Delphi - Komponenten entwickeln

Wolfgang Zelinka

Für die Entwicklung eines professionellen Windows-Programms ist es oft unerläßlich, eigene spezielle Komponenten zu verwenden. Diese müssen dann selbst entwickelt und codiert (programmiert) werden.

Das derzeit wohl bekannteste Programm zum visuellen Entwickeln von Windows-Anwendungen ist Visual Basic. Visual Basic 3 hat aber eine Reihe von Nachteilen, die es für manche professionelle Entwicklungen unbrauchbar macht. Neben der relativ langsamen Abarbeitung der Befehle (Interpreter) ist ein gravierender Nachteil die Unmöglichkeit der Vererbung und der Erstellung von eigenen neuen Komponenten in Visual Basic selbst. Wenn Sie zum Erzeugen von VBXs zB. C++ benötigten, dann stellt sich die Frage, ob es nicht besser wäre, gleich Visual C++ anstelle von Visual Basic zu verwenden.

Borland Delphi hingegen enthält alles, was Sie für die Erzeugung von Komponenten benötigen. Objekt Orientierte Programmierung (OOP) und Vererbung sind daher in Delphi selbstverständlich. Die Bausteine einer jeden Delphi-Anwendung sind Komponenten. Die meisten Komponenten sind sichtbare Teile einer Benutzeroberfläche. Einige Komponenten sind auch nicht sichtbare Elemente in einem Programm wie beispielsweise Timer oder Datenbanken.

Für den Programmentwickler sind Komponenten einzelne Elemente (Objekte), die aus einer Palette ausgewählt werden können, im Formular Designer oder durch Programmcode manipuliert werden und dann in einer Anwendung zum Einsatz kommen. Aus der Sicht des Komponentenentwicklers ist eine Komponente jedoch ein Objekt im Programmcode. In Delphi wird in der Sprache Pascal programmiert.

Jedes Objekt, das vom Typ TComponent abgeleitet wurde, stellt eine Komponente dar. TComponent definiert das für die Delphi-Umgebung erforderliche grundlegende Verhalten, das alle Komponenten aufweisen müssen. Spezielle Ergänzungen müssen Sie selbst festlegen und sich dabei eng an die Standards und Konventionen halten, die im Handbuch „Komponentenentwicklung“ beschrieben werden.

Drei wichtige Unterschiede existieren zwischen der Entwicklung einer Komponente für den Einsatz in Delphi und der gängigeren Aufgabenstellung, eine Windows-Anwendung zu erzeugen, die mit Komponenten arbeitet:

- Das Entwickeln von Komponenten ist nicht visuell
- Das Entwickeln von Komponenten verlangt fundiertes Wissen über Objekte
- Das Entwickeln von Komponenten verlangt die Einhaltung mehrerer Konventionen

Beim Erzeugen einer neuen Komponente muß ein neuer Objekttyp aus einem bereits bestehenden abgeleitet und neue Eigenschaften und Methoden hinzugefügt werden. Hiebei greifen Sie auf Teile des Vorfahrobjekts zu, die dem Anwendungsprogrammierer nicht zur Verfügung stehen. Diese Teile, die nur für Komponentenentwickler gedacht sind, werden gemeinsam als *geschützte Schnittstelle* zu den Objekten bezeichnet. Anwender von Komponenten verwenden hingegen bestehende Komponenten und passen ihr Verhalten durch Ändern von Eigenschaften und Festlegen von Reaktionen auf Ereignisse beim Programmieren an.

Die Entwicklung einer eigenen Komponente gliedert sich in folgende Schritte:

1. Erzeugen einer Unit für die neue Komponente.
2. Ableitung eines Komponententyps aus einem bereits bestehenden Komponententyp.
3. Hinzufügen von Eigenschaften, Methoden und Ereignissen, die benötigt werden.
4. Registrieren Ihrer Komponente bei Delphi.
5. Erzeugen einer Hilfedatei für Ihre Komponente, ihre Eigenschaften, Methoden und Ereignisse.

Der letzte Punkt (5) ist nicht unbedingt erforderlich, erleichtert aber das Arbeiten mit der neuen Komponente.

