kleiner kommunikation
max@kleiner.com

1 Alles über Ressourcen

Zu was dienen eigentlich Ressourcen, mal abgesehen vom Projektmanagement? Diese Technik der Einbindung von Ressourcen und Dateien hat einen weit größeren Stellenwert als gemeinhin angenommen. Kompakte Programme sind im Zeitalter von Mobilität und Mehrsprachigkeit durch einfache Distribution auf CD-ROM oder Internet gefragt, denn eine ausführbare Datei hat fast alles an Bord. Anhand des Spielprojektes „Memory“ und zugehörigem Bildkatalog will ich die Welt der Ressourcen näherbringen und Ihrem Gedächtnis auf die Sprünge helfen.

1.1 Ressourcen oder outsourcen?

Man stelle sich vor, eine Unternehmung produziert aus einem Produktkatalog und den zugehörigen Bildern ein Memory als Gedächtnistraining für die Verkaufsabteilung oder als Geschenk für die werte Kundschaft. Die Installation soll mobil und einfach sein und beim Fotokatalog dürfen keine Änderung oder ein versehentliches Löschen durch externe Daten möglich sein. Hier sind Ressourcen die ideale Technik. Aus der Sicht der Architektur könnte man einwenden, eine vollbepackte EXE sei bezüglich Datenänderung nicht flexibel genug. Ressourcen lassen sich auch extern, z.B. als DLL, speichern und erhalten somit die Form eines UML – Package, die sich verschachteln läßt. Ob man das Package dann in die EXE ein- oder auslagert, ist je nach Strategie im Komponentendiagramm ersichtlich.

Später wird aus dieser Technik eine Dienstleistung mit einem Web Service, indem man seine eigenen Bilder mit definiertem Format in ein Webverzeichnis legt und Minuten später ein persönliches Memory zurückerhält.

Beim Linken einer Anwendung oder einer Bibliothek (nach dem Compilieren des Programms bzw. der DLL) erfolgt die Verarbeitung der Ressourcen-Dateien, die in den benutzten Units und im Programm bzw. in der Bibliothek angegeben sind. Dabei werden die Ressourcen aller Dateien in die ausführbare Datei gelinkt, in meinem Fall Bilder und Tondateien. Während der Ressourcenverarbeitung sucht der Linker nach den zugehörigen res-Dateien; diese Suche erfolgt in dem Verzeichnis, in dem auch das Modul mit der Direktive $R liegt.

Doch wie erstellt man ein Ressourcenskript, die der Compiler *brcc32.exe* verarbeiten kann. Der Compiler befindet sich übrigens im Bin-Verzeichnis von Delphi. Im Grunde genügt als Anfang ein einfacher Texteditor, der mit folgenden Einträgen (name, restype, filename) gefüllt wird:

```
bmp0 Bitmap "gurtenturm.bmp"
bmp1 Bitmap "genfkunst.bmp"
bmp2 Bitmap "sepp.bmp"
wav1 WAVE "tick.wav"
wav2 WAVE "maxmor.wav"
wav3 WAVE "crowd.wav"
```

Dieses im Editor erstellte Skript nenne ich *memoryres.rc* oder allgemein *.RC. Die Namen der Ressourcen sind frei wählbar (bmp0, wav2), müssen aber mit dem Aufruf aus der EXE identisch sein. Die Art der Ressource wird als definierter Typ vom entsprechenden API erkannt. Die Dateinamen (*gurtenturm.bmp* etc.) werden in einem ersten Schritt in eine res-Datei gepackt und später in die EXE eingebunden. Ich kompiliere zuerst das Ressourcen Skript aus der Shell:

```
D:\DELPHI\PEntwickler18 > BRCC32.exe memoryres.rc
```

Bei der Compilierung wird die Datei *memoryres.res* generiert. Anschließend bindet der Linker diese res-Datei in die gewünschte EXE ein. Die Direktive $R legt den Namen der res-Datei fest, die der Linker in eine Anwendung oder eine Bibliothek einbindet. Wenn der Dateiname ein Leerzeichen enthält, schließen Sie ihn in halbe Anführungszeichen ein: `{$R 'my memoryres.res'}`.

