

Palmpilot und Handspring

Anwendungsprogrammierung

Am Beispiel
„Schiffeversenken“

Andreas Scholz

Übersicht

- IDE
- Anwendungsstart
- Benutzerschnittstelle
- Nachrichtenbehandlung
- Speicher- und Ressourcenmanagement
- Stringmanipulation
- Zusammenfassung

IDE – Vor- und Nachteile

- IDE: PRC-Tools, PiIRC, SDK 3.5, Visual C++
- Vorteil
 - Kostenlos (bis auf Visual C++)
 - Unterstützung von Linux und Windows
- Nachteile
 - Installation
 - Probleme mit PiRC>Edit
 - teilweise Bugs bei Icons, IDs, Bitmaps
 - löscht ungefragt aus Ressourcendateien
 - Kein direktes Starten des Emulators
 - pdf-Hilfe umständlich

Übersicht

- IDE
- Anwendungsstart
- Benutzerschnittstelle
- Nachrichtenbehandlung
- Speicher- und Ressourcenmanagement
- Stringmanipulation
- Zusammenfassung

Anwendungsstart - Übersicht

Startkommando

Parameter

Flags

```

    Uint32 PilotMain (Uint16 cmd, Ptr cmdPBP, Uint16 launchFlags)
    {
        ...
        if (cmd == sysAppLaunchCmdNormalLaunch)
        {
            error = StartApplication();           // Initialisierung
            ...
            EventLoop();                         //Nachrichtenschleife
            StopApplication ();                  // Aufräumen
        }
        ...
    }

```

Anwendungsstart - PilotMain()

- **Startkommando**
 - sysAppLaunchCmdNormalLaunch (normaler Start)
 - sysAppLaunchCmdFind (Textsuche)
 - sysAppLaunchCmdSyncNotify (HotSync fertig)
 - sysAppLaunchCmdAddRecord (Eintrag in Datenbank einfügen)
- **Parameter**
 - Zeiger auf einen Record mit Parametern
z.B. für sysAppLaunchCmdAddRecord
- **Flags**
 - sysAppLaunchFlagNewGlobals (neue globale Variablen)
 - sysAppLaunchFlagSubCall (rekursiver Aufruf des Programms)

Übersicht

- IDE
- Anwendungsstart
- Benutzerschnittstelle
 - Forms
 - Controls
 - Alerts
 - Menüs
- Nachrichtenbehandlung
- Speicher- und Ressourcenmanagement
- Stringmanipulation
- Zusammenfassung

Forms - Übersicht

- Äquivalent zu Fenstern unter anderen OS
- Größe beim Kompilieren fest eingestellt
- Keine geschützten Bereiche wie Titel-, Fußleiste
- Keine Menüleiste (Einblenden auf Knopfdruck)
- Eingeschränkte Tiefenstaffelung von Fenstern
- Wechsel zu neuem Fenster

`void FrmGotoForm (UInt16 formId)` (wird als Programmierrichtlinie von Palm empfohlen damit Speicher gespart wird)

- Fenster über aktuellem Fenster einblenden

`void FrmPopupForm (UInt16 formId)`

`void FrmReturnToForm (UInt16 formId)`

Forms - Ressource

Ressourcenskript:

```
#include "Schiffe.h"  
...  
FORM ID Startform AT (0 0 160 160)  
NOFRAME  
BEGIN  
TITLE "Schiffeversenken 1.0"  
...  
END
```

...
#define Startform 1000
...

ID: Integer Variable, kann durch
#define bestimmt sein

AT: Position X, Position Y,
Größe X, Größe Y

NOFRAME: kein Rahmen
TITLE: Titel

Controls

Controls - Übersicht

- Standard Controls bei Palm vorhanden
- Wenig Platz für Beschriftungen, meist nur ein Wort
- Keine Tab-Stopps
- Layout durch Pixelgenaues hinschieben, nicht durch Lineale, etc
- Zugriff über Index innerhalb des aktuellen Fensters

Controls – Ressource

<CTRL> <TITEL> ID <ID> AT <(Px,Py,Gx,Gy)> FONT <S>

<CTRL>: LABEL, BUTTON, FIELD, ...

<TITEL>: Titel in Anführungszeichen
Ausnahme SCROLLBAR

<ID>: Integer evtl. in Header definiert
AUTOID für keine

<Px,Py,Gx,Gy>: Position (zentriert mit CENTER)
Größe (automatisch mit AUTO)

: Auswahl des Fonts (0,...,7)
0: normal 1: bold
2: large 7: large bold

<S>: je nach Control
(z.B. MAXCHARS 15 für FIELD)
(z.B. VALUE, MIN, MAX für SCROLLBAR)

Controls – Code

Zugriff auf Controls

```
FormPtr ft;  
FieldType* fieldname;  
UInt16 ind;  
Char* name;
```

```
ft=FrmGetFormPtr(Startform);  
ind=FrmGetObjectIndex(ft,StartformName);
```

```
fieldname=(FieldType*)FrmGetObjectPtr(ft,ind);  
name=FldGetTextPtr(fieldname);
```

Zeiger auf Form holen

Index des Control Elements
in der angegebenen Form
holen

Zeiger auf Control holen
Typecast nötig

Zeiger auf Inhalt holen

Alerts - Übersicht

- Äquivalent zu Nachrichtenboxen unter anderen Betriebssystemen
- 4 Typen:
 - Information (I-Icon) - Confirmation (Fragezeichen)
 - Warning (Warndreieck) - Error (X-Icon)
- Beliebige Anzahl von Buttons, aber kein Zeilenumbruch
- Buttons können frei beschriftet werden
- Höhe variiert automatisch je nach Inhalt
- Nachricht kann Platzhalter enthalten (^1,^2,^3)

Alerts - Ressource

Ressourcenskript:

```
ALERT ID AlertWin
INFORMATION
BEGIN
  TITLE "Ende"
  MESSAGE "^1 ^2\nSie haben ^3!"
  BUTTONS "Ende"
END
```

Typ: INFORMATION,
CONFIRMATION, WARNING,
oder ERROR

TITLE: Titel,
eine Zeile, zentriert

MESSAGE: Nachrichtentext,
bis zu 9 Zeilen, Zeilenumbruch
mit "\n"

BUTTONS: Beschriftung der
Buttons, getrennt durch
Leerzeichen

^1, ^2, ^3 Platzhalter
maximal 3

Alerts - Code

- Rückgabewert ist der Index des Buttons der gedrückt wurde, von links mit 0 beginnend

- Anzeigen des Alarms

```
FrmAlert(UInt16 alertId);
```

- Alarm mit Platzhaltern anzeigen

```
FrmCustomAlert(UInt16 alertId, const char *s1, const char *s2,  
               const char *s3);
```

```
Bsp.: FrmCustomAlert(AlertWin,"Schade",name,"verloren");
```

Menüs

- Große Ähnlichkeit zu anderen OS
- Normalerweise erst auf Knopfdruck sichtbar
- Pulldown Menüs nur einfach verschachtelt
- Shortcuts für die einzelnen Befehle
- Trennstriche möglich

```
MENU ID MenueGame
```

```
BEGIN
```

```
  PULLDOWN "Datei"
```

```
  BEGIN
```

```
    MENUITEM "Beenden" ID MenueGameQuit "Q"
```

```
  END
```

```
END
```



Name, ID, Shortcut

Übersicht

- IDE
- Anwendungsstart
- Benutzerschnittstelle
- Nachrichtenbehandlung
 - Nachrichtenschleife
 - Behandlung von Nachrichten
- Speicher- und Ressourcenmanagement
- Stringmanipulation
- Zusammenfassung

Nachrichtenschleife - Übersicht

- Aktive Anwendung erhält alle Nachrichten
- Fertige Handler für Menü- und Systemnachrichten müssen von der Anwendung aufgerufen werden
- Registrierung von Nachrichtenbehandlungsroutinen für Forms möglich
- Bei Verwendung von blockierendem Warten wird die CPU automatisch in doze mode versetzt
- Zum Beenden wird `appStopEvent` an die Anwendung geschickt

Nachrichtenschleife – Standardhandler

Boolean SysHandleEvent (EventPtr eventP);

- Behandelt Nachrichten die Soft- und Hardkeys betreffen
- Sollte immer als erste Funktion aufgerufen werden

Boolean MenuHandleEvent (MenuBarType *menuP, EventType *event, UInt16 *error)

- Behandelt Nachrichten die im Menübereich auftreten
- Schickt menuEvent an die Anwendung wenn ein Befehl ausgewählt wurde

Boolean FrmDispatchEvent (EventType *eventP)

- Leitet Nachrichten an die Fensternachrichtenbehandlung weiter

Nachrichtenschleife – Code

```
EventType event;
```

```
do
```

```
{
```

```
    EvtGetEvent(&event, evtWaitForever);
```

```
    if (SysHandleEvent(&event)) continue;
```

```
    if (MenuHandleEvent((void *)0, &event, &err)) continue;
```

```
    if(event.eType == frmLoadEvent)
```

```
    {
```

```
        ...
```

```
        FrmSetEventHandler( form, (FormEventHandlerPtr) frmMainEventH);
```

```
        ...
```

```
    }
```

```
    FrmDispatchEvent(&event);
```

```
}
```

```
while(event.eType != appStopEvent);
```

Nachricht holen,
blockierendes warten

Standardhandler aufrufen

Neues Fenster geladen?

Nachrichtenbehandlungs-
routine setzen

Nachrichten an die Fenster
weiterleiten

Nachrichten - Datenstruktur

```
typedef struct {  
    eventsEnum eType;  
    Boolean penDown;  
    UInt8 tapCount  
    Int16 screenX  
    Int16 screenY  
    Union{  
        ...  
    } data;  
}EventType;
```

eType: Nachrichtenart
z.B. appStopEvent, penDownEvent,
menuEvent, keyDownEvent

true: Stift gedrückt
false: Stift oben

Anzahl der Tips mit dem Stift

Nachrichtenspezifische Daten
z.B. ID des Menüpunktes,
ID des Control Elementes, etc

Position (X,Y) des Stiftes
als das Ereignis ausgelöst
wurde

Nachrichten – Code

```
Boolean frmMainEventH(EventPtr event)
{...
switch (event->eType)
{
case frmOpenEvent:
...
break;
case ctlSelectEvent:
if (event->data.ctlEnter.controlID == Startformok )
...
break;
case nilEvent:
break;
}
}
```

Welcher Nachrichtentyp ?

Form wird neu geöffnet
Aufruf von Zeichenfunktion, etc

Control wurde aktiviert
Control: `data.ctlEnter`
ID: `data.ctlEnter.controlID`

Wird geschickt wenn timeout
Bei `EvtGetEvent()` abgelaufen ist
z.B. zur Animationssteuerung

Übersicht

- IDE
- Anwendungsstart
- Benutzerschnittstelle
- Nachrichtenbehandlung
- Speicher- und Ressourcenmanagement
 - Dynamischer Speicher, Memory Manager
 - Statischer Speicher, Streams
- Stringmanipulation
- Zusammenfassung

Dynamischer Speicher - Übersicht

- Moveable Chunks
 - Speicherzugriff nur über Handles
 - Handles müssen locked und unlocked werden (Maximal 14 locks gleichzeitig)
 - Höherer Aufwand bei Programmierung
- Non-moveable Chunks
 - Direkter Zugriff
 - Reduzieren verfügbaren Speicher evtl. erheblich
 - Schnellere Programmierung

Dynamischer Speicher - Code

```
UInt8 AllocMemory()
```

```
{
```

```
    UInt8* s;
```

```
    memhandleSpielfeld=MemHandleNew(100);
```

```
    s=(UInt8*)MemHandleLock(memhandleSpielfeld);
```

```
    for(i=0;i<10;i++)
```

```
        for(j=0;j<10;j++)
```

```
            s[i*10+j]=0;
```

```
    MemHandleUnlock(memhandleSpielfeld);
```

```
}
```

```
UInt8 FreeMemory()
```

```
{
```

```
    MemHandleFree(memhandleSpielfeld);
```

```
}
```

Größe des Speicherblocks in Bytes



Zeiger gültig zwischen Lock und Unlock

Speicherfreigabe

Statischer Speicher, Streams

- Keine Verzeichnisstruktur
- Stream Funktion ähnlich zu c
- Öffnen

```
FileHand fh=FileOpen(0,"schiffe",0,0,fileModeReadWrite,NULL);
```

Kartenummer, Name, type, creator, openmode, error

- Schreiben

```
count=FileWrite(fh,&g_NrMoves,1,1,NULL);
```

Handle, Buffer, Elementgröße, Anzahl Elemente, error

- Schliessen

```
FileClose(fh);
```

Übersicht

- IDE
- Anwendungsstart
- Benutzerschnittstelle
- Nachrichtenbehandlung
- Speicher- und Ressourcenmanagement
- Stringmanipulation
- Zusammenfassung

Stringmanipulation

- gcc-Compiler unterstützt nur theoretisch alle c string Funktionen
- Probleme treten bei allen Funktionen auf die Speicher mittels "malloc" etc belegen wollen
- Programm wird durch einbinden von Methoden aus „string.h“ unnötig groß
- Lösung: Stringroutinen werden direkt vom Betriebssystem zur Verfügung gestellt
- Funktionsnamen der API-Funktionen sind nahezu identisch mit denen der c-Funktionen (meist bis auf Groß-, Kleinschreibung)

Übersicht

- IDE
- Anwendungsstart
- Benutzerschnittstelle
- Nachrichtenbehandlung
- Speicher- und Ressourcenmanagement
- Stringmanipulation
- Zusammenfassung

Zusammenfassung - 1

- Sehr beschränkte Möglichkeiten bzgl. Grafik und Schriftausgabe (geringe Auflösung, Farbtiefe)
- Sorgfältiges GUI-Design nötig
- Speicher- / Ressourcenzugriff ungewohnt
- Umständliche Memhandles verleiten zur Verwendung von non-moveable chunks
- Nur sehr wenig statischer und dynamischer Speicher, Datenstrukturen müssen möglichst effizient gewählt werden

Zusammenfassung - 2

- API leicht verständlich, besonders mit Hintergrundwissen
- Leichter Einstieg, da die meisten Funktionen identisch zu Standard c Funktionen sind (I/O, Strings,...)

Quellen

- IDE

<http://www.palmos.com/dev/tech/tools/>

- Installationsanleitung

<http://www.palmos.com/dev/tech/tools/gcc/install-win.html>

- Palm OS Reference (3.5)
- Palm OS Companion (3.5)

Vielen Dank für die
Aufmerksamkeit