Die fertige Komponente besteht aus folgenden Dateien:

1 Die Pascal-Quellendatei als Unit (.PAS-Datei)
2 Die compilierte Unit (.DCU-Datei)
3 Eine Paletten-Bitmap (.DCR-Datei)
4 Eine Hilfedatei (.HLP-Datei)
5 Eine Hilfe-Schlüsselwort-Datei (.KWF-Datei)

Nur die compilierte Unit (.DCU-Datei) ist für den weiteren Einsatz in Anwendungen unbedingt notwendig.

Die Visuelle Komponenten-Bibliothek

Delphis Komponenten gehören alle zu einer Objekthierarchie, die Visuelle Komponenten-Bibliothek (Visual Component Library - VCL) genannt wird. Der Typ TComponent ist der gemeinsame Vorfahre aller Komponenten in der VCL. TComponent stellt die Minimaleigenschaften und -ereignisse zur Verfügung, die eine Komponente zum Funktionieren in Delphi benötigt. (Die nachstehende Objekthierarchie stellt nicht alle Abhängikeiten dar)

```
TObject
   +-...
   +-TPersistent
      +-...
      +-TComponent
         +-TGlobalComponent     TApplication
         +-...
         +-TControl
            +-TGraphicControl
            +-TWinControl
               +-TScrollBar
               +-...
               +-TCustomComboBox
               +-TCustomListBox
               +-TCustomEdit
               +-TScrollingWinControl  ->TForm
               +-TButtonControl     ->TButton
               +-TCustomControl     ->TCustomPanel   ->TPanel
```

Die folgenden Verfahren sind zum Entwickeln einer neuen Komponente möglich:

- Anpassen einer bestehenden Komponente: Jede bereits bestehende Komponente, oder ein abstrakter Komponententyp wie beispielsweise TCustomListBox.
- Entwickeln eines eigenen Dialogelements: TCustomControl
- Entwickeln eines grafischen Dialogelements: TGraphicControl
- Verwenden eines bestehenden Windows-Dialogelements: TWinControl
- Entwickeln einer nicht-visuellen Komponente: TComponent

Eine Komponente enthält:

Eigenschaften vermitteln dem Komponentenanwender die Illusion, daß er den Wert einer Variablen in der Komponente setzt oder ausliest. Dadurch stellen Sie auch sicher, daß der aus der Eigenschaft ausgelesene Wert immer gültig ist. Der Komponentenentwickler hingegen verbirgt dadurch die zugrundeliegende Datenstruktur oder eingebaute Nebeneffekte beim Zugriff auf den Wert.

Ereignisse verbinden Vorkommnissen (wie beispielsweise Mausaktionen oder Tastatureingaben) und Programmcode.

Methoden sind Prozeduren oder Funktionen, die in eine Komponente eingebaut sind. Methoden sind nur zur Laufzeit verfügbar, da für sie Programmcode ausgeführt werden muß.

Grafikkapselung vereinfacht die Handhabung von Windows-Grafiken durch Kapselung der diversen Grafikwerkzeuge in einem *Canvas* (Zeichenfläche).

Die **Registrierung** informiert Delphi darüber, wo Ihre Komponente auf der Komponentenpalette erscheinen soll.

Schutzklassen für Objektteile

private Implementierungsdetails verbergen: Der Code ist nur innerhalb dieses Objektes zugänglich.

protected Die Entwicklerschnittstelle definieren: Der Code ist nur innerhalb dieser Unit zugänglich. Von diesem Objekttyp abgeleitete Objekttypen können aber darauf zugreifen. Benutzer haben daher keinen Zugriff auf die geschützten Teile, Entwickler aber sehr wohl.

public Die Laufzeitschnittstelle definieren: Diese Teile sind für jeden Code zugänglich, dar auf das Objekt als Ganzes Zugriff hat. Sie stehen aber zur Entwurfszeit nicht zur Verfügung (zB: Eigenschaften, die nur Lesezugriff gestatten).

published Die Schnittstelle für den Entwurfsmodus definieren (Objektinspektor): Diese Teile gehören zur Schutzklasse public, und es werden außerdem Laufzeit-Typinformationen für diese Teile generiert. Dadurch kann der Objektinspektor auf diese Eigenschaften und Ereignisse zugreifen.

