_INTERFACE_TABLE_MEMBER in AddRef/Release die Funktionen AfxOleLockApp/AfxOleUnlockApp aufgerufen werden.

5. Ein Beispiel

Das folgende Beispiel ist im Bereich der statischen Berechnung gewählt. Es soll hier an einem Programm zur Berechnung von Querschnittswerten einer gegebenen Fläche veranschaulicht werden, daß durch die Benutzung der COMET+-Bibliothek der Implementierungsaufwand für die Unterstützung der COM-Technologie auf ein auch für kleinere Applikationen vertretbares Maß reduziert wurde.

Die Klasse Processor wird unter dem Namen "Processor.Object" registriert und bietet die Funktionen "AddNode" und "GetResult" über COM und Automation an.

Es folgen nun die Quellen unseres Beispiels:

Abbildung 9

interface.idl

```
import "oaidl.idl";

[
  object,dual,pointer_default(unique),
  uuid(9CEDE610-0116-11D1-86CA-E9A742759030)
]
interface IProcessor : IUnknown
{
  HRESULT AddNode([in] double X,[in] double Y);
  HRESULT GetResult([in] long Name,[out,retval] double * Result);
};

[
  uuid(9CEDE610-0116-11D1-86CA-E9A742759031),
```

```
  version(1.0)
]
library Processor
{
  [
    uuid(9CEDE610-0116-11D1-86CA-E9A742759032)
  ]
  coclass Processor
  {
    [default] interface IProcessor;
  };
};
```

Abbildung 10

processor.h

```
#ifndef __processor__
#define __processor__

////////////////////////////////////////////////////////////////////////////
// class Point2D

class Point2D : public CObject
{
  public:
  double _X;
  double _Y;
};

////////////////////////////////////////////////////////////////////////////
// class Processor

class Processor : public ComClass , public IProcessor
{
  COM_CLASS_INTERFACE_TABLE_BEGIN()
    COM_CLASS_INTERFACE_TABLE_MEMBER(IProcessor,IID_IProcessor)
  COM_CLASS_INTERFACE_TABLE_END()

  // Processor members
  private:
  CObList _Geometry;
  double _Xs,_Ys,_A,_Sx,_Sy,_Ix,_Iy,_Ixy;

  public:
  Processor();
  void Process();

  // IProcessor members
  public:
  HRESULT __stdcall AddNode(double X,double Y);
  HRESULT __stdcall GetResult(long Name,double * Result);
```

```
};

#endif
```