Aufgepaßt: Die von Delphi und dem IDE-Wizard standardmäßig generierte Ressourcen - Datei ist erforderlich, um das Projekt ohne Fehler öffnen zu können. Die Projektdatei bezieht sich auf diese res-Datei, welche die Versionsinformationen und das Symbol der Anwendung enthält. In diesem Fall bewirkt die Verwendung von {$R *.res} in der Projektdatei, daß die zugehörige res-Datei mit der Anwendung verknüpft wird. Es empfiehlt sich, diese Datei nicht zu verändern. Selbstverständlich kann man auch eine zweite res-Datei, als Compiler - Direktive durch den Linker einbinden lassen:

```
{$R *.RES}   //IDE Wizard
{$R memoryres.RES}
```

Diese Direktiven lassen sich auch verschachteln:

```
{$IFDEF MEMSOUND}
  {$R memsound.RES}
{$ELSE}
  {$R memory4.RES}
{$ENDIF MEMSOUND}
```

Ein Ziel sei nun das automatische Generieren einer Ressourcen - Datei aus einem Verzeichnis von Fotos, Klängen und dergleichen, so daß ein manuelles Erstellen des Skriptes im Editor entfällt. Mein Bildkatalog besteht aus rund 35 Bitmaps. Folgender Code des Skriptgenerator befindet sich in der Unit *Stats.pas* auf der CD-ROM:

```
procedure TStatForm.scriptresourceFile2(restype: string);
var
  f: textfile;
  ResFile: shortstring;
  resstr: string;
  s: array[0..2048] of Char;
  i, filecount: byte;
  myResList: TStringList;
begin
  myresList:= TStringList.Create;
  filecount:= getfilelist(myResList);
  if filecount > totalPictures then
     filecount:= totalPictures;
  for i:= 0 to filecount - 1 do begin
    resstr:= Format('%s%d %s %s%s%s',
               ['bmp', i, restype,'"',myReslist.strings[i],'"']);
    StrCat(s, pchar(resstr));
    StrCat(s,#13#10);
  end;
  ResFile:= 'membmp.rc';
  AssignFile(f, ResFile);
  Rewrite(f);
  write(f,s);
  closefile(f);
  myResList.Free;
  compileResfile(ResFile);
end;
```

Als erstes fülle ich eine Stringliste mit den Dateinamen der Bilder ab, so daß innerhalb der Iteration das Skript als File mit der Format Funktion erstellt wird. Das anschließende Compilieren läßt sich mit dem übergebenen Dateinamen *membmp.rc* weiter automatisieren:

```
procedure TStatForm.compileResfile(vfile: string);
var i, iCE: integer;
begin
  {$IFDEF MSWINDOWS}
  iCE:= shellapi.shellExecute(0, NIL, pChar('BRCC32.exe'),
                         pChar(vfile), NIL, 0);
  i:= 0;
  repeat
    inc(i);
    sleep(600);
    application.ProcessMessages;
  until i >= 10;
  if iCE <= 32 then showMessage('compError Nr. '+ inttoStr(iCE));
  {$ENDIF}
end;
```

Mit der einfachen Warteschlaufe erzwinge ich eine Synchronisation mit `shellExecute`. Besser wäre hierzu `WaitForSingleObject`() zu gebrauchen oder die Klasse `TCustomFileRun` einzusetzen. Beim Aufruf der Methode `ExecuteTarget` übergibt `TCustomFileRun` diverse Eigenschaften an die Windows API-Funktion `ShellExecute`.

Nach dem Linken der res-Datei in die EXE ist es nun an der Zeit, die Ressourcen aufzurufen. Die eingebetteten Bilder möchte ich nun innerhalb von Memory wie ein Fotokatalog abrufen. Hierzu dienen diverse Load Methoden, die ein Handle benötigen:

```
Image:= PChar('BMP' + IntToStr(Random(TotalPictures)+1));
ABmp.Handle:= LoadBitmap(HInstance, Image);
PaintBox1.Canvas.Draw(0,0,ABmp);
```

Die Methode `LoadFromResourceName` lädt auch eine Bitmap-Ressource in das Bitmap-Objekt. Verwenden Sie diese Routine anstelle von `LoadBitmap`, denn `LoadBitmap` unterstützt keine Bilder mit nur 256 Farben. Der Parameter `HInstance` bezeichnet das Handle des Moduls, das die Ressource enthält. Und wenn sich sogar ein Audio - Memory ergeben soll, ist `PlaySound` nicht fern. Memory ist auch als Klangspiel konzipiert, indem man gleiche Töne finden muss:

```
for i:= 0 to TotalSounds -1  do begin
     mSound:= PChar('WAV' + IntToStr(i));
     PlaySound(pchar(msound),HInstance, snd_Sync
                         or snd_Memory or snd_Resource);
```

Eine weitere Form sind die sogenannten String-Ressourcen, die vor allem bei der Internationalisierung zum Einsatz kommen. String-Ressourcen werden nicht in die Formulardatei eingebunden. Sie lassen sich isolieren, indem man Mithilfe des reservierten Worts `resourcestring` Konstanten deklariert.

```
Resourcestring
  RSEmailAdr = 'mailto:max@kleiner.com?subject=[ressourcen]';
  CreateError = 'Cannot create file %s';
```

Das Auslagern von Ressourcen vereinfacht den Übersetzungsprozess. Eine fortgeschrittene Stufe der Auslagerung besteht im Erstellen einer DLL. Eine Ressourcen-DLL ist eine DLL, die alle Ressourcen –

und nur diese – eines Programms enthält. Eine Anwendung unterstützt dann eine Vielzahl von Übersetzungen, indem man einfach das Ressourcenmodul austauscht.

Mit dem Ressourcen-DLL-Experten ist ein Ressourcenmodul erstellbar. Der Experte erstellt eine rc-Datei, in der die Stringtabellen der verwendeten rc-Dateien und die oben erwähnten String-Ressourcen des Projekts enthalten sind. Die res-Datei wird dann aus der neuen rc-Datei compiliert.

Delphi erstellt ja automatisch eine .dfm-Datei (in CLX .xfm), welche die Ressourcen von Menüs, Dialogfeldern, Cursor und Bitmap-Grafiken enthält. Außer diesen offensichtlichen Bestandteilen der Benutzeroberfläche müssen Sie deshalb mit dem Experten alle eigenen Strings isolieren, die man dem Anwender präsentiert, beispielsweise Fehlermeldungen und Hinweise. Diese String-Ressourcen werden deshalb nicht in die Formulardatei eingebunden.

Beim Laden ermittelt die EXE das Gebietsschema des Rechners, auf dem man sie ausgeführt. Wenn sie ein Ressourcenmodul findet, das denselben Namen hat wie die EXE-, DLL- oder DPL-Dateien, überprüft sie die Dateinamenerweiterung dieser DLL. Falls die Dateinamenerweiterung eines Ressourcenmoduls mit dem Gebietsschemacode des Rechners übereinstimmt, verwendet die Anwendung anstelle der Ressourcen in den EXE-, DLL- oder DPL-Dateien die Ressourcen aus dem Modul.

Früher wurden auch Typenbibliotheken für Automatisierungsanwendungen in getrennten Dateien mit der Namenserweiterung .TLB als Binärdatei gespeichert. Inzwischen werden bei Automatisierungsanwendungen die Typenbibliotheken normalerweise direkt in die .OCX oder .EXE kompiliert. Das Betriebssystem erwartet die Typenbibliothek als erste Ressource in der ausführbaren Datei [DE01].

```
tlb1 typelib mylib1.tlb
tlb2 typelib mylib2.tlb
```

Eine DLL die nur Ressourcen enthält und diese dann zur Laufzeit benötigt, lässt sich auch ohne Experten einrichten, Die Idee ist, einen Wrapper um die res-Datei zu bauen:

```
library MemoryModule;

uses
  SysUtils;

{$R memoryres.RES}

begin
end.
```

Im laufenden Einsatz gilt es dann, die DLL zu laden und den Zugriff zu ermöglichen:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  h: THandle;
  Icon: THandle;
begin
  h:= LoadLibrary('MEMORYMODULE.DLL');
  if h <= BadDllLoad then
    ShowMessage('Bad Dll Load')
  else begin
    Icon:= LoadIcon(h, 'ICON_1');
    DrawIcon(Form1.Canvas.Handle, 10, 10, Icon);
    FreeLibrary(h);
  end;
end;
```

1.2 Benutzerdefinierte Ressourcen

Der Typ der Ressource wird mit dem Parameter ResType identifiziert. Anwendungen können eigene Typen definieren und per Name in der RC-Datei referenzieren. Außerdem gibt es eine Reihe vordefinierter Ressourcetypen (die den Win-Ressourcetypen aus der API entsprechen).

Mit dem ResType `RT_RCDATA` lassen sich beliebige Rohdaten strukturiert oder binär als Ressource speichern. Eine HTML-Datei kann bspw. aus einer EXE in einem `TWebBrowser` Objekt angezeigt werden. Auch Delphi selbst macht regen Gebrauch von RCData, wie in Abb. 2 offensichtlich zu sehen ist. Die globale Funktion `ReadComponentRes` verwendet `TResourceStream` für den Zugriff auf die kompilierten Ressourcen, welche die EXE benötigt.

Jedes in Delphi in der IDE erstellte Formular wird beim Programmstart automatisch im Speicher erzeugt, indem der entsprechende Code in die Funktion aufgenommen wird, die den Haupteinsprungspunkt der Anwendungsfunktion bildet. Diese dfm-Datei ist zur Laufzeit als Ressource unter dem Typ RCData zu finden. Daher die bekannte Compiler Direktive {$R *.DFM} !

Nicht alle Formulare müssen sich jedoch während der Ausführungszeit der Anwendung im Speicher befinden. Bspw. lassen sich Dialogfelder auch dynamisch aus der Ressource erstellen.

Im Editor ist ein RCData schnell eingerichtet:

```
mystring RCDATA PRELOAD
BEGIN
  "Now it’s time for Patterns."
  255001001,
  "Null-terminated string\0",
END
```

Man kann eine beliebige „gültige“ Datei als RCData Ressource in eine EXE einbinden. Auch für das Einlesen ist mit der Klasse `TResourceStream` gesorgt. Das folgende Beispiel zeigt, wie ich einen RTF-Text aus einer Ressource lade und in einem `TRichEdit` anzeigen lasse.

Nach dem Erstellen der Datei *rtftext*.rc mit folgendem Inhalt:

```
MaxDoc RCDATA rtftext.rtf
```

erzeugt der Compiler die binäre Datei `rtftext.res`.

Diese Datei binde ich nun in die Unit ein, was zu folgendem Code führt:

```
implementation

{$R *.dfm}
{$R rtftext.res}

procedure TForm1.Button1Click(Sender: TObject);
var
  rs: TResourceStream;
begin
  rs:= TResourceStream.Create(hinstance, 'MAXDOC', RT_RCDATA);
  try
    Richedit1.PlainText:= False;
    rs.Position:= 0;
    Richedit1.Lines.LoadFromStream(rs);
  finally
    rs.Free;
  end;
end;

Variante
  RS := TResourceStream.CreateFromID(HInstance,
    100, RT_RCDATA);
```

Wenn ein zusätzliches Kopieren in ein File erwünscht wird (da ein Schreiben in die EXE nicht erlaubt ist) oder das Umbetten in einen anderen Stream möglich sein muß, sieht der Ansatz so aus:

```
  AStream.CopyFrom(RS, RS.Size);
        FileStream.CopyFrom(RS, 0);
```

Ähnlich sieht das Laden eines JPEG Formates aus:

```
function GetResourceAsJpeg(const resname: string): TJPEGImage;
var
  Stream: TResourceStream;
begin
  Stream:= TResourceStream.Create(hInstance, ResName, 'JPEG');
  try
    Result:= TJPEGImage.Create;
    Result.LoadFromStream(Stream);
  finally
    Stream.Free;
  end;
end;
```

```
   Jpg:= GetResourceAsJpeg('soap_jpg');
   Image2.Picture.Bitmap.Assign(Jpg);
   Jpg.Free;
```

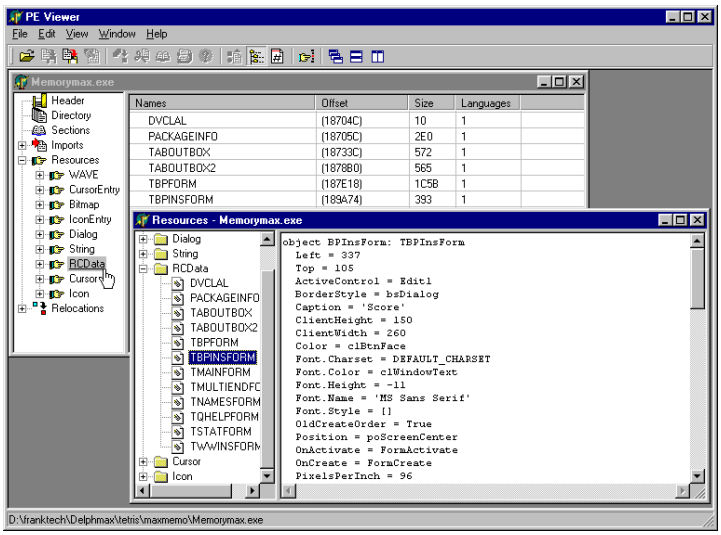

Abb. 2: Der PEViewer steht bereit zum Ressourcenjagen

1.3 Der Compiler und andere Tools

Beim Einsatz von Ressourcen ist man auf einige Tools angewiesen. Essentiell ist sicher der bereits bekannte Ressourcen Compiler BRCC32.EXE, der in der Version 5.4 vorliegt. Wenn der Ressourcen Compiler mit dem Client mitgeht, benötigt man zusätzlich die Datei *rw32core.dll* im Installationsumfang.

Das eigentliche Bearbeiten oder Erzeugen von Ressourcen kann mit dem guten alten Resource Workshop erfolgen, der mit Delphi 5 Professional und Enterprise daherkommt oder im Borland RAD Pack enthalten ist. Der Resource Workshop 5.0 kompiliert und dekompiliert 16 und 32 Bit Ressourcen aus rc, res, exe, dll, drv, vbx, cpl, ico, bmp, rle, dlg, fnt, und cur Dateien. Diejenigen Zeitgenossen aus der Borland Pascal Ära unter uns agierten sicher auch als Ressourcenjäger, um die tollsten Ikonen zu sammeln ;).

Delphi besitzt unter Tools / Bildeditor auch einen eigenen Ressourcen Editor, der aber von den Möglichkeiten den Resource Workshop nicht ersetzen kann. Dennoch hat der Bildeditor ein Einsatzgebiet im Komponentenbau. Jede Komponente benötigt ja eine Bitmap-Grafik für die Anzeige in der Komponentenpalette. Wenn man kein eigenes Bitmap angibt, wird von der IDE die Standardgrafik verwendet. Die Paletten-Bitmaps werden nur zur Entwurfszeit benötigt und dürfen

daher nicht in die Unit der Komponente kompiliert werden. Sie werden in einer res-Datei mit dem Namen der Unit und der Erweiterung *.dcr (Dynamik Component Ressource) gespeichert.

Tip: Sofern Sie für das Symbol eine .res-Datei anstelle einer dcr-Datei benutzen, müssen Sie aber eine Referenz auf die Quelle der Komponente hinzufügen, um die Ressource zu binden.

Wenn Sie übrigens eine Grafik in eine Image-Komponente laden, können Sie veranlassen, dass sich die Größe der Grafik automatisch an die Komponente anpaßt wird, indem man die Eigenschaft `Stretch` des Steuerelements auf true setzt.

Ein weiteres Tool befindet sich als Source im Delphi Verzeichnis *\Demos\ResXPlor*, welches Ressourcen aus den kompilierten Dateien holt, wobei die Luxusvariante aus der Jedi Code Library stammt, nämlich der *PEViewer.exe* (siehe Abb. 2), der auch auf der CD-ROM zu finden ist. [JVCL]

Von MS stammt der *resourcecompiler.exe*, der einzig und allein Stringtabellen als UTF8-encoded Text erzeugt, nebst dem Vollcompiler *rc.exe* aus dem SDK.

1.4 Auch unter Kylix und Delphi.NET

Am Schluß meines Streifzuges sollen die langsam aber stetig zunehmenden Linux - Entwickler auf ihre Kosten kommen. Ob schnell aber stetig auf Delphi.NET umgestiegen wird, lässt sich noch nicht beurteilen. In Kylix existiert das Tool Resbind, das aus einer EXE Ressourcen extrahieren kann. Mit dem Aufruf

```
Resbind –r myprogram.res myprogram
```

kommt man auch ohne Workshop zu den Ressourcen. Es ist auch möglich Win-Ressourcen in ein Shared Object (Linux DLL) mit resbind zu konvertieren. Wie aber bindet man die Ressourcen?

Eine Res-Datei für Kylix kann man mit dem Ressourcen Compiler unter Windows generieren. Kylix unterstützt aber wenig vordefinierte Typen wie ein BitMap. Deshalb muß man alle Ressourcen, die keinen vordefinierten ResType haben, als RCData definieren. Im Folgenden schreibe ich eine Ressource in eine Datei:

```
uses
  SysUtils, Types, Classes, Variants, QGraphics, QControls,
  QForms, QDialogs, QStdCtrls;

implementation

{$R *.xfm}
{$R userdefined.res}

procedure TForm1.Button1Click(Sender: TObject);
var
  stm: TResourceStream;
begin
  stm:= TResourceStream.Create(HInstance, 'MYRES1', RT_RCDATA);
  with TFileStream.Create('test.txt', fmCreate) do begin
    CopyFrom(stm, stm.Size);
    Free;
  end;
end;
```

Der Compiler reserviert standardmäßig 1 MB Adressraum zusätzlich zu dem von der Anwendung beim Linken verwendeten Speicher für Ressourcen unter Kylix.

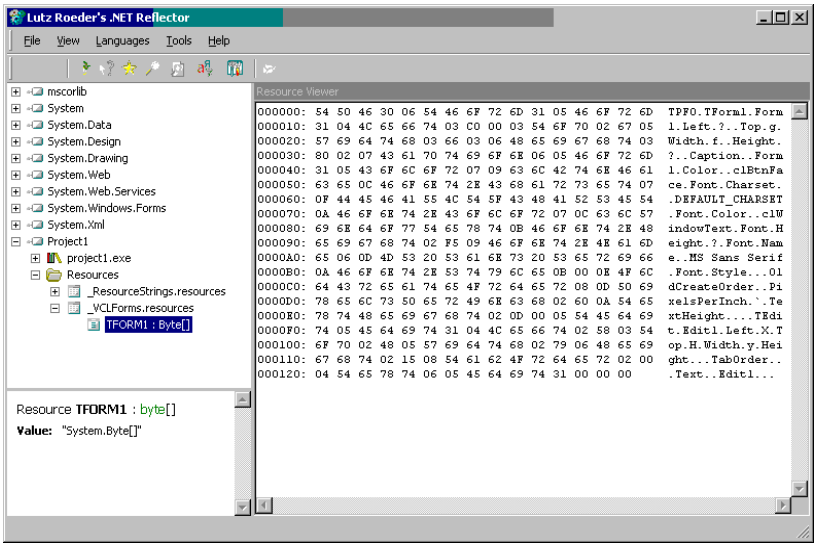

Abb. 3: Auch die VCL.NET bindet die Ressourcen

Unter Delphi.NET gab es in der ersten Phase das Tool *dfm2pas*, das die `InitializeControls`() Methode in einer pas-Datei erzeugte, so daß wie bei den WinForms alles in einer Datei liegt. Die strikte Trennung von Form und Funktion hat aber aus der Sicht der Implementierung und Versionierung mehr Vorteile. Im offiziell erschienenen Delphi for .NET herrschen als VCL.NET wieder die bewährten Gesetze, wie Abb. 3 zeigt.

Nur sind es keine Win32 Ressourcen mehr sondern .NET Ressourcen die bspw. mit einem .NET Reflector aus der EXE gezogen werden. Die Bedeutung ist aber gleichgeblieben, wie Dmitry bemerkt. Auch die Stringtabellen bezüglich Sprachen haben überlebt. Somit sind auch in der .NET Welt lesbare Ressourcen vorhanden, die Tools dazu haben sich massiv geändert.

Fazit: Ressourcen haben nicht nur im Projektmanagement einen hohen Stellenwert. Mithilfe der Projektverwaltung in Delphi haben auch Sie eine Übersicht über die Projekte einer Projektgruppe und über die in der Projektdatei enthaltenen Formular-, Unit-, Ressourcen-, Objekt- und Bibliotheksdateien.

Literatur & Links:

[DE01], Das Plus beim Export, Der Entwickler 4.2001

[JVCL], http://delphi-jedi.org/Jedi:CODELIBJCL

http://www.delphi3000.com – Umfangreiche Codebank

Buch „Patterns konkret“

Max Kleiner

Memory Beispiel und Tools auf der CD-ROM