Schutz von Methoden

Methoden Ihrer Komponente sollten entweder public oder protected sein. Ausgenommen jene Methoden, die Eigenschaften implementieren; diese sollten immer private sein.

Hinweis: Konstruktoren und Destruktoren sollten *immer* public sein.

Wenn alle Implementierungsmethoden der Komponente protected sind, können Sie nicht vom Quelltext eines Benutzers aufgerufen werden, sind aber für abgeleitete Objekte verfügbar.

Aufgabenstellung für das Beispiel: SuperButton

Da ein normaler Button die Änderung seiner Farbe nicht zuläßt und ebenso nur als Taster funktioniert, soll ein neuer Button mit folgenden Eigenschaften erstellt werden:

1) Er soll auch als Schalter funktionieren können, dh. er soll einen EIN- und einen AUS-Zustand haben.
2) Je nach Zustand sollen die Farbe, die Textfarbe und der Text unterschiedlich sein können.
3) In allen anderen Eigenschaften soll er möglicht wie ein normaler Button reagieren.

Im nachfolgenden Beispiel wird gezeigt, wie Sie eine neue Komponente von einer bestehenden ableiten, bestehende Eigenschaften ändern und neue Eigenschaften hinzufügen. Die Zeilen des Listings der Unit sind durchnummeriert und werden jeweils darauffolgend erklärt. Die fehlenden Zeilennummer enthalten entweder Leerzeilen oder Erweiterungen, die vollständig auf der Diskette vorhanden sind, aber im Rahmen dieses Artikels nicht besprochen werden.

Z.1..7: Definition der Unit, Beginn des Interface-Teils und Angabe der verwendeten Units.

```
1: unit Subut;
3: interface
5: uses
6:   SysUtils, WinTypes, WinProcs, Messages, Classes,
Graphics, Controls,
7:   Forms, Dialogs, ExtCtrls, Menus;
```

Z.9..13: Beginn der Typ-Vereinbarungen

```
 9: type
11:   TStatus = (stEin, stAus);
13:   TSuperButton = class(TCustomPanel)
```

11 = Aufzählungstyp für den Status.

13 = Ein TPanel hat alle erforderlichen Eigenschaften, die für die neue Komponente gefordert werden. Da aber TButton und TPanel teilweise über unterschiedliche Eigenschaften verfügen, ist es notwendig diese anzugleichen. Daher wird der TSuperButton von TCustomPanel abgeleitet. Dadurch ist es möglich, nur jene Eigenschaften durch published freizugeben, die benötigt werden.

Z.15..30: Private-Deklarationen

```
15:   private
18:     FColorAus, FColorEin: TColor;
20:     FModalResult: TModalResult;
21:     FStatus: TStatus;
22:     FToggle: boolean;
25:     procedure SetColorAus(AColor: TColor);
26:     procedure SetColorEin(AColor: TColor);
29:     procedure SetStatus(AStatus: TStatus);
30:     procedure SetToggle(AToggle: boolean);
```

Die Eigenschaften ColorAus, ColorEin, Status und Toggle werden in den internen Objektfeldern FColorAus, FColorEin, FStatus und FToggle gespeichert und sollen private deklariert sein (Z.18..22). Gemäß Konvention beginnt der Bezeichner für dieses Objektfeld mit dem Buchstaben F gefolgt von dem Namen der Eigenschaft.

Die read/write-Methoden zum Setzen der obigen Eigenschaften werden mit Get/Set, gefolgt von dem Namen der Eigenschaft benannt (SetColorAus, SetColorEin, SetStatus und SetToggle). Es werden nur write-Methoden benötigt, da beim Auslesen direkt die interne Variable gelesen werden darf.

Z.34..39: Public-Deklarationen

```
34:   public
35:     constructor Create(AOwner: TComponent); override;
36:     procedure Loaded; override;
37:     procedure Click; override;
38:     procedure MouseDown(Button:TMouseButton;
Shift:TShiftState; X,Y:integer); override;
39:     procedure MouseUp(Button:TMouseButton;
Shift:TShiftState; X,Y:integer); override;
```

Z.35 = Deklaration des Konstruktors, der den vom Vorfahr überschreibt.