Abbildung 11

processor.cpp

```
#include <afxwin.h>
#include <afxdisp.h>
#include "interface.h"
#include "interface.c"
#include "Comet+.h"
#include "Comet+.cpp"
#include "resource.h"
#include "processor.h"

////////////////////////////////////////////////////////////////////////////
// class Processor

Processor::Processor()
{
  _Geometry.RemoveAll();
}

void Processor::Process()
{
  _A = 0.0;
  _Sx = 0.0;
  _Sy = 0.0;
  _Ix = 0.0;
  _Iy = 0.0;
  _Ixy = 0.0;

  Point2D * Point;
  double X,Y,X1,Y1;

  POSITION Position = _Geometry.GetHeadPosition();

  if (Position != NULL)
  {
    Point = (Point2D *) _Geometry.GetNext(Position);
    if (Point != NULL)
    {
      X1 = Point->_X - _Xs;
      Y1 = Point->_Y - _Ys;
    }
  }

  while (Position != NULL)
  {
    Point = (Point2D *) _Geometry.GetNext(Position);
```

```
    if (Point != NULL)
    {
      X = Point->_X - _Xs;
      Y = Point->_Y - _Ys;

      double A = (X1 * Y) - (X * Y1);
      double B = X1 + X;
      double C = Y1 + Y;
      double D = ((X1 * Y) + (X * Y1)) / 2;

      _A += A / 2;
      _Sx += (A * C) / 6;
      _Sy += (A * B) / 6;
      _Ix += (A * ((C * C) - (Y1 * Y))) / 12;
      _Iy += (A * ((B * B) - (X1 * X))) / 12;
      _Ixy += (A * ((B * C) - D)) / 12;

      X1 = X;
      Y1 = Y;
    }
  }
}

HRESULT Processor::AddNode(double X,double Y)
{
  Point2D * Point = new Point2D;
  Point->_X = X;
  Point->_Y = Y;
  _Geometry.AddTail(Point);
  return S_OK;
}

HRESULT Processor::GetResult(long Name,double * Result)
{
  _Xs = 0;
  _Ys = 0;
  Process();
  if (_A != 0)
  {
    double Sx = _Sx;
    double Sy = _Sy;
    _Xs = _Sy / _A;
    _Ys = _Sx / _A;
    Process();
    _Sx = Sx;
    _Sy = Sy;
  }

  if (_A < 0)
  {
    _A *= -1;
```

```
    _Sx *= -1;
    _Sy *= -1;
    _Ix *= -1;
    _Iy *= -1;
    _Ixy *= -1;
  }

  * Result = 0.0;

  if (Name == 0)
    * Result = _A;
  if (Name == 1)
    * Result = _Xs;
  if (Name == 2)
    * Result = _Ys;
  if (Name == 3)
    * Result = _Sx;
  if (Name == 4)
    * Result = _Sy;
  if (Name == 5)
    * Result = _Ix;
  if (Name == 6)
    * Result = _Iy;
  if (Name == 7)
    * Result = _Ixy;

  return S_OK;
}

////////////////////////////////////////////////////////////////////////////
// global COM managment for inproc servers

ComModule _ComModule;

COM_CLASS_TABLE_BEGIN()
  COM_CLASS_TABLE_MEMBER(Processor,CLSID_Processor,"Processor.Object")
COM_CLASS_TABLE_END()

extern "C" BOOL WINAPI DllMain(HINSTANCE Instance,DWORD Reason,void *)
{
  if (Reason == DLL_PROCESS_ATTACH)
  {
    _ComModule.Initialize(Instance);
    DisableThreadLibraryCalls(Instance);
  }
  if (Reason == DLL_PROCESS_DETACH)
  {
    _ComModule.Uninitialize();
  }

  return TRUE;
```

```
}

STDAPI DllGetClassObject(REFCLSID rclsid,REFIID riid,void ** ppv)
{
  return _ComModule.GetClassObject(rclsid,riid,ppv);
}

STDAPI DllCanUnloadNow()
{
  return _ComModule.CanUnloadNow();
}

STDAPI DllRegisterServer()
{
  return _ComModule.UpdateRegistry();
}
```

ANLAGE A:

COMET+ Sourcecode

comet+.h

```
#ifndef __comet+__
#define __comet+__

////////////////////////////////////////////////////////////////////////////
// ATL Ansi/OLE/Unicode string conversion

#include <atlconv.h>

////////////////////////////////////////////////////////////////////////////
// Component Object Model members

class ComClass;
class ComClassFactory;
class ComModule;

////////////////////////////////////////////////////////////////////////////
// class ComClass

class ComClass : public IDispatch
{
  // ComClass members
  protected:
  ULONG _ReferenceCounter;
  ComModule * _ClientSite;
  ITypeInfo * _TypeInformation;
  unsigned short _NumberOfInterfaces;

  public:
  ComClass();
  ~ComClass();
  void SetComModule(ComModule * ClientSite);
  void SetTypeInformation(ITypeInfo * TypeInformation);

  // IDispatch members
  public:
  HRESULT __stdcall GetTypeInfoCount(UINT * pctinfo);
  HRESULT __stdcall GetTypeInfo(UINT itinfo, LCID lcid,
                                ITypeInfo ** pptinfo);
  HRESULT __stdcall GetIDsOfNames(REFIID riid,OLECHAR ** rgszNames,
                                  UINT cNames,LCID lcid,DISPID * rgdispid);
  HRESULT __stdcall Invoke(DISPID dispidMember,REFIID riid,LCID lcid,
                           WORD wFlags,DISPPARAMS * pdispparams,
                           VARIANT * pvarResult,EXCEPINFO * pexcepinfo,
                           UINT * puArgErr);
};

////////////////////////////////////////////////////////////////////////////
// class ComClassFactory
```

```
class ComClassFactory : public IClassFactory
{
  // ComClassFactory members
  private:
  ULONG _ReferenceCounter;
  ComModule * _ClientSite;
  CLSID _ClassId;
  DWORD _Register;

  public:
  ComClassFactory();
  void SetComModule(ComModule * ClientSite);
  void SetClassId(CLSID ClassId);
  void SetRegister(DWORD Register);
  DWORD GetRegister();

  // IClassFactory members
  public:
  HRESULT __stdcall QueryInterface(REFIID riid,void ** ppv);
  ULONG   __stdcall AddRef();
  ULONG   __stdcall Release();
  HRESULT __stdcall CreateInstance(IUnknown * UnkOuter,
                                   REFIID riid,void ** ppv);
  HRESULT __stdcall LockServer(BOOL Lock);
};

////////////////////////////////////////////////////////////////////////
// class ComModule

class ComModule
{
  // ComModule members
  private:
  HINSTANCE _Instance;
  ULONG _GlobalReferenceCounter;
  ITypeLib * _TypeLibrary;

  public:
  ComModule();
  HRESULT Initialize(HINSTANCE Instance);
  HRESULT Uninitialize();
  HRESULT UpdateRegistry();
  HRESULT GetClassObject(REFCLSID rclsid,REFIID riid,void ** ppv);
  HRESULT CanUnloadNow();
  HRESULT ClassTable(long Command,CLSID ClassId,void ** ppv);
  HRESULT RegistryClass(CLSID ClassId,char * ProgId);
  HRESULT RegisterClass(CLSID ClassId,ComClassFactory ** Factory);
  HRESULT RevokeClass(ComClassFactory * Factory);
  ULONG GlobalAddRef();
  ULONG GlobalRelease();
  ITypeLib * GetTypeLibrary();
```

```
};

/////////////////////////////////////////////////////////////////////////////
// Component Object Model members

#define IID_IObject IID_IDispatch

#ifdef __AFXDISP_H__
#define ComClassLock()   AfxOleLockApp()
#define ComClassUnlock() AfxOleUnlockApp()
#else
#define ComClassLock()
#define ComClassUnlock()
#endif

#define COM_CLASS_TABLE_BEGIN()                    \
        HRESULT ComModule::ClassTable              \
          (long Command,CLSID ClassId,void ** ppv) \
        {                                          \
          HRESULT Result;                          \

#define COM_CLASS_TABLE_MEMBER(_Class,_ClassId,_ProgId)               \
          static ComClassFactory * _Com##_Class##ClassFactory = NULL; \
          if (Command == 0)                                           \
          {                                                           \
            Result = RegistryClass(_ClassId,_ProgId);                 \
            if (FAILED(Result))                                       \
              return Result;                                          \
          }                                                           \
          if (Command == 1)                                           \
          {                                                           \
            Result = RegisterClass(_ClassId,                          \
                                   & _Com##_Class##ClassFactory);      \
            if (FAILED(Result))                                       \
              return Result;                                          \
          }                                                           \
          if (Command == 2)                                           \
          {                                                           \
            Result = RevokeClass(_Com##_Class##ClassFactory);         \
