Inside Components

Eine besondere Form von shared Libraries sind die „Components", welche Apple mit QuickTime eingeführt hat. Inzwischen sind die Components jedoch ein fester Bestandteil des MacOS geworden und werden an vielen Stellen eingesetzt. Selbst in eigenen Programmen kann man die Components gewinnbringend einsetzen.

Was sind Components nun genau? Eine Component ist eine Sammlung von Routinen, welche man wie eigene Routinen nutzen kann. Man muß die Component lediglich vorher registrieren. Dabei spielt es keine Rolle, ob diese Component nur im eigenen Programm eingesetzt wird oder als Systemerweiterung im Systemordner existiert und so von mehreren Programmen genutzt werden kann.

Zu jeder Component gehört ein Name, ein Info-String und ein Icon. Dies ermöglicht es dem User, eine Liste von Components – z.B. bei Plug-ins – ordentlich zu präsentieren. Als weiteres Highlight gibt es eine automatische Versionsverwaltung, die sicherstellt, daß immer die neuste Component genutzt wird. Dabei besteht sogar die Möglichkeit, zur Laufzeit eine neue Version einer Component zu installieren! Components können natürlich auch andere Components aufrufen um z.B. die Funktionalität einer bestehenden Component zu verändern.

Wofür werden Components denn nun im MacOS genutzt? Als erstes ist natürlich QuickTime zu nennen: Nahezu die gesamte Funktionalität von QuickTime besteht aus Components. So kann man z.B. neue Movie-Import bzw. Exportformate, aber auch neue Packer installieren. Selbst einen neuen Movie-Controller kann man so definieren! Videodigitizer sind selbstverständlich ebenfalls als Components implementiert. Previews für diverse Fileformate im Fileselector: Components. Der Sound-Manager nutzt natürlich auch Components z.B. für Formatwandlungen, Audio-Packer und die Sound-Ein- bzw. Ausgabe.

Ein sehr bekanntes Component, welches nicht von Apple stammt, ist Internet Config. Es dient zur Konfiguration von allen Internet-Programmen über eine streng definierte Oberfläche. Der Vorteil wird schnell klar: Fast alle Internet-Programme brauchen die gleichen Einstellungen. Es bietet sich also an, diese den User nur einmal tätigen zu lassen und sie dann von allen Programmen gemeinsam zu nutzen.

Wer einmal wissen will, welche Components (ursprünglich „Things" genannt) im Rechner installiert sind, gebe im MacsBug – den jeder Programmierer wohl eh installiert hat – einmal ‚thing' ein. Bei mir sind es z.B. über 100! Viele Components liegen inzwischen sowohl im 68k, wie auch als PowerPC-Code vor. Was sich insbesondere bei QuickTime und dem Sound Manager sehr bemerkbar macht.

Die ‚thng'-Resource

Wie sieht nun so eine Component aus? Eine Component besteht zunächst einmal aus einer *thng*-Resource. Zuerst kommt hier der Description Record. Er umfaßt einen Typ und Subtype, sowie einen Manufacturer Code. Ein Typ von *sdev* entspricht dabei dem *kSoundOutputDeviceType*, beschreibt

also einen Ausgabetreiber für den Sound Manager. Der Subtype *awac* beschreibt den speziellen Treiber, hier die Hardware eines PowerMac 6100. Der Manufacturer *appl* steht – wie man unschwer erraten wird – für Apple Computer... Will man eine eigenständige Component schreiben, so wird man einen neuen Typ definieren, ansonsten wird lediglich ein neuer Subtype definiert.

Ferner stehen in der *thng*-Resource Verweise auf den Namen der Component, einen Info-String, ein Icon, bzw. eine Icon-Familie und natürlich auf die Resourcen mit dem Programm-Code.

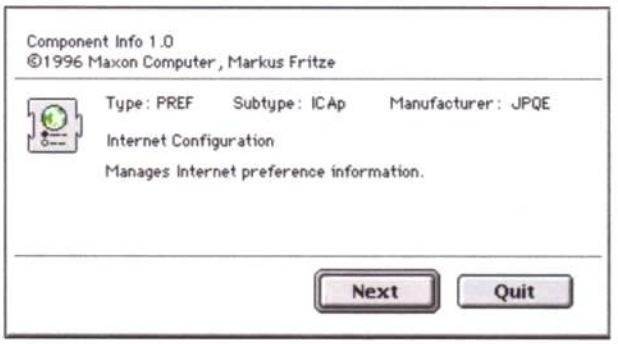
Zu guter Letzt gibt es in der Resource noch die Versionsnummer der Component und einige Flags.