Z.36..39 = Deklarationen zum Anpassen des Verhaltens vom Vorfahr TCustomPanel an die Ähnlichkeit mit TButton.

Z.41..94: Published-Deklarationen. Diese Eigenschaften sind im Entwurfsmodus im Objektinspektor verfügbar.

```
41:   published
45:     property ColorAus: TColor read FColorAus write
SetColorAus default clRed;
46:     property ColorEin: TColor read FColorEin write
SetColorEin default clLime;
49:     property ModalResult: TModalResult read FModalResult
write FModalResult default 0;
50:     property Status: TStatus read FStatus write SetStatus
default stAus;
51:     property Toggle: boolean read FToggle write SetToggle
default False;
```

Veröffentlichen geerbter Eigenschaften erfolgt durch Neudeklaration dieser ererbten Eigenschaft als **published**, wodurch Sie im Entwurfsmodus zur Verfügung steht und somit auch im Objektinspektor editierbar ist.

Hier sind die neuen Eigenschaften ColorAus, ColorEin, Status und Toggle deklariert mit direktem Zugriff auf die jeweiligen internen Objektfelder FColorAus, FColorEin, FStatus und FToggle für Lesen und über die jeweiligen Zugriffsmethoden SetColorAus, SetColorEin, SetStatus und SetToggle für Schreiben und festlegen eines Defaultwertes. Dieser Standardwert (default ...) ist optional und ist jener Wert, der für diese Eigenschaft im Konstruktor der Komponente gesetzt ist. Delphi stellt anhand des deklarierten Standardwertes fest, ob eine Eigenschaft in einer Formulardatei gespeichert werden soll. Die Deklaration eines Standardwerts in der Eigenschaftsdeklaration setzt die Eigenschaft nicht automatisch auf diesen Wert. Sie als Komponentenentwickler müssen sicherstellen, *daß der Konstruktor der Komponente die Eigenschaft tatsächlich auf den eingegebenen Wert setzt!.*

Aus der VCL wurden aus der Quellendatei EXTCTRLS.PAS die Eigenschaften, die von TPanel=class(TCustomPanel) freigegeben wurden, auch hier eingefügt und entsprechende Änderungen vorgenommen:

Eigenschaften:

```
59:     property Align;
60:     property Alignment;
61:     property BevelInner;
62:     property BevelOuter;
63:     property BevelWidth;
64:     property BorderWidth;
65:     property BorderStyle;
```

```
66:     property Caption;
67:   { property Color;    ersetzt durch ColorAus & ColorEin }
```

Durch die Erweiterung von TSuperButton mit den Eigenschaften ColorEin und ColorAus ist Color unnötig geworden und wird daher nicht mehr in published angeführt und wird auch nicht mehr im Objektinspektor angezeigt.

```
68:   { property Ctl3D;           in TButton nicht vorhanden }
69:     property DragCursor;
70:     property DragMode;
71:     property Enabled;
72:     property Font;
73:   { property Locked;           in TButton nicht vorhanden }
74:   { property ParentColor;      in TButton nicht vorhanden }
75:   { property ParentCtl3D;      in TButton nicht vorhanden }
76:     property ParentFont;
77:     property ParentShowHint;
78:     property PopupMenu;
79:     property ShowHint;
80:     property TabOrder;
81:     property TabStop;
82:     property Visible;
```

Gegenüber dem TButton fehlen noch die Eigenschaften Cancel und Default.

Ereignisse:

```
84:     property OnClick;
85:   { property OnDblClick;       in TButton nicht vorhanden }
86:     property OnDragDrop;
87:     property OnDragOver;
88:     property OnEndDrag;
89:     property OnEnter;
90:     property OnExit;
91:     property OnMouseDown;
92:     property OnMouseMove;
93:     property OnMouseUp;
94:   { property OnResize;         in TButton nicht vorhanden }
```

Gegenüber dem TButton fehlen noch die Ereignisse OnKeyDown, OnKeyPress und OnKeyUp.

Z.96..101: Ende der Typ-Vereinbarungen, Definition der Registrierung, Beginn des Implementation-Teils.

```
096:   end;
098: procedure Register;
101: implementation
```

Es folgen nun alle Prozeduren, die im Interface-Teil definiert wurden:

Z.103..122: Implementation des Konstruktors, der auch das Erscheinungsbild der neuen Komponente im Entwurfsmodus festlegt.