            if (FAILED(Result))                                       \
              return Result;                                          \
          }                                                           \
          if ((Command == 3) && (ClassId == _ClassId))                \
          {                                                           \
            * ppv = new _Class;                                       \
            if (* ppv == NULL)                                        \
              return E_OUTOFMEMORY;                                   \
            return S_OK;                                              \
          }                                                           \
          if ((Command == 4) && (ClassId == _ClassId))                \
          {                                                           \
```

```
            * ppv = new ComClassFactory;                              \
            if (* ppv == NULL)                                        \
              return E_OUTOFMEMORY;                                   \
            return S_OK;                                              \
          }                                                           \

#define COM_CLASS_TABLE_END()                                         \
          if ((Command == 0) || (Command == 1) || (Command == 2))     \
            return S_OK;                                              \
          if ((Command == 3) || (Command == 4))                       \
            return CLASS_E_CLASSNOTAVAILABLE;                         \
          return S_FALSE;                                             \
        }                                                             \

#define COM_CLASS_INTERFACE_TABLE_BEGIN()                             \
        public:                                                       \
        ULONG __stdcall AddRef()                                      \
        {                                                             \
          ComClassLock();                                             \
          _ClientSite->GlobalAddRef();                                \
          return ++_ReferenceCounter;                                 \
        }                                                             \
        ULONG __stdcall Release()                                     \
        {                                                             \
          ComClassUnlock();                                           \
          _ClientSite->GlobalRelease();                               \
          if (--_ReferenceCounter == 0)                               \
          {                                                           \
            delete this;                                              \
            return 0;                                                 \
          }                                                           \
          return _ReferenceCounter;                                   \
        }                                                             \
        HRESULT __stdcall QueryInterface(REFIID riid,void ** ppv)     \
        {                                                             \
          * ppv = NULL;                                               \
          if (riid == IID_IUnknown)                                   \
          {                                                           \
            * ppv = this;                                             \
            AddRef();                                                 \
            return S_OK;                                              \
          }                                                           \
          if (riid == IID_IDispatch)                                  \
          {                                                           \
            * ppv = (IDispatch *) this;                               \
            AddRef();                                                 \
            return S_OK;                                              \
          }                                                           \

#define COM_CLASS_INTERFACE_TABLE_MEMBER(_Interface,_InterfaceId)     \
          if (riid == _InterfaceId)                                   \
```

```
          {                                                          \
            * ppv = (_Interface *) this;                             \
            AddRef();                                                \
            return S_OK;                                             \
          }                                                          \

#define COM_CLASS_INTERFACE_TABLE_END() \
           return E_NOINTERFACE;         \
         }                               \

#endif
```