Der Component Manager

Das MacOS stellt natürlich einige Funktionen zur Verwaltung von Components bereit:

FindNextComponent() ermöglicht es einem, eine oder mehrere Components zu suchen. Dabei kann man nach dem Typ, Subtype, Manufacturer und einer Bitmaske mit Flags suchen. So ist es z.B. möglich, ganz allgemein nach allen QuickTime-Videoeingabetreibern zu suchen, oder aber sehr speziell nach der Component eines bestimmten Typs eines bestimmten Manufacturer.

CountComponents() ermittelt die Anzahl der auf die Suchmaske passenden Components.



Unser Beispielprogramm „Component Info" erlaubt, sich alle Components im System einmal anzusehen.

Um mit einer Component zu kommunizieren, muß man sie vorher öffnen. Dies geschied entweder mit *OpenDefaultComponent()*, der man einen Typ und – optional – einen Subtype übergibt. Oder aber indem man zuerst eine bestimmte Component mit *FindNextComponent()* sucht und diese dann mit *OpenComponent()* öffnet. Nutzt man die Component von seinem Programm aus nicht mehr, so muß man sie mit *CloseComponent()* wieder schließen.

Zusätzlich kann man mit *GetComponentInfo()*, *GetComponentIconSuite()* und *GetComponentVersion()* die Einträge in der *thng*-Resource abfragen.

Bevor eine Component gesucht werden kann, muß sie beim System registriert sein. Dabei gibt es zwei Möglichkeiten: Man legt die Component in einem File mit dem Filetype *thng* im Systemerweiterungen-Ordner ab. Beim Neustart werden diese dann automatisch registriert. Dies ist wohl die übliche Methode für Sound Treiber und andere globale Systemcomponents. Einige Plug-ins wird man wahrscheinlich eher außerhalb des Systemordners lagern wollen. Diese muß man dann mit den Funktionen *RegisterComponent()*, *RegisterComponentResource()* oder *RegisterComponentResourceFile()* selbst registrieren. Dabei kann man u.a. auch noch angeben, ob diese Components nur für das eigene Programm verfügbar sein sollen oder ob auch noch andere Programme auf die Components zugreifen dürfen. Mit *UnregisterComponent()* melden wir eine Component wieder ab.

Mehr Systemfunktionen braucht ein normales Anwenderprogramm üblicherweise nicht von einer Component zu kennen.

Da alle Funktionsaufrufe einer Component über einen Dispatcher laufen - eine Component hat ja nur einen Einsprungpunkt – werden die Funktionen über einen Requestcode unterschieden. Mit der Funktion *ComponentFunctionImplemented()* kann man nun überprüfen, ob die Component einen bestimmten Requestcode unterstützt. Jede Component muß dabei die internen Requestcodes für Open, Close, CanDo und Version erfüllen. Optional

```
 1: /***
 2:  *  Component Info.cp
 3:  *  (c)1996 Maxon Computer
 4:  *  Autor: Markus Fritze
 5:  ***/
 6:
 7: #define THE_DIALOG      128
 8: enum {
 9:   itemNext = 1,
10:   itemQuit,
11:   itemName,
12:   itemTyp,
13:   itemSubTyp,
14:   itemManu,
15:   itemInfo,
16:   itemIcon
17: };
18:
19: Handle        gIconSuite;
20:
21: /***
22:  *  neuen Text in einem Static-Text-Item
    setzen
23:  ***/
24: static void SetItemText(
25:       DialogRef iDialog, short iItem,
26:       StringPtr iTextStr)
27: {
28:   short   theKind;
29:   Handle  theHandle;
30:   Rect    theRect;
31:
32:   GetDialogItem(iDialog, iItem, &theKind,
33:                 &theHandle, &theRect);
34:   SetDialogItemText(theHandle, iTextStr);
35: }
36:
37: /***
38:  *  4 Byte Typ in ein Static-Text-Item
39:  *  setzen
40:  ***/
41: static void SetItemTextType(
42:       DialogRef iDialog, short iItem,
43:       OSType iType)
44: {
45:   Str15 theStr;
46:   BlockMoveData(&iType, theStr + 1, 4);
47:   theStr[0] = 4;
```

→

sind Register, Unregister und Target. Näheres zu den einzelnen Requestcodes später. Zusätzlich zu diesen Codes des Component Managers gibt es üblicherweise noch weitere für die eigentlichen Funktionen der Component. Handelt es sich dabei um einen von Apple definierten Componenttyp, so sind diese ebenfalls zu implementieren, damit sich die Component wie eine von Apple verhält. Natürlich kann man als Component trotzdem weitere eigene Funktionen zur Verfügung stellen.

