Inside Components

Nachdem ich im letzten Teil etwas über die Nutzung von Components geschrieben habe, will ich nun etwas über den Aufbau von Components und die Entwicklung schreiben.

2

Components können 68k- und PowerPC-Code enthalten. Sie werden durch einen Namens-String, einen Info-String und ein Icon näher beschrieben. All dies wird durch die *thng*-Resource definiert. Diese Resource definiert man am besten als Rez-Script, man kann sie aber auch mit einem Resource-Editor, wie dem Resorcerer, erstellen.

Der Aufbau von Components ist stets gleich, wenn auch bei PowerPC-Components etwas aufwendig. Wer Näheres über PowerPC-Components wissen will, dem lege ich die Technote 1004: „QuickTime Component Manager 3.0 & PowerPC Native Components" ans Herz.

Der Programmcode einer Component hat einen allgemeinen Einsprungpunkt, welcher der *main()*-Funktion in C-Programmen entspricht. Übergeben wird eine Struktur mit den Parametern – *ComponentParameters* – und ein Handle, welches es uns ermöglicht, globale Daten der Component abzulegen. Man könnte auf die Idee kommen, globale Variablen ganz normal zu definieren, aber dies wäre bei Components eine schlechte Idee: Components können gleichzeitig von mehreren Programmen genutzt werden und brauchen somit auch mehrere Sätze von globalen Variablen.

Alle Components müssen einen bestimmten Satz an internen Funktionen kennen und darauf reagieren. Andere interne Funktionen sind optional. Natürlich kann eine Component nahezu beliebig viele eigene Funktionen enthalten – dafür machen wir uns schließlich die Mühe der Programmierung ... Die Funktion wird der Component als short Integer übergeben. Negative Zahlen beschreiben Component-Manager-interne Funktionen, positive Zahlen stehen für eigene Funktionen. Hierbei sollten die Funktionsnummern bis 255 für alle Components eines Types auch gleich genutzt werden, damit z.B. Packer für Bilder alle ein gleiches API haben und problemlos austauschbar sind. Die restlichen Nummern sind für jede Component frei nutzbar. Die Zuordnung von Funktionsnummern zu Funktionsnamen wird im Headerfile zur Component gemacht, so daß man als Nutzer der Component nach dem Öffnen der Component ganz „normale" Funktionsaufrufe zur Component machen kann. Einzig die ComponentInstance muß man als ersten Parameter übergeben.

Zu den nötigen internen Funktionen gehören neben *Open* und *Close* noch *CanDo* und *Version*. *Version* gibt lediglich die Versionsnummer der Component zurück. Der Component-Manager wählt nämlich automatisch die neuste Version einer Component. Dazu muß er natürlich die Version einer Component abfragen können. Mittels *CanDo* kann man abfragen, ob eine Component eine bestimmte Funktion kennt. Diese ermöglicht es, das Vorhandensein einer Funktion abzuprüfen, ohne sie aufzurufen. *Open* und *Close* werden beim Öffnen bzw. Schließen der Component aufgerufen. Üblicherweise wird man bei *Open* Speicherplatz für globale Variablen allozieren und sonstige Initialisierungen vornehmen. Falls die Component ein Hardware-Treiber ist, so sollte man sie hier nicht initialisieren, sondern erst beim direkten ersten Zugriff. Immer daran denken, daß die Component durch mehrere Programme gleichzeitig geöffnet werden kann!

Zu den optionalen internen Funktionen gehören *Register* und *Unregister*. *Register* wird beim Anmelden der Component im System bzw. im Anwenderprogramm aufgerufen. *Unregister* beim Abmelden. Dies kann übrigens auch automatisch passieren, wenn der Component-Manager eine neuere Version der Component entdeckt. Ferner gibt es noch den *TargetSelect*-Aufruf, welcher nötig ist, wenn man als Component die Funktionen einer anderen Component ersetzt.

Unser kleines Beispiel ist eine wirklich minimale Component, welche lediglich n SysBeeps ausgeben kann. Zur Demonstration sollte es aber reichen. Ein sehr gutes Beispiel für eine Component ist das bekannte „Internet Config". Es ist sogar Freeware, und man kann alle Sourcen dazu erhalten! Der Autor von „Internet Config"

hat auch einen sehr netten Artikel in der develop – Ausgabe 23 – geschrieben, den ich jedermann nur empfehlen kann.

Der Component-Manager wird übrigens in System 8 zumindest in QuickTime und dem Sound Manager weiterbestehen, auch wenn der Code-Fragment-Manager ihn in vielen Anwendungsfällen ersetzen kann. Momentan muß der Code-Fragment-Manager für 68k-Macs zudem speziell installiert werden, wohingegen QuickTime fast zwangsweise auf jedem Mac vorhanden ist. Deswegen lohnt es sich auch heute noch, kleinere Dinge mit dem Component-Manager auszulagern – zumal eine spätere Anpassung an den CFM sehr einfach ist.

MFR

Literatur:

develop 7, 12, 23, Apple Computer (1/4jährliche Zeitschrift von Apple)

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 Entwickler-Supports 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!)

```
/***
 *  Eine minimale Component
 *
 *  (c)1996 Maxon Computer, Markus Fritze
 ***/
#include <A4Stuff.h>

pascal ComponentResult main(
          ComponentParameters *iParams,
          Handle ioStorage);

/***
 *  Welche Funktionen kennen wir?
 ***/
static pascal ComponentResult
      MyCanDo(short iSelector)
{
 switch(iSelector) {
 case kComponentOpenSelect:
 case kComponentCloseSelect:
 case kComponentCanDoSelect:
 case kComponentVersionSelect:
 case 1:
    return true;
 }
 return false;
}

/***
 *  Unsere Aktion:
 *
 *  n SysBeeps ausgeben
 ***/
static pascal ComponentResult
      MyAction(short iCount)
{
 while(iCount-- > 0)
  SysBeep(10);
}

/***
 *  Unsere minimale Component
 ***/
pascal ComponentResult main(
          ComponentParameters *iParams,
          Handle ioStorage)
{
ComponentResult oErr = noErr;

 EnterCodeResource();

 // interne Funktion des Component Managers?
 if(iParams->what < 0) {

  switch(iParams->what) {
  // bei Open und Close haben wir nichts zu tun
  case kComponentOpenSelect:
  case kComponentCloseSelect:
   break;

  // Welche Funktionen supporten wir?
  case kComponentCanDoSelect:
   oErr = CallComponentFunction(iParams,
            ComponentRoutine(MyCanDo));
   break;

  // und unsere Versionsnummer
  case kComponentVersionSelect:
   return 0x00010000L;

  // der Rest ist Schweigen...
  default:
   oErr = badComponentSelector;
  }

 } else {

  if(iParams->what == 1) {
   oErr = CallComponentFunctionWithStorage(ioStorage,
            iParams, ComponentRoutine(MyAction));
  } else {
   oErr = badComponentSelector;
  }

 }

 ExitCodeResource();

 return oErr;
}


/***
 *  unser Rez-File (68k only)
 *
 *  (c)1996 Maxon Computer, Markus Fritze
 ***/

resource 'thng' (128, locked) {
 'FUN!',        /* Typ */
 'BEEP',        /* SubTyp */
 '????',        /* Creator = Manufacturer */
 $00000000,     /* Flags */
 $00000000,     /* Flagmaske */
 'Code', 128,   /* 68k Code-Resource der Component */
 'STR ', 128,   /* Name der Component */
 'STR ', 129,   /* Infozeile der Component */
 'ICON', 128,   /* Icon der Component */
};
```