comet+.cpp

```
#include "comet+.h"

////////////////////////////////////////////////////////////////////////
// ATL Ansi/OLE/Unicode string conversion

#include <atlconv.cpp>

USES_CONVERSION;

////////////////////////////////////////////////////////////////////////
// class ComClass

ComClass::ComClass()
{
  _ReferenceCounter = 0;
  _ClientSite = NULL;
  _TypeInformation = NULL;
  _NumberOfInterfaces = 0;
}

ComClass::~ComClass()
{
  if (_TypeInformation != NULL)
    _TypeInformation->Release();
}

void ComClass::SetComModule(ComModule * ClientSite)
{
  _ClientSite = ClientSite;
}

void ComClass::SetTypeInformation(ITypeInfo * TypeInformation)
{
  _TypeInformation = TypeInformation;

  if (_TypeInformation != NULL)
  {
```

```
    TYPEATTR * TypeAttr;
    if (SUCCEEDED(_TypeInformation->GetTypeAttr(& TypeAttr)))
    {
      _NumberOfInterfaces = TypeAttr->cImplTypes;
      _TypeInformation->ReleaseTypeAttr(TypeAttr);
    }
  }
}

HRESULT ComClass::GetTypeInfoCount(UINT * pctinfo)
{
  if (_TypeInformation == NULL)
    * pctinfo = 0;
  else
    * pctinfo = 1;
  return S_OK;
}

HRESULT ComClass::GetTypeInfo(UINT itinfo, LCID lcid,ITypeInfo ** pptinfo)
{
  if (_TypeInformation == NULL)
    return S_FALSE;

  * pptinfo = _TypeInformation;

  return S_OK;
}

HRESULT ComClass::GetIDsOfNames(REFIID riid,OLECHAR ** rgszNames,
                                UINT cNames,LCID lcid,DISPID * rgdispid)
{
  if (_TypeInformation == NULL)
    return S_FALSE;

  return (_TypeInformation->GetIDsOfNames(rgszNames,cNames,rgdispid));
}

HRESULT ComClass::Invoke(DISPID dispidMember,REFIID riid,LCID lcid,
                         WORD wFlags,DISPPARAMS * pdispparams,
                         VARIANT * pvarResult,EXCEPINFO * pexcepinfo,
                         UINT * puArgErr)
{
  if (_TypeInformation == NULL)
    return S_FALSE;

  HREFTYPE RefType;
  ITypeInfo * TypeInformation;
  BOOL Status = FALSE;

  for (int i = 0; (i < _NumberOfInterfaces) && (Status == FALSE); i++)
    if (SUCCEEDED(_TypeInformation->GetRefTypeOfImplType(i,& RefType)))
```

```
      if (SUCCEEDED(_TypeInformation->GetRefTypeInfo
           (RefType,& TypeInformation)))
      {
        TYPEATTR * TypeAttr;
        IUnknown * Interface;
        if (SUCCEEDED(TypeInformation->GetTypeAttr(& TypeAttr)))
        {
          if (SUCCEEDED(QueryInterface(TypeAttr->guid,
                                       (void **) & Interface)))
          {
            if (SUCCEEDED(TypeInformation->Invoke
                 (Interface,dispidMember,wFlags,pdispparams,
                  pvarResult,pexcepinfo,puArgErr)))
              Status = TRUE;

            Interface->Release();
          }
          TypeInformation->ReleaseTypeAttr(TypeAttr);
        }
      }

  if (Status == TRUE)
    return S_OK;
  else
    return DISP_E_MEMBERNOTFOUND;
}

//////////////////////////////////////////////////////////////////////////
// class ComClassFactory

ComClassFactory::ComClassFactory()
{
  _ReferenceCounter = 0;
  _ClientSite = NULL;
  _ClassId = GUID_NULL;
}

void ComClassFactory::SetComModule(ComModule * ClientSite)
{
  _ClientSite = ClientSite;
}

void ComClassFactory::SetClassId(CLSID ClassId)
{
  _ClassId = ClassId;
}

void ComClassFactory::SetRegister(DWORD Register)
{
  _Register = Register;
}
```

```
DWORD ComClassFactory::GetRegister()
{
  return _Register;
}

HRESULT ComClassFactory::QueryInterface(REFIID riid,void ** ppv)
{
  * ppv = NULL;

  if (riid == IID_IUnknown)
  {
    * ppv = this;
    AddRef();
    return S_OK;
  }