Z.105 = Aufruf von Create des Vorfahrs, gefolgt von den Zuweisungen, mit welchen Eigenschaftswerten die Komponente beim Einfügen ins Formular erscheinen soll. Die angegebenen Werte sind die gleichen, die bei der Deklaration der Eigenschaften als default-Werte angegeben wurden!

Die Eigenschaften FColorEin und FColorAus steuern die Eigenschaft Color, die im Vorfahr (TCustomPanel) deklariert ist und die Farbe der Fläche festlegt.

```
103: constructor TSuperButton.Create(AOwner: TComponent);
104: begin
105:   inherited Create(AOwner);
107:   Height := 33;              { Anpassungen der
Abmessungen an TButton }
108:   Width  := 89;              { -"- }
109:   BevelInner := bvRaised;
110:   BevelOuter := bvRaised;
114:   FColorAus := clRed;
115:   FColorEin := clLime;
116:   Color := FColorAus;
120:   FStatus := stAus;
121:   FToggle := False;
122: end;
```

Wenn auf die Eigenschaft Status schreibend zugegriffen wird, dann wird diese Behandlungsroutine aufgerufen:

```
124: procedure TSuperButton.SetStatus(AStatus: TStatus);
125: begin
126:   if AStatus <> FStatus then
127:   begin
128:     FStatus := AStatus;
129:     if AStatus = stEin
130:     then begin
131:            BevelInner := bvLowered;
132:            BevelOuter := bvLowered;
134:            Color := FColorEin;
136:          end
137:     else begin
138:            BevelInner := bvRaised;
139:            BevelOuter := bvRaised;
141:            Color := FColorAus;
143:          end;
144:     Invalidate;
145:   end;
146: end;
```

Z.126 = Zuerst wird überprüft, ob sich der Wert auch wirklich geändert hat.

Z.128 = Die interne Variable wird dann auf den neuen Wert gesetzt.

Z.129 = Je nachdem, welcher Wert, wird unterschiedlicher Programmcode durchgeführt.

Z.131..134 = Setzen der Eigenschaften für den Zustand stEin.

Z.138..141 = Setzen der Eigenschaften für den Zustand stAus.

Z.144 = Neuzeichnen der Komponente erzwingen.

Z.179..203: Wird die Eigenschaft ColorAus/ColorEin verändert, dann wird die entsprechende Behandlungsroutine SetColorAus/SetColorEin aufgerufen. Sie sind nach dem gleichen Schema wie SetStatus aufgebaut:

```
179: procedure TSuperButton.SetColorAus(AColor: TColor);
180: begin
181:   if AColor <> FColorAus then
182:   begin
183:     FColorAus := AColor;
184:     if FStatus = stAus then
185:     begin
186:       Color := AColor;
187:       Invalidate;
188:     end;
189:   end;
190: end;
191:
192: procedure TSuperButton.SetColorEin(AColor: TColor);
193: begin
194:   if AColor <> FColorEin then
195:   begin
196:     FColorEin := AColor;
197:     if FStatus = stEin then
198:     begin
199:       Color := AColor;
200:       Invalidate;
201:     end;
202:   end;
203: end;
```

Ändern der Behandlung von Standardereignissen

Wenn Ihre benutzerdefinierte Komponente auf ein bestimmtes Ereignis anders reagieren soll, so können Sie die geschützte Methode überschreiben, und durch Aufruf der ererbten Methode können Sie die Standardbehandlung einschließlich des Ereignisses für den Anwendercode beibehalten.