Der Requestcode und der eigentliche Aufruf der Component wird üblicherweise im Headerfile zu dieser Component versteckt. Ein Nutzer einer Component braucht sich damit also nicht zu belasten und kann einfach die Funktionen der Component direkt aufrufen – wie Funktionen innerhalb seines Programms. Dafür ist das Erstellen der Headerfiles für den Entwickler von Components etwas aufwendiger. Da es aber stets nach dem gleichen Schema abläuft, kann man bei anderen Components (schließlich hat Apple ja bereits diverse mitgeliefert) einfach abschreiben.

Das Beispielprogramm

So, zum Abschluß unseres ersten Teils zum Thema Components ein kleines Beispielprogramm, welches es einem erlaubt, sich alle Components im System einmal anzusehen. Interessant dabei sind wohl vor allen Dingen die teilweise sehr phantasievolle Gestaltung der Icons... Weil das Programm die Iconsuite der Component abfragt, muß die Version des Component Manager mindestens 3 sein (z.B. QuickTime 1.6 oder später installiert sein), alternativ könnte man den Aufruf entfernen und das mit *GetComponentInfo()* ermittelte Icon zeichnen – das gibt es dann allerdings nur in s/w. Wie immer ist das ausführbare Programm nebst Source-Code im FirstClass-Server der MacOPEN (06196-484457) verfügbar. Im nächsten Teil schreibe ich dann etwas zur Entwicklung einer eigenen Component.

MFR

Literatur:

QuickTime Components, Addison-Wesley, ISBN 0-201-62202-5, $34.95

Sound, Addison-Wesley, ISBN 0-201-62272-6, $29.95 (Sound und Audio Components)

More Macintosh Toolbox, Addison-Wesley, ISBN 0-201-63299-3, $34.95 (ein spezielles Kapitel über Components, Pflichtlektüre)

TechNote 1004: On QuickTime Component Manager 3.0 & PowerPC Native Components, auf <URL: http://dev.info.apple.com***> oder auf der Reference Library CD-ROM des Entwicklersupports zu finden.***

A fragment of your imagination, Addison-Wesley, ISBN 0-201-48358-0, $39.95 (ausführliche Beispiele zu allen Arten von Code-Resourcen, sehr zu empfehlen!)


```
48:   SetItemText(iDialog, iItem, theStr);
49: }
50:
51: /***
52:  *  Handle in ein Static-Text-Item setzen
53:  ***/
54: static void SetItemTextH(
55:       DialogRef iDialog, short iItem,
56:       Handle iTextH)
57: {
58:   HLock(iTextH);
59:   if(GetHandleSize(iTextH) == 0)
60:     SetItemText(iDialog, iItem, "\p???");
61:   else
62:     SetItemText(iDialog, iItem,
63:                 StringPtr(*iTextH));
64:   HUnlock(iTextH);
65: }
66:
67: /***
68:  *  Zeichensatz in der Dialogbox setzen
69:  ***/
70: static void SetWindowFont(
71:         DialogRef iDialog, short iFontNum,
72:         short iFontSize, Style iFontStyle,
73:         short iFontMode)
74: {
75:   GrafPtr   theSavePort;
76:   GetPort(&theSavePort);
77:   SetPort(iDialog);
78:
79:   TextFont(iFontNum);
80:   TextSize(iFontSize);
81:   TextFace(iFontStyle);
82:   TextMode(iFontMode);
83:
84:   FontInfo  fInfo;
85:   GetFontInfo(&fInfo);
86:
87: (*((DialogPeek)iDialog)->textH)-
    >fontAscent = fInfo.ascent;
88:
89: (*((DialogPeek)iDialog)->textH)-
    >lineHeight = fInfo.ascent + fInfo.descent
    + fInfo.leading;
90:
91: (*((DialogPeek)iDialog)->textH)->txFont =
    iFontNum;
92:
93: (*((DialogPeek)iDialog)->textH)->txFace =
    iFontStyle;
94:
95: (*((DialogPeek)iDialog)->textH)->txMode =
    iFontMode;
96:
97: (*((DialogPeek)iDialog)->textH)->txSize =
    iFontSize;
98:
99:   SetPort(theSavePort);
100: }
101:
102: /***
103:  *  Icon einer Component zeichnen
104:  *
105:  *  Ein Useritem in der Dialogbox
106:  ***/
107: static pascal void DrawComponentIcon(
108:       DialogRef iDialog, short iItem)
109: {
110:   GrafPtr theSavePort;
111:
112:   GetPort(&theSavePort);
113:   SetPort(iDialog);
114:
115:   // Größe und Position vom Icon holen
116:   short   theKind;
117:   Handle  theHandle;
118:   Rect    theRect;
119:   GetDialogItem(iDialog, iItem, &theKind,
120:                 &theHandle, &theRect);
121:
122:   EraseRect(&theRect);
123:   if(gIconSuite &&
124:     GetHandleSize(gIconSuite) > 0)
```