  if (riid == IID_IClassFactory)
  {
    * ppv = (IClassFactory *) this;
    AddRef();
    return S_OK;
  }

  return E_NOINTERFACE;
}

ULONG ComClassFactory::AddRef()
{
  _ClientSite->GlobalAddRef();
  return (++_ReferenceCounter);
}

ULONG ComClassFactory::Release()
{
  _ClientSite->GlobalRelease();
  if (--_ReferenceCounter == 0)
  {
    delete this;
    return 0;
  }
  return _ReferenceCounter;
}

HRESULT ComClassFactory::CreateInstance
  (IUnknown * UnkOuter,REFIID riid,void ** ppv)
{
  HRESULT Result;

  * ppv = NULL;
```

```
  if (UnkOuter != NULL)
    return CLASS_E_NOAGGREGATION;

  Result = _ClientSite->ClassTable(3,_ClassId,ppv);

  if (FAILED(Result))
  {
    * ppv = NULL;
    return Result;
  }

  ((ComClass *) * ppv)->SetComModule(_ClientSite);

  ITypeLib * TypeLibrary = _ClientSite->GetTypeLibrary();
  ITypeInfo * TypeInformation = NULL;

  if (TypeLibrary != NULL)
    TypeLibrary->GetTypeInfoOfGuid(_ClassId,& TypeInformation);

  ((ComClass *) * ppv)->SetTypeInformation(TypeInformation);

  Result = ((IUnknown *) * ppv)->QueryInterface(riid,ppv);

  if (FAILED(Result))
    delete (* ppv);

  return Result;
}

HRESULT ComClassFactory::LockServer(BOOL Lock)
{
  if (Lock)
    AddRef();
  else
    Release();

  return S_OK;
}

////////////////////////////////////////////////////////////////////////////
// class ComModule

ComModule::ComModule()
{
  _GlobalReferenceCounter = 0;
  _TypeLibrary = NULL;
}

HRESULT ComModule::Initialize(HINSTANCE Instance)
{
  _Instance = Instance;
```

```
  TCHAR FileName[_MAX_PATH];

  if (::GetModuleFileName(_Instance,FileName,_MAX_PATH) == 0)
    return S_FALSE;

  HRESULT Result;

  Result = LoadTypeLib(T2OLE(FileName),& _TypeLibrary);
  if (FAILED(Result))
    _TypeLibrary = NULL;

  #ifndef _WINDLL
  Result = CoInitialize(NULL);
  if (FAILED(Result))
    return Result;
  return ClassTable(1,GUID_NULL,NULL);
  #else
  return S_OK;
  #endif
}

HRESULT ComModule::Uninitialize()
{
  if (_TypeLibrary != NULL)
    _TypeLibrary->Release();

  #ifndef _WINDLL
  HRESULT Result = ClassTable(2,GUID_NULL,NULL);
  CoUninitialize();
  return Result;
  #else
  return S_OK;
  #endif
}

HRESULT ComModule::UpdateRegistry()
{
  TCHAR FileName[_MAX_PATH];

  if (::GetModuleFileName(_Instance,FileName,_MAX_PATH) != 0)
  {
    ITypeLib * TypeLibrary = NULL;

    if (SUCCEEDED(LoadTypeLib(T2OLE(FileName),& TypeLibrary)))
    {
      RegisterTypeLib(TypeLibrary,T2OLE(FileName),NULL);
      TypeLibrary->Release();
    }
  }
```

```
  return ClassTable(0,GUID_NULL,NULL);
}

HRESULT ComModule::GetClassObject(REFCLSID rclsid,REFIID riid,void ** ppv)
{
  * ppv = NULL;

  HRESULT Result;
  ComClassFactory * Factory;

  Result = ClassTable(4,rclsid,(void **) & Factory);

  if (FAILED(Result))
    return Result;

  Factory->SetComModule(this);
  Factory->SetClassId(rclsid);

  Result = Factory->QueryInterface(riid,ppv);

  if (FAILED(Result))
  {
    delete Factory;
    return Result;
  }

  return S_OK;
}

HRESULT ComModule::CanUnloadNow()
{
  if (_GlobalReferenceCounter == 0)
    return S_OK;
  else
    return S_FALSE;
}

HRESULT ComModule::RegistryClass(CLSID ClassId,char * ProgId)
{
  HKEY Key,ClassIdKey,ProgIdKey,Server32Key;

  LPOLESTR ClassIdString;
  ::StringFromCLSID(ClassId,& ClassIdString);

  TCHAR FileName[_MAX_PATH];

  if (::GetModuleFileName(_Instance,FileName,_MAX_PATH) == 0)
    return S_FALSE;

  if (::RegCreateKey(HKEY_CLASSES_ROOT,_T("CLSID\\"),
                     & Key) != ERROR_SUCCESS)
```

```
  return S_FALSE;