Ein Toggle-Switch wird realisiert durch überschreiben des Standardereignisses Click, worin des Status gewechselt wird, aber nur dann, wenn die Eigenschaft Toggle=true ist.

```
148: procedure TSuperButton.SetToggle(AToggle: boolean);
149: begin
150:   if AToggle <> FToggle then FToggle := AToggle;
151: end;
```

Z. 231..254: Die folgenden Behandlungsroutinen Click, MouseDown und MouseUp dienen zur Anpassung des Verhaltens von TCustomPanel (Vorfahr von TSuperButton) an das Verhalten eines TButton.

```
231: procedure TSuperButton.Click;
232: var   Form: TForm;
233: begin
234:   if FToggle
235:   then begin if FStatus = stEin then Status := stAus
                              else Status := stEin;
236:        end
237:   else begin { Codesequenz aus TButton.Click }
```

```
238:          Form := GetParentForm(Self);
239:          if Form <> nil then
          Form.ModalResult := ModalResult;
240:        end;
241:   inherited Click;   { Aufruf der Methode des Vorfahrs }
242: end;
243:
244: procedure TSuperButton.MouseDown(Button:TMouseButton;
                              Shift:TShiftState;
                              X,Y:integer);
245: begin
246:   if not FToggle then Status := stEin;
247:   inherited MouseDown(Button, Shift, X, Y );
248: end;
249:
250: procedure TSuperButton.MouseUp(Button:TMouseButton;
                            Shift:TShiftState;
                            X,Y:integer);
251: begin
252:   if not FToggle then Status := stAus;
253:   inherited MouseUp(Button, Shift, X, Y );
254: end;
```

Durch die Erweiterung von TSuperButton mit den Eigenschaften ColorEin und ColorAus ist Color unnötig geworden und wird daher nicht mehr in published angeführt und wird auch nicht mehr im Objektinspektor angezeigt. Wenn diese neue Eigenschaft im Objektinspektor geändert wird, wird dies aber nicht im Formular sichtbar. Damit aber die gesetzte Eigenschaft dargestellt wird, muß die Komponente nach dem Laden ihrer Eigenschaftswerte noch initialisiert werden. Dies erfolgt mit der Methode Loaded, die die nunmehr nicht mehr sichtbare Eigenschaft Color setzt und anschließend das Objekt neu zeichnet. **(Z. 256..271)**

```
256: procedure TSuperButton.Loaded;
257: begin
258:   inherited Loaded;
259:   if FStatus=stEin
260:   then begin
262:          Color := FColorEin;
264:        end
265:   else begin
267:          Color := FColorAus;
269:        end;
270:   Invalidate;
271: end;
```

Z.273..279: Zum Abschluß muß die neue Komponente noch bei Delphi registriert werden, wofür die Prozedur Register zuständig ist. In Z.279 wird die Unit beendet.

```
273: procedure Register;
274: begin
275:   RegisterComponents('Beispiele', [TSuperButton]);
276: end;
279: end.
```

Auf der Diskette sind noch weitere Eigenschaften ergänzt: FontColorEin, FontColorAus und statt Caption sind CaptionAus und CaptionEin vorhanden.

Unerläßlich für das Verständnis des Verhaltens waren aber die Quelldateien aus der VCL, die im Delphi V1.0 Paket nicht enthalten waren und zusätzlich gekauft wurden.