→

```
125:     PlotIconSuite(&theRect, atNone,
126:                   ttNone, gIconSuite);
127:
128:   SetPort(theSavePort);
129: }
130:
131: /***
132:  *  ...
133:  ***/
134: void main(void)
135: {
136:   // initialize the Mac
137:   InitGraf(&qd.thePort);
138:   InitFonts();
139:   FlushEvents(everyEvent, 0);
140:   InitWindows();
141:   InitMenus();
142:   TEInit();
143:   InitDialogs(nil);
144:   InitCursor();
145:   DrawMenuBar();
146:
147:    long theResult;
148:    if(Gestalt(gestaltComponentMgr,
       &theResult) != noErr
149:       || theResult < 3) {
150:       SysBeep(1);
151:       SysBeep(1);
152:       SysBeep(1);
153:       ExitToShell();
154:    }
155:
156:   // Dialogbox holen und alles setzen
157:   DialogRef theDialog =
158:           GetNewDialog(THE_DIALOG, nil,
159:                        (WindowPtr)-1L);
160:   if(theDialog == nil) ExitToShell();
161:   SetDialogDefaultItem(theDialog,
162:                        itemNext);
163:   SetDialogCancelItem(theDialog,
164:                       itemQuit);
165:
166:   // Dialogbox auf Geneva 9 Punkt
167:   // umschalten
168:   SetPort(theDialog);
169:   SetWindowFont(theDialog, geneva, 9, 0,
170:                 srcCopy);
171:
172:   // Useritem setzen
173:   short   theKind;
174:   Handle  theHandle;
175:   Rect    theRect;
176:   GetDialogItem(theDialog, itemIcon,
177:     &theKind, &theHandle, &theRect);
178:   UserItemUPP drawComponentIconUPP =
179:     NewUserItemProc(DrawComponentIcon);
180:   SetDialogItem(theDialog, itemIcon,
181:     theKind, (Handle)drawComponentIconUPP,
182:     &theRect);
183:
184:   ShowWindow(theDialog);
185:
186:   // unsere Suchmaske für Components
187:   ComponentDescription  theDesc;
188:
189:   theDesc.componentType =
190:               kAnyComponentType;
191:   theDesc.componentSubType =
192:               kAnyComponentSubType;
193:   theDesc.componentManufacturer =
194:               kAnyComponentManufacturer;
195:   theDesc.componentFlags =
196:               0L;
197:   theDesc.componentFlagsMask =
198:               kAnyComponentFlagsMask;
199:
200:   short     itemHit;
201:   Component theComponent = nil;
202:   do {
203:     // alte Icons freigeben
204:     if(gIconSuite) {
205:       DisposeIconSuite(gIconSuite, true);
206:       gIconSuite = nil;
207:     }
208:
209:     // zur nächsten Component
210:     do {
211:       theComponent =
212:        FindNextComponent(theComponent,
213:                          &theDesc);
214:     } while(!theComponent);
215:     // while, damit wir am Ende der
216:     // Liste wieder zum Anfang springen
217:
218:     // Infos über die Component holen
219:     ComponentDescription  cd;
220:     Handle cn, ci;
221:     cn = NewHandle(0L);
222:     ci = NewHandle(0L);
223:     OSErr err =
224:       GetComponentInfo(theComponent, &cd,
225:                        cn, ci, nil);
226:     if(err == noErr) {
227:       SetItemTextType(theDialog, itemTyp,
228:         cd.componentType);
229:       SetItemTextType(theDialog,
230:         itemSubTyp, cd.componentSubType);
231:       SetItemTextType(theDialog, itemManu,
232:         cd.componentManufacturer);
233:       SetItemTextH(theDialog, itemName,
234:         cn);
235:       SetItemTextH(theDialog, itemInfo,
236:         ci);
237:
238:       // Icon-Suite der Component (falls
239:       // vorhanden) holen
240:       GetComponentIconSuite(theComponent,
241:                             &gIconSuite);
242:     }
243:     DisposeHandle(cn);
244:     DisposeHandle(ci);
245:
246:     // Icon neu zeichnen
247:     CallUserItemProc(drawComponentIconUPP,
248:                      theDialog, itemIcon);
249:
250:     ModalDialog(nil, &itemHit);
251:   } while(itemHit != itemQuit);
252: }
```

```
1: /***
2:  *  Component Info.r
3:  *  (c)1996 Maxon Computer,
4:  *  Autor: Markus Fritze
5:  ***/
6:
7: #include "Types.r"
8:
9: resource 'DLOG' (128) {
10:   {118, 142, 301, 487},
11:   dBoxProc, invisible,
12:   noGoAway, 0x0, 128, ""
13: };
14:
15: resource 'DITL' (128, purgeable) {
16: { {151, 179, 171, 247},
17:   Button { enabled, "Next" },
18:
19:   {151, 260, 171, 328},
20:   Button { enabled, "Quit" },
21:
22:   {66, 53, 82, 312},
23:   StaticText { disabled, "" },
24:
25:   {44, 83, 59, 116},
```