// Create HKEY_CLASSES_ROOT//CLSID//{...}
if (::RegCreateKey(Key,OLE2T(ClassIdString),
                   & ClassIdKey) != ERROR_SUCCESS)
{
  ::RegCloseKey(Key);
  return S_FALSE;
}

::RegCloseKey(Key);

if (::RegSetValue(ClassIdKey,NULL,REG_SZ,A2T(ProgId),
                  strlen(ProgId)) != ERROR_SUCCESS)
{
  ::RegCloseKey(ClassIdKey);
  return S_FALSE;
}

// Create HKEY_CLASSES_ROOT//CLSID//{...}//ProgId
if (::RegCreateKey(ClassIdKey,_T("ProgID"),
                   & ProgIdKey) != ERROR_SUCCESS)
{
  ::RegCloseKey(ClassIdKey);
  return S_FALSE;
}

if (::RegSetValue(ProgIdKey,NULL,REG_SZ,A2T(ProgId),
                  strlen(ProgId)) != ERROR_SUCCESS)
{
  ::RegCloseKey(ProgIdKey);
  ::RegCloseKey(ClassIdKey);
  return S_FALSE;
}

::RegCloseKey(ProgIdKey);

#ifndef _WINDLL

// Create HKEY_CLASSES_ROOT//CLSID//{...}//LocalServer32
if (::RegCreateKey(ClassIdKey,_T("LocalServer32"),
                   & Server32Key) != ERROR_SUCCESS)
{
  ::RegCloseKey(ClassIdKey);
  return S_FALSE;
}

#else

// Create HKEY_CLASSES_ROOT//CLSID//{...}//InprocServer32
if (::RegCreateKey(ClassIdKey,_T("InprocServer32"),
```

```
                    & Server32Key) != ERROR_SUCCESS)
{
  ::RegCloseKey(ClassIdKey);
  return S_FALSE;
}

#endif

if (::RegSetValue(Server32Key,NULL,REG_SZ,FileName,
                  strlen(T2A(FileName))) != ERROR_SUCCESS)
{
  ::RegCloseKey(Server32Key);
  ::RegCloseKey(ClassIdKey);
  return S_FALSE;
}

::RegCloseKey(Server32Key);
::RegCloseKey(ClassIdKey);

// Create HKEY_CLASSES_ROOT//ProgId
if (::RegCreateKey(HKEY_CLASSES_ROOT,A2T(ProgId),
                   & ProgIdKey) != ERROR_SUCCESS)
  return S_FALSE;

if (::RegSetValue(ProgIdKey,NULL,REG_SZ,A2T(ProgId),
                  strlen(ProgId)) != ERROR_SUCCESS)
{
  ::RegCloseKey(ProgIdKey);
  return S_FALSE;
}

// Create HKEY_CLASSES_ROOT//ProgId//CLSID
if (::RegCreateKey(ProgIdKey,_T("CLSID\\"),
                   & ClassIdKey) != ERROR_SUCCESS)
{
  ::RegCloseKey(ProgIdKey);
  return S_FALSE;
}

if (::RegSetValue(ClassIdKey,NULL,REG_SZ,OLE2T(ClassIdString),
                  strlen(OLE2A(ClassIdString))) != ERROR_SUCCESS)
{
  ::RegCloseKey(ClassIdKey);
  ::RegCloseKey(ProgIdKey);
  return S_FALSE;
}

::RegCloseKey(ClassIdKey);
::RegCloseKey(ProgIdKey);

return S_OK;
```

```
}

HRESULT ComModule::RegisterClass(CLSID ClassId,ComClassFactory ** Factory)
{
  HRESULT Result;
  IUnknown * Unknown;
  DWORD Register;

  * Factory = new ComClassFactory;

  if (* Factory == NULL)
    return E_OUTOFMEMORY;

  (* Factory)->SetComModule(this);
  (* Factory)->SetClassId(ClassId);

  Result = (* Factory)->QueryInterface(IID_IUnknown,(void **) & Unknown);

  if (FAILED(Result))
    return Result;

  Result = ::CoRegisterClassObject(ClassId,Unknown,CLSCTX_LOCAL_SERVER,
                                   REGCLS_MULTIPLEUSE,& Register);
  if (FAILED(Result))
  {
    Unknown->Release();
    return Result;
  }

  (* Factory)->SetRegister(Register);

  return S_OK;
}

HRESULT ComModule::RevokeClass(ComClassFactory * Factory)
{
  if (Factory == NULL)
    return S_FALSE;

  HRESULT Result = ::CoRevokeClassObject(Factory->GetRegister());

  if (FAILED(Result))
    return Result;

  if (Factory->Release() != 0)
    delete Factory;

  return S_OK;
}

ULONG ComModule::GlobalAddRef()
```

```
{
  return (++_GlobalReferenceCounter);
}

ULONG ComModule::GlobalRelease()
{
  if (_GlobalReferenceCounter != 0)
    _GlobalReferenceCounter--;

  return _GlobalReferenceCounter;
}

ITypeLib * ComModule::GetTypeLibrary()
{
  return _TypeLibrary;
}
```