Trotz einiger kleiner Problem war das Erstellen einer neuen Komponente aber recht einfach und das Ergebnis zeigt auch einen gut verständlichen Programmcode. Ein großer Vorteil ist aber darin zu sehen, daß alles in Delphi selbst (Pascal) zu realisieren ist und nicht eine weitere Programmsprache gelernt werden muß. □

```
Master Programmer
=================
[
uuid(2573F8F4-CFEE-101A-9A9F-00AA00342820)
]
library LHello
{
// bring in the master library
importlib("actimp.tlb");
importlib("actexp.tlb");

// bring in my interfaces
#include "pshlo.idl"

[
uuid(2573F8F5-CFEE-101A-9A9F-00AA00342820)
]
cotype THello
{
interface IHello;
interface IPersistFile;
};
};

[
exe,
uuid(2573F890-CFEE-101A-9A9F-00AA00342820)
]
module CHelloLib
{

// some code related header files
importheader(<windows.h>);
importheader(<ole2.h>);
importheader(<except.hxx>);
importheader("pshlo.h");
importheader("shlo.hxx");
importheader("mycls.hxx");

// needed typelibs
importlib("actimp.tlb");
importlib("actexp.tlb");
importlib("thlo.tlb");

[
uuid(2573F891-CFEE-101A-9A9F-
00AA00342820), aggregatable
]
coclass CHello
{
cotype THello;
};
};


#include "ipfix.hxx"

extern HANDLE hEvent;

class CHello : public CHelloBase
{
public:
IPFIX(CLSID_CHello);

CHello(IUnknown *pUnk);
~CHello();

HRESULT  __stdcall PrintSz(LPWSTR
pwszString);

private:
static int cObjRef;
};


#include <windows.h>
#include <ole2.h>
#include <stdio.h>
#include <stdlib.h>
#include "thlo.h"
#include "pshlo.h"
#include "shlo.hxx"
#include "mycls.hxx"

int CHello::cObjRef = 0;

CHello::CHello(IUnknown *pUnk) :
CHelloBase(pUnk) {
cObjRef++;
return;
}

HRESULT  __stdcall  CHello::PrintSz(LPWSTR
pwszString) {
printf("%ws\/n", pwszString);
return(ResultFromScode(S_OK));
}


CHello::~CHello(void)
{

// when the object count goes to zero,
stop the server cObjRef--;
if( cObjRef == 0 ) PulseEvent(hEvent);

return;
}

#include <windows.h>
#include <ole2.h>
#include "pshlo.h"
#include "shlo.hxx"
#include "mycls.hxx"

HANDLE hEvent;
int _cdecl main(
int argc,
char * argv[]
) {
ULONG ulRef;
DWORD dwRegistration;
CHelloCF *pCF = new CHelloCF();

hEvent = CreateEvent(NULL, FALSE, FALSE,
NULL);

// Initialize the OLE libraries
CoInitializeEx(NULL,
COINIT_MULTITHREADED);

CoRegisterClassObject(CLSID_CHello, pCF,
AL_SERVER,
REGCLS_MULTIPLEUSE, &dwRegistration);

// wait on an event to stop
WaitForSingleObject(hEvent, INFINITE);

// revoke and release the class object
CoRevokeClassObject(dwRegistration); ulRef
= pCF->Release();

// Tell OLE we are going away.
CoUninitialize();

return(0); }

extern CLSID CLSID_CHello;
extern UUID LIBID_CHelloLib;

CLSID CLSID_CHello = { /*
2573F891-CFEE-101A-9A9F-00AA00342820 */
0x2573F891,
0xCFEE,
0x101A,
{ 0x9A, 0x9F, 0x00, 0xAA, 0x00, 0x34,
0x28, 0x20 }
};

UUID LIBID_CHelloLib = { /*
2573F890-CFEE-101A-9A9F-00AA00342820 */
0x2573F890,
0xCFEE,
0x101A,
{ 0x9A, 0x9F, 0x00, 0xAA, 0x00, 0x34,
0x28, 0x20 }
};

#include <windows.h>
#include <ole2.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "pshlo.h"
#include "shlo.hxx"
#include "clsid.h"

int _cdecl main(
int argc,
char * argv[]
) {
HRESULT  hRslt;
IHello        *pHello;
ULONG  ulCnt;
IMoniker * pmk;
WCHAR  wcsT[_MAX_PATH];
WCHAR  wcsPath[2 * _MAX_PATH];

// get object path
wcsPath[0] = '\/0';
wcsT[0] = '\/0';
if( argc > 1) {
mbstowcs(wcsPath, argv[1], strlen(argv[1])
+ 1); wcsupr(wcsPath);
} else { fprintf(stderr, "Object path must
be specified\/n");
return(1); }

// get print string
if(argc > 2)
mbstowcs(wcsT, argv[2], strlen(argv[2]) +
1);
else
wcscpy(wcsT, L"Hello World");

printf("Linking to object %ws\/n",
wcsPath); printf("Text String
%ws\/n", wcsT);

// Initialize the OLE libraries
hRslt = CoInitializeEx(NULL,
COINIT_MULTITHREADED);

if(SUCCEEDED(hRslt)) {


hRslt = CreateFileMoniker(wcsPath, &pmk);
if(SUCCEEDED(hRslt))
hRslt = BindMoniker(pmk, 0, IID_IHello,
(void **)&pHello);

if(SUCCEEDED(hRslt)) {

// print a string out
pHello->PrintSz(wcsT);

Sleep(2000);
ulCnt = pHello->Release();
}
else
printf("Failure to connect, status: %lx",
hRslt);

// Tell OLE we are going      CoUninitial-
ize(); }

return(0);
}
□
```