→

```
26:   StaticText { disabled, "" },
27:   {44, 173, 59, 212},
28:   StaticText { disabled, "" },
29:   {44, 291, 59, 328},
30:   StaticText { disabled, "" },
31:
32:   {84, 53, 134, 312},
33:   StaticText { disabled, "" },
34:
35:   {48, 8, 80, 40},
36:   UserItem { disabled },
37:
38:   {8, 7, 32, 287},
39:   StaticText { disabled,
40:     "Component Info 1.0\n(c)1996 Maxon
        Computer,"
41:     " Markus Fritze"
42:   },
43:
44:   {36, -1, 37, 500},
45:   Picture { disabled, 128 },
46:   {140, -1, 141, 500},
47:   Picture { disabled, 128 },
48:
49:   {44, 53, 59, 84},
50:   StaticText { disabled, "Type:" },
51:
52:   {44, 128, 60, 172},
53:   StaticText { disabled, "Subtype:" },
54:
55:   {44, 220, 60, 292},
56:   StaticText { disabled, "Manufacturer:" }
57: }
58: };
59:
60: resource 'dctb' (128) {
61:   { wContentColor, 61166, 61166, 61166,
62:     wFrameColor, 0, 0, 0,
63:     wTextColor, 0, 0, 0,
64:     wHiliteColor, 0, 0, 0,
65:     wTitleBarColor, 65535, 65535, 65535
66:   }
67: };
68:
69: resource 'PICT' (128) {
70:   228,
71:   {0, 0, 1, 1},
72:   $"0011 02FF 0C00 FFFF FFFF 0000 0000
      0000"
73:   $"0000 0001 0000 0001 0000 FFFF FFFF
      001E"
74:   $"0001 000A 0000 0000 0001 0001 0013
      0001"
75:   $"AA55 AA55 AA55 AA55 8004 0000 0000
      0008"
76:   $"0008 0000 0000 0000 0000 0048 0000
      0048"
77:   $"0000 0000 0004 0001 0004 0000 0000
      002A"
78:   $"5734 0000 0000 0000 0415 0000 0007
      0000"
79:   $"80E8 80E8 80E8 0001 7D00 7D00 7D00
      0002"
80:   $"0000 64AF 11B0 0003 0000 0000 D400
      0004"
81:   $"0241 AB54 EAFF 0005 F2D7 0856 84EC
      0006"
82:   $"DD6B 08C2 06A2 0007 FC00 F37D 052F
      1010"
83:   $"1010 0101 0101 1010 1010 0101 0101
      1010"
84:   $"1010 0101 0101 1010 1010 0101 0101
      0030"
85:   $"0000 0000 0001 0001 00FF"
